Computer Engineering thesis code CE-MS-2017-08

p-VEX on Chip

The Design of an ASIC for a Dynamically Reconfigurable VLIW Processor with 24-port Register File

L. van Bremen

CE-MS-2017-08

TU Delft

ρ-VEX on Chip

The Design of an ASIC for a Dynamically Reconfigurable VLIW Processor with 24-port Register File

by

L. van Bremen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday August 30, 2017 at 10:00 AM.

Abstract

The ρ-VEX is a runtime reconfigurable VLIW processor. It is able to exploit both ILP as well as TLP by running one program in multiple lanes, or several programs concurrently. To accurately quantify its performance compared to other processors, it is implemented as an IC.

A fully automatic scripted flow is described, constructing an optimized, error-free ASIC. The design area is limited to $3.5\,\mu m^2$, to allow it to be manufactured with the cheapest $65\,nm$ prototyping run available at IMEC. In order to achieve this small design area, a new 24-port register file design has to be devised. In its current state, it is implemented using standard logic cells. Clever optimizations involving cell padding and minimum overhead power routing are necessary to reduce a total routing overcongestion from 7% to zero.

This implementation is expected achieve a clock speed of $141\,MHz$, only limited by the ρ-VEX reconfiguration logic. By lowering the frequency to $100\,MHz$, the energy dissipation is reduced by 96%, to a total of $367\,\mu W\,MHz^{-1}$. This is comparable to similar VLIW designs. Using LVDS communication clocked at four times the core frequency, a data bandwidth of $4\,Gbit\,s^{-1}$ is achieved using only 26 external pins. This allows the main data and instruction memory to reside off-chip.

To ensure a correct design, it is verified using DRC, LVS, and ERC. For improved reliability, IR-drop is kept within 1% of the core voltage, using minimal power routing. Furthermore, the effect of electromigration is kept extremely low such that the mean time to failure on nets is 90 years.

The scripted flow will aid in prototyping, allowing accurate estimates to be calculated in a mere 7% of the original time. Proposals for a future design are expected to reduce the register file area and power dissipation by more than 40%. By improving the pipeline and shortening the critical path, a two to four fold improvement in clock speed can be expected, without adversely affecting power dissipation.

Thesis code:	CE-MS-2017-08	
Student number:	4143620	
Project duration:	August 26, 2016 – August 30, 2017	
Thesis committee:	Dr. ir. J.S.S.M. Wong,	TU Delft, CE, supervisor
	Dr. ir. T.G.R.M. van Leuken,	TU Delft, CAS
	Dr. ir. Z. Al-Ars,	TU Delft, CE

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

Thanks to my friends, family, colleagues and supervisors, for their continued support.

Special thanks to Mark Willmot at the STFC and Erwin Deumens at IMEC, for their patience and responses to various questions otherwise left unanswered.

And above all, thanks to my loving parents, for both their mental as well as financial support throughout all my years of education.

L. van Bremen
Delft, August, 2017

List of Figures

List of Tables

Contents

1 Introduction

1.1. Context

Over the past decade, the development focus of (micro)processors has shifted. Before the approximate mark of 3 GHz, processor developments have been mostly towards improving performance. The speed of these Integrated Circuits (ICs) has been doubling every 18 months since the 70's. Around 2008 heat dissipation became a major problem for increasing processor frequencies, thereby guiding major processor manufacturers towards different means of improving performance. Multi-core processors become popular as they allow an easy duplication of existing hardware while apparently doubling or quadrupling performance numbers. The downside, however, is also a doubling or quadrupling of power consumption, raising the power bar even higher.

While the consumer processors are led by marketing numbers and performance benchmarks, the enterprise products and multi-purpose Digital Signal Processors (DSPs) are designed with a different mindset. These products, including the recent market of mobile phones, primarily focus on reducing power consumption. The specific important metric is the performance per Watt, how much power does it consume to perform a certain task? Enterprise processors based on the PowerPC and SPARC architectures implement a Reduced Instruction Set Architecture (RISC) as opposed to a Complex Instruction Set Architecture (CISC). The simpler instructions allow for a smaller and faster processor design which again allows for improved performance per Watt.

1.2. Parallelism

As frequency scaling has very much slowed down and power consumption scales poorly with high frequencies (more details in Chapter 3), there was a need for different performance gains. Hence, parallelism was exploited. In any program or design, two types of parallelism can be differentiated: thread-level parallelism and instruction level parallelism.

1.2.1. Thread-Level Parallelism

Thread-Level Parallelism (TLP) is parts of a task or program that can be split in subtasks which can be executed independently. Only the data inputs and data outputs might need sharing. A very simple example of TLP is running two entirely different programs on the same processor. Instead of executing them sequentially, they might be run in parallel.

A problem with TLP is that it needs to be inherent in a program. Usually it is relatively costly in terms of cycle count to start a new thread, copy over the input data and finally copy back the result. As this overhead is incurred, exploiting TLP is only beneficial for larger chunks of a program. Consequently, only the programmer and to a limited degree the compiler can predict when to split up a program in threads.

Multi-core processors are the most obvious way of exploiting TLP in hardware. Each core can run a thread separately from the other cores.

1.2.2. Instruction Level Parallelism

Instruction Level Parallelism (ILP) is the art of detecting instructions (or even parts of instructions) that don't influence each other directly. Again the prerequisite for two instructions to be run in parallel is that the inputs and result of an instruction do not depend on the parallel instruction. As instructions typically have only one result and often not more than two inputs, ILP is way more common than TLP. Furthermore, it is easier to overcome the extra overhead as required data is local and only very small.

A superscalar processor fetches multiple instructions at once, inspects them for ILP and, if possible, executes them in parallel. This happens at run time and requires little to no effort of the programmer or compiler. Almost all modern processors are superscalar. However, the same discussion can be held as to whether it is beneficial to move from RISC to CISC. Although it seems smart to always allow the processor to exploit this parallelism, the incurred cost is some speed, extra silicon area for the prediction and parallel execution logic and most notably, an increase in power consumption by the added hardware.

1.2.3. VLIW architecture

In contrast to superscalar processors, a processor based on the Very Long Instruction Word (VLIW) architecture is much simpler. VLIW processors also exploit ILP, however this is achieved by creating a smart compiler, as opposed to a complicated processor. Before program execution, the compiler is instructed to detect any ILP in a program. Subsequently, it places multiple instructions that can be run in parallel in a single instruction word for the processor to execute. The compiler has plenty of time and resources available and can 'see' the whole program, as opposed to a superscalar processor, which can see only as much as the hardware allows. Therefore, the compiler can better exploit ILP. On top of that, the program has to be compiled only once. Subsequently, it can be run as many times as desired, without the need to detect ILP over and over again. All the VLIW processor needs is the hardware to run the designated instructions in parallel. This leads to a much simpler and smaller design that can be faster and more power efficient.

1.3. The ρ-VEX reconfigurable VLIW processor

The ρ-VEX is a processor based on this VLIW methodology of keeping the core as simple as possible and pre-computing all complex tasks in the compiler. More specifically, it is based on the VLIW Example (VEX) suite. VEX includes an Instruction Set Architecture (ISA), a compiler and a simulation environment.

The ρ-VEX is designed such that it can run eight instructions in parallel in its pipelanes. Furthermore, the ρ-VEX allows for reconfiguration of these pipelanes. Instead of using all eight for a single program, it can use four for one program and four for another. All possible modes of configuration include 1x8, 2x4, 4x2 and 1x4+2x2. These modes allow for the processor to exploit TLP as well as ILP dynamically. Either by running one program in a large 1x8 mode, or by running up to four programs in the smaller 4x2 mode. Note that in 4x2 mode each program will have two pipelanes available, allowing for only two instructions to be run in parallel.

Allowing the processor to run up to eight instructions in parallel puts constraints on the processors register file. In its current architecture, any instruction can have two input values and one result value. This means that every single clock cycle, up to 16 inputs might be read and up to 8 results written to and from the register file. Therefore, the implementation as described in this document must include a 24-port register file implementation running at speed of the rest of the core. In Section 7.5 it will become clear why this is a problem.

Furthermore, it is important to note that there is a large design time configuration flexibility allowed for the ρ-VEX. Recompiling the design allows for a different configuration that might be smaller, faster or more power efficient. The different design options are described in Chapter 6.

More information about the ρ-VEX can be found on the website and, therefore, will not be repeated here. It is mostly important to notice that a primary goal of the ρ-VEX project is delivering a dynamic trade-off between performance and power. Therefore, any design should be created with this in mind.

1.4. Problem definition

As of the start of this research project, the ρ-VEX is written purely in VHDL. It can be compiled and run on a Field Programmable Gate Array (FPGA). As an FPGA is re-programmable, clear advantages of an FPGA include quick iterations, low development cost, and easy testing and debugging. However, the FPGA fabric is very area inefficient and the ρ-VEX has only been tested on development boards that include extra peripherals. As the goal of the ρ-VEX includes to be a simple, small, and low-power processor, there is a need to compare these metrics with comparable processor designs. The FPGA overhead is interfering with acquiring accurate area numbers and real power consumption measurements.

Therefore, there is a need for an Application Specific Integrated Circuit (ASIC). This ASIC will include the current state of the ρ-VEX core, with the necessary register file. No unnecessary features are added, such that the area and power of the ρ-VEX core can be quantified. The ρ-VEX is designed to be flexible and design time re-configurable. As an ASIC will simply be a single snapshot, not allowing reprogramming, a specific configuration has to be chosen.

As will be explained in Section 5.1, there is not enough space for the full data and instruction memories on-chip. Therefore, only a small cache is included with a High Speed Interface (HSI) connecting this cache with larger off-chip memory. The HSI is a separate project, but developed in conjunction with this project. Some silicon area is reserved for this HSI and it will affect the design choices.

Furthermore, the ρ-VEX is planned to go into space for future project work. Space missions usually take several years up to several decades. As repairs in space are nearly impossible, a future re-spin must reliably operate for the duration of such a mission. In space, power is also a limited resource. Therefore, it is extra important to achieve a design that consumes as little power as possible.

From these design problems the following research question was composed and will serve as guideline for this document.

“How to design and implement a reliable, power-optimized Application Specific Integrated Circuit prototype of the ρ-VEX core with external memory?”

1.5. Design constraints

In general, ASIC design is a very costly procedure. Designing a chip and setting up a foundry are the main, one time cost. Therefore, the costs are usually accounted for by producing and selling large volumes of the same chip. To also enable universities and small research institutes to prototype an IC, without the large incurred costs, Europractice* was founded. Through European funding, expensive software tools and small, cheap prototyping chips are possible.

This project will make use of the software available through Europractice, as installed on the universities servers. Therefore, the choice for tools and technology are limited. The following constraints are set prior to this project. Each constraint is preceded by **C**., indicating a design constraint.

C.1 All used software must be available through Europractice.

C.2 The ASIC will be fabricated by IMEC using their mini@sic Multi-Project Wafer (MPW) run. This allows for a cheap prototype, see Section 2.2.

C.3 The used process is *UMC L65N 1P10M lowK Logic/MixedMode LL*, see Section 2.1.1.

C.4 The only available metal stack is *1P8M1T0F1U*, see Section 2.1.4.

C.5 The chip area is $1875\,\mu m \cdot 1875\,\mu m$. A 'double width' chip, with twice the area, may be used. However, due to the doubling of the costs, this should be avoided.

1.6. Design goals and methodology

Following the problem definition, the design goals can be defined. These design goals clearly state the essential requirements of the project and will be a further guideline to this work. Any design choice made during the project and described in this document will refer back to one of these goals. The references are preceded by **G**., indicating a primary design goal is considered.

G.1 Construct a first iteration ASIC prototype of the ρ-VEX

G.2 Design and implement an interface for off-chip memory

G.3 Explore the design space as much as possible

- (a) How should the ρ-VEX be configured during implementation?
- (b) What is the smallest possible ρ-VEX ASIC prototype?
- (c) What are important design improvements?
- (d) What can an ASIC achieve with these improvements?

G.4 Ensure a useful final design

- (a) Verify the design in every way possible to prevent mistakes
- (b) If the manufactured ASIC does contain a mistake, make sure it can be detected and corrected

G.5 Design with future space applications in mind

- (a) The ASIC must be able to reliably keep running for years
- (b) The ASIC must consume as little power as possible

G.6 Every design step must be well documented and reproducible for future improvements

*europractice-ic.com

1.7. Thesis overview

This document has two main goals for the reader. First of all, as the design is a prototype and it is expected to be improved upon, this document can act as a user guide for the inexperienced reader. All knowledge gained throughout the project is very verbosely explained in Chapters 2 to 4. These chapters may be skipped by anyone who is already knowledgeable in the field of study. The second goal of this project is the actual design and implementation. Chapters 5 to 9 explain all problems encountered during the project and explain every decision made. This second half is the actual technical part of this document.

Following this introductory chapter, all required background for understanding all design parameters and choices and the parlance used throughout the rest of the document. The reader is recommended to at least quickly look at Chapter 2. The rest of the document will expect the knowledge described in this chapter as known to the reader and might be confusing when some definitions are not known. Note that in addition to this verbose background chapter, there is also a glossary describing almost every term or abbreviation. If the online version of this document is used, every glossary term is a hyperlink to its definition. Of the general background, in particular Sections 2.1.1 and 2.1.4 are useful even for the experienced reader. These describe the used process in slightly more detail and determine the capabilities of the IC.

In addition to the general background, more in-depth background information is given on power consumption in Chapter 3. This chapter is separate due to its explicit importance for the document. Most of the design choices regarding low power design will find their roots in this chapter and it will be frequently referenced. The reader is encouraged to understand its content well.

The goal of Chapter 4 is partially background but targeted specifically on this project. It contains an overview of the used software in several design flow steps. It starts off with a flow diagram of all the steps required for converting any HDL design like the ρ-VEX into an ASIC. The used steps are subsequently described in more detail in the rest of the chapter. This chapter is mainly concerned with (**G**.1) and (**G**.6).

In Chapter 5, the interface of (**G**.2) connecting the external memory is globally described. As the actual interface is implemented in a separate project, only the requirements and results are repeated. Any extra requirements imposed by this interface on the rest of the project are described as well.

Chapters 6 to 8 will focus on Design Space Exploration. (**G**.3) is extensively dealt with. In particular, Section 7.5 touches upon a main problem with the ρ-VEX, where no single perfect solution is possible. Chapter 8 describes the actual final implementation of the ASIC and all problems tackled during implementation. When reproducing or expanding on this work, this chapter might be the most significant. When every choice in this chapter is fully understood, wasting time on similar problems in future projects can be avoided.

To ensure the reliability required for (**G**.5a), Chapter 9 will describe every used tool towards a robust design. Every single one of these tools has been developed and described such that they can be used in future re-spins and ensure the same reliability.

In Chapter 10, all final design characteristics are listed. The design is compared to similar chips and the achieved results are verified.

Finally, Chapter 11 concludes the document with possible future improvements, recommendations for similar designs and a short recap of the whole document.

2

General background

In this chapter, the basic knowledge required to understand the document is described. In particular, this chapter starts with describing the steps required for fabrication of the final ASIC. Fabrication is split in the accessed foundry, Section 2.1, and the used IC services, Section 2.2. These fabrication steps limit what can possibly be implemented on the ASIC.

Next, the chapter continues on by describing different metrics for reliability. As a future re-spin of the design is intended to go to space (**G**.5), the design has to be able to reliably operate for multiple years. Furthermore, functional changes like fault tolerant design [1], memory redundancy [2] and error correction [3][4] are required. The latter changes require adapting the functional code and result in a higher cost chip, conflicting with the prototyping requirement (**G**.1). Even though the design as described in this document will not be entirely ready for a space mission, as many steps as possible towards the required reliability will taken in Section 9.5. The background for these processes is introduced in Section 2.3.

As power consumption is a more prominent topic of this document, it is split off in Chapter 3.

2.1. Foundry

IC fabrication is a very delicate and complicated process. Going down to a feature size of just a few nano-metre, the machines required for fabrication become more and more expensive. Furthermore, these machines can only operate in a clean room, as even small dust particles can destroy the chip.

A foundry is specialized in IC fabrication. They have the required machines and knowledge to achieve an acceptable yield (percentage of devices that behave correctly after fabrication). Usually, it takes several years to design a new, smaller process and improve the yield up to the point where the improved process becomes profitable. The knowledge gained in these years are boiled down to a set of rules and components, which are subsequently compiled in the Topological Layout Rules (TLR) and the foundry Design Kit (FDK).

2.1.1. Foundry process

The foundry process determines which process will be used for fabricating the IC. Main contributors to choosing a specific design process is maturity, availability, cost, and performance. Using a more advanced, less mature process will result in reduced reliability due to unknown aging factors. This is unacceptable for space applications, as post launch repairs are very costly and neigh on impossible. A low availability of the process causes fabrication to be postponed until the foundry is able to accept more designs. This might cause a re-spin to take way longer than expected, something that is not desirable for prototyping. Similarly, high manufacturing costs are undesirable for prototyping. Lastly, as will be explained in Chapter 3, there is a performance trade-off between operating speed and power consumption. Choosing a foundry process influences the achievable performance metrics.

For this project, the used foundry process is called *UMC L65N 1P10M lowK Logic/MixedMode LL* (**C**.3). In short, this means the following:

1. UMC, the accessed foundry is United Microelectronics Corporation.
2. L65N, the length of a gate is $65\,nm$, better known as the feature size.
3. 1P10M, the metal stack consists of 1 poly layer and up to 10 metal layers, more will be explained in Section 2.1.4.
4. lowK, referring to the material constant κ of insulating layers being 'low'. This usually positively affects both speed and power consumption.
5. Logic, for this process libraries are available containing standardized logic gates for developing digital circuits.
6. MixedMode, the foundry process allows for custom drawing of transistors, which might be useful for digital and is required for analog circuits. Here mixed refers to the inclusion of both analog and digital circuitry.
7. LL, the foundry process is optimized for Low Leakage circuits.

This process results in several important aspects. The availability is good (about four runs per year), both digital libraries as well as analog devices are included (which will be important in Chapter 7) and this design requires the power optimized process characteristics.

2.1.2. Topological Layout Rules (TLR)

The TLR, also known as design rules define the set of rules imposed on any design. These rules determine what the foundry can and can't fabricate. If a design does not follow the rules, the foundry can't guarantee functionality of the IC and, therefore, will not accept the design submission. The TLR is checked in the Design Rule Check (DRC), which is just an automated check programmed with all rules. In 9.5.1, this DRC is performed.

Examples of TLR include a minimal distance between two wires to prevent a short circuit or the minimum size of a device down to which it can be fabricated. The more mature the process, the more extensive these rules become, as previous fabrications might have shown defects in very specific cases.

2.1.3. Foundry Design Kit (FDK)

An FDK is simply a set of libraries, rules and configurations to properly design an IC in a specific software suite. It is important to check which software is supported by the FDK as insufficient knowledge in operating the software will result in an inferior design. More information about the used software is described in Chapter 4. For this project a course was taken to improve the knowledge in (part) of the used software.

An FDK is usually split in two parts. These two parts are called Front end of Line (FEOL) and Back end of Line (BEOL). The FEOL is concerned with drawing, sizing and characterizing individual transistors. The BEOL is concerned with efficiently connecting all transistors to create the functionality of the IC.

2.1.4. Metal stack

To connect every transistor on an IC, metal wires are used. When two wires cross, one wire has to go over top the other wire. To slightly simplify the manufacturing process, all wires will run at predefined metal layers. Two consecutive metal layers can be connected using a via. For a simple design, it might be sufficient to use one or two metal layers. For example, the Intel 4004, produced in 1971 was fabricated using a process allowing only two metal layers [5]. More recent technologies can go up to eleven metal layers.

For this design, the metal stack described as *1P8M1T0F1U* (**C**.4) is available. The described layers are:

1. 1P, a single poly layer, used to construct the transistor gates.
2. 8M, eight metal layers, of which:
   (a) 1T: One thick metal layer (twice the regular thickness).
   (b) 0F: Zero fat metal layers (four times the regular thickness).
   (c) 1U: One ultra-thick metal layer (eight times the regular thickness).

The thick, fat and ultra-thick metal layers have different properties regarding sheet resistance and fringe capacitance. They might be beneficial for specific functions in the design, but less beneficial for other functions. More details are in Chapter 8. All parameters regarding these metal layers can be found in the Electrical Design Rules [6] (EDR), as part of the FDK.

2.1.5. Transistor types

Many different parameters can influence the performance of a transistor. For example, the width and length of a transistor can be increased to increase the drive strength and resistance respectively. The width and length can vary from transistor to transistor, all on the same chip.

On the other hand, a different foundry process can change characteristics of all transistors on a wafer. In Eqs. (2.1) and (2.2), the threshold voltage V_{th} is given. V_0 is an intrinsic parameter and V_{SB} is usually fixed. In this particular process, the gate oxide thickness t_{ox} is increased to reduce Fowler-Nordheim tunnelling, achieving low-leakage devices.

Furthermore, the foundry process supports three different transistor types. In [6], these are given as ‘LL_RVT’, ‘LL_HVT’ and ‘LL_LVT’. specifically, these are Low Leakage devices with respectively Regular Voltage Threshold (RVT), High Voltage Threshold (HVT) and Low Voltage Threshold (LVT). From the datasheet t_{ox} is equal for the three devices. The oxide thickness is easy to vary for the whole wafer (resulting in Low Leakage), but more difficult to control per device. Instead, the channel doping N_A in Eq. (2.2) is varied. This can very precisely be controlled per device by including an extra step in the process. This will result in an extra wafer mask, significantly increasing the cost. However, because the used IC service (Section 2.2) already combines multiple projects on one wafer, all devices are available for the same cost. This is very useful for prototyping.

$$V_{th} = V_0 + \gamma\left(\sqrt{|V_{SB} + 2\phi_F|} - \sqrt{|2\phi_F|}\right) \quad \text{Equation (2.1)}$$

$$\gamma = \frac{t_{ox}}{\epsilon_{ox}}\sqrt{2q\epsilon_{si}N_A} \quad \text{Equation (2.2)}$$

2.2. IC services

Fabricating an IC consists of many different steps. These steps are outside the scope of this document and, therefore, they are not repeated here. However, as UMC will only perform part of these, the fabrication will involve more than just sending over a design. This is where an IC service comes in. The IC service will take care of all final design checks and ensures the intended design is correctly fabricated by UMC.

For this design, IMEC is the chosen IC service. IMEC is the official service for UMC projects. On top of that, they enable a Multi Project Wafer MPW run.

2.2.1. Multi Project Wafer

The largest cost for an IC are setup costs for a mask set. Masks for a $65\,nm$ process can cost up to a million dollars. Only when tens of millions of a chip produced with one such mask sets are sold will it actually become affordable. For this project only a few prototypes are required, for which a million dollars will be too costly.

As the name suggests, an MPW simply combines multiple projects on a single wafer. This allows for sharing the costs of a mask set with all the different projects. A disadvantage of this is the actual design size. As multiple projects have to fit the same wafer, each project can only be a fraction of the original size. On the other hand, the costs can be reduced up to a hundred times. Furthermore, as different projects might use different devices, all production process steps always have to be performed. This allows using all the different devices without additional costs. Again making it ideal for prototyping.

2.3. Reliability

The reliability of a device is mainly concerned with making sure the devices don't fail to operate. Where a broken phone is a minor mishap better to be avoided, a failure of any space oriented device is catastrophic. It is simply unfeasible to return a satellite from space for failure analysis.

Reliability can be roughly split in two parts. Manufacturing and operation reliability. Improving manufacturing reliability leads to an increased yield. As already discussed in Section 2.1.2, the yield can be improved in fabrication itself. On top of that, in Section 2.3.1, an extra yield improvement is described, which will be applied in Section 9.4.2. Operation reliability is a function of the device failure rate.

A very common device failure rate is given by the 'bathtub distribution' [7][8], see Fig. 2.1. For space applications, the failure rate at any point in time should be kept as low as possible. Infant mortality can be overcome by long term device testing. If all failing devices are detected up to t_{infant}, it can be ensured no failing device is placed in the satellite. The constant failure rate, as its name suggests, is constant and can mostly be influenced during fabrication or redundant design, both outside the scope of this document. Any device has a wear out phase and this can't be overcome. The main concern here is to prolong the useful lifetime, $(t_{operation} - t_{infant})$, as much as possible.

Many wear out failure mechanisms are described in many different projects. A list of common mechanisms is given in [9]. The main contributor which can also be influenced purely in IC design is electromigration (EM). This effect is described in more detail in Section 2.3.2.

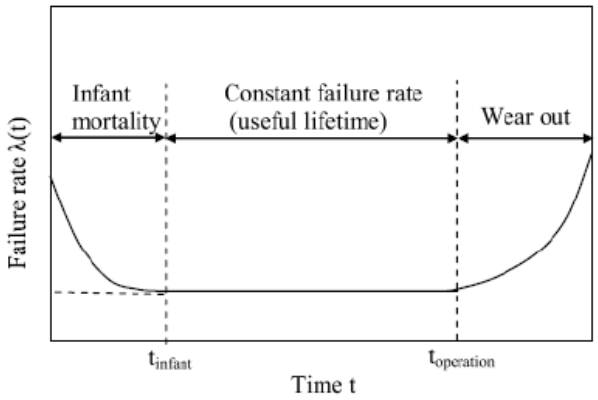


Figure 2.1: Bathtub distribution for a typical device failure rate [10].

2.3.1. IR-drop

During IC design, every transistor will be (in)directly connected to a supply line. This can either be power (VDD) or ground (VSS). For this particular foundry process, VDD is expected to be $1.2\,V$ (when VSS is $0\,V$). Increasing or decreasing this voltage will increase or decrease the current through a transistor. This will respectively speed up and slow down a circuit. Furthermore, if the voltage becomes too high, a short circuit spark might break the transistor. On the other hand, if the voltage is too low, the voltage threshold might not be reached, causing the transistor to never start working at all.

Clearly, in an ideal chip, VDD would be $1.2\,V$ across the whole chip. This voltage is distributed across the chip using metal wires, as described in Section 4.2.2. As these wires have a non-zero resistance and a current will flow through the wire every time a capacitance has to be charged, a voltage will drop across the wire. This drop is called IR-drop and causes VDD to be slightly lower than $1.2\,V$.

Similarly, when a capacitance has to be discharged, a current will flow through the ground wires causing a voltage drop as well. This means the VSS will not be an ideal $0\,V$, but slightly higher instead. This phenomenon is called ground bounce and usually taken as part of IR-drop.

As already stated, IR-drop will influence the speed of devices. Therefore, any IC is designed with a non-ideal voltage source in mind. Usually, the worst case variation of 10% is taken. But when the voltage varies more than this, the functionality might still fail or transistors might break. Therefore, Sections 4.2.2 and 9.4.2 verify the total IR-drop in the design is within the stated 10% limit.

2.3.2. Electromigration

EM is a failure mechanism that has already been a problem for over five decades [11]. In short, EM is the movement of free metal ions resulting from a transfer of momentum from the conducting electrons [12]. Metal ions become free due to thermal fluctuations in the IC. The more electrons bumping in a metal ion, the larger the EM. Therefore, EM is mainly induced by current density. When too many metal ions are moved out of place, the wire is considered 'broken'. No more current can flow through the wire causing IC failure. See Fig. 2.2 for a SEM image of a wire broken due to EM.

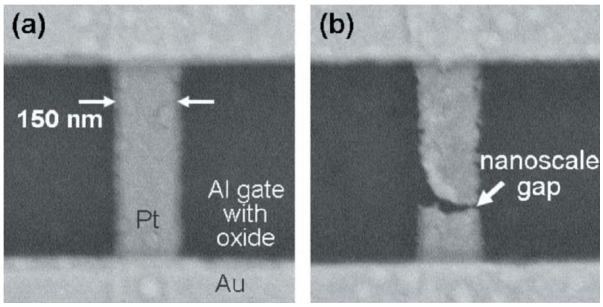


Figure 2.2: Broken wire due to electromigration [13].

To mitigate EM means to lower the current density in a wire. The current density J through a wire is defined by Eq. (2.3), with, I_{max} the maximum total current through a wire and W and H the wire width and height. To reduce the current density, either the maximum current can be reduced or the width or height of the wire can be increased. The height of a wire depends on the manufacturing process and, therefore, can not be influenced. What can be done is adding a second wire, such that the total current is split between the two. Otherwise, the wire width W can be increased.

$$J_{max} = \frac{I_{max}}{W \cdot H} \qquad \text{Equation (2.3)}$$

The total effect of EM on the functionality of a chip depends on the process and parameters vary with different foundries. To limit the amount of failures induced by EM, the foundry will give a limit on the current through a wire or via. For this particular process, the limits are given as part of the TLR. The relevant parameters will be explicitly cited when required.

2.3.3. Useful lifetime

After wires start breaking due to EM, the IC is in the wear out phase of the bathtub curve. The later this happens, the longer the expected useful lifetime of a product. As EM is a function of the foundry process, the expected lifetime can be calculated from formulas given in the TLR. More specifically, a maximum rated current density J_{rated} is constrained per metal stack layer and wire width [14, p. 110].

For this process, the EM is given for an expected lifetime of ten years, when operated at a temperature of $100\,°C$. If the operating temperature is different, or the product is expected to operate for a period other than ten years, the current density constraint can be loosened or must be strengthened. To be more precise, Eqs. (2.5) and (2.6) are multipliers on the maximum current density for EM effects [14, p. 112]. T_{max} is the maximum operating temperature in K and EOL_{actual} is the total on-time of the chip in years. In Sections 4.2.2 and 9.4.3, the influence of EM is verified.

$$J_{max} = J_{rated} \cdot M_T \cdot M_{LT} \quad \text{Equation (2.4)}$$

$$M_T = e^{\left(\frac{9178}{T_{max}} - 23.96\right)} \quad \text{Equation (2.5)}$$

$$M_{LT} = \left(\frac{10}{EOL_{actual}}\right)^{0.909} \quad \text{Equation (2.6)}$$

2.4. Conclusion

All background generally necessary for understanding IC fabrication has been expanded upon. For this particular project, the $65\,nm$ process of UMC is used. Both analog and digital libraries are available. Because IMEC is used as IC service, their MPW projects can be accessed for cheap prototyping. Furthermore, it allows for transistors in three different flavors, low-, regular- and high-threshold voltage. To connect all devices, one poly layer and eight metal layers are available. Metal layer seven is double thickness and layer eight is eight times thickness.

In general, it is important to check for reliability problems on chip. Two methods have been described important for determining the useful lifetime in the ‘bathtub’ curve. Taking IR-drop into account allows for improving yield as well as reducing infant mortality. To extend the useful lifetime up to wear out, the total effect of EM will be monitored and kept within bounds, as specified in the FDK.

3

Power dissipation on chip

As one of the primary design goals is to create a low-power IC (**G**.5b), it is important to know what parameters affect the power dissipation and how. Section 3.1 will start by introducing the on-chip dissipation and terminology. Next, Section 3.2 will explain a method to reduce power dissipation by using the provided technology. Section 3.4 will follow up by explaining the trade-off in performance and power. Some real-life examples are given. Finally, in Section 3.4, possible functional changes are given to reduce power dissipation.

3.1. Terminology

For any specific IC technology, the voltage is fixed. For UMC $65\,nm$, $U = 1.2\,V$. The instantaneous power is subsequently calculated as trivially as shown in Eq. (3.1). The current can vary over time, resulting in the dynamic power, as explained in Section 3.1.1. Any current that doesn't vary results in static power, as explained in Section 3.1.2.

$$P(t) = U \cdot I(t) \qquad \text{Equation (3.1)}$$

$$E = \int_0^T P(t)dt \qquad \text{Equation (3.2)}$$

3.1.1. Dynamic power

As the dynamic power dissipation of a circuit changes with time, and the time window is usually fixed, an easier definition is the energy E. The energy can be pre-calculated for the set time window, as shown in Eq. (3.2). The total energy consumption is subsequently a simple summation of all events. Figure 3.1 shows the main power dissipation paths for a simple inverter.

The shown capacitor is mainly the transistor gate capacitance of the following logic gate, combined with the metal wiring. If it is charged to VDD, by convention the value is a '1'. If it is discharged to VSS, the value is a '0'. Therefore, the capacitance is required for correct functionality, however the size depends on parasitics as well.

Short-circuit power

For the inverter shown in Fig. 3.1, and for any CMOS gate in general, the transistor states depend on the input value. The top P-MOS is only blocking current when the input value is '1'. The bottom N-MOS is only blocking current when the input value is '0'. When the inverter switches, due to an imperfect source and non-zero wiring resistance, it takes a non-zero amount of time.

Due to the non-zero switching time, there is a moment where the input value is somewhere between '1' and '0', causing both the P-MOS and N-MOS to be conducting current simultaneously. The direct current flowing from VDD to VSS, shown by the left arrow, is a short-circuit current. The resulting power dissipation is called the short-circuit power and, this power integrated over the switching time is the short-circuit energy. It is sometimes also called the internal energy, as the energy consumption is internal to the logic gate and does not depend on the output load.

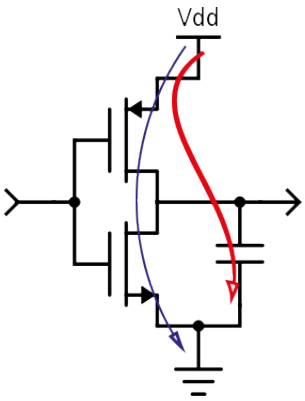


Figure 3.1: Power dissipation paths for an inverter

For multiple input logic gates, like a NAND, it is possible for an input value to switch without the output value changing. As the transition is not visible on the output of the logic gate, it is said to be hidden. The resulting energy is the hidden energy. It is part of the short-circuit energy.

The short-circuit energy can be lowered by shortening the input transition time. By making the driving logic gate 'larger', by using wider transistors capable of conducting a larger current, the transition is faster. Therefore, the total short-circuit time is shorter and the energy consumed is lower.

Switching power

The switching power consumption is shown in Fig. 3.1 with the right arrow. When the output of the inverter switches from low to high, the load capacitance is charged to VDD until it is '1'. Therefore, the total energy consumed in this process can be calculated as $E_{sw} = C \cdot V_{DD}^2$. When an output transition from '1' to '0' occurs, the capacitor is discharged to VSS. Therefore, the energy is consumed.

The total energy consumed by switching is mainly influenced by the capacitance C. A smaller logic gate with narrow transistors has smaller gate capacitance as well as fewer parasitics. As may be clear now, this is a conflicting requirement with the short-circuit energy. Software tools usually are capable of correctly sizing the transistors, for minimal energy losses. Another influence is the amount of transitions that occur. The software will try to reduce the amount of unwanted transitions, called glitches. Other options are blocking transition propagation using latches or applying 'operand isolation' [15].

Switching activity

Of course, not every transition can be blocked, as that would defy the function of the IC. All remaining transitions, are called the switching activity. The switching activity can aid in finding transition hot spots and thereby large sources of switching power dissipation. Furthermore, the short-circuit power also in large part depends on the (input) switching activity.

Gathering switching activity is useful in directing software optimization to certain parts of logic. Furthermore, it is essential in the correct calculation of power dissipation. Without switching activity, only an estimation of the amount of transitions can be made, usually resulting in a poor prediction.

Clock tree

The only signal in the design that constantly switches is the clock. Therefore, correct choice of the clock tree design is crucial to a low power consumption. A very common technique to reducing power consumption is clock gating. By disabling parts of the clock tree that isn't useful for any calculation, the consumption can go down by a large margin. Clock gating is so common that it is automated in every design tool. As little more optimization can be performed, this technique is not further discussed for this project, although it is implemented.

3.1.2. Static power

The static power consumption, as opposed to the dynamic power consumption, does not depend on transitions. It is the current through a logic gate, from power to ground, due to non-ideal characteristics of a MOSFET. As a P-MOS and N-MOS are never truly non-conducting, a small leakage current will exist. The leakage current is inherent to the technology and can never fully be removed. Only removing the VDD and thereby disconnecting the circuit causes the leakage current to be zero.

Even though there will always be some static power consumption, it is usually a lot smaller than the dynamic power consumption. It is 100 or a 1000 orders of magnitude smaller and will only play a significant role when the switching activity is really low.

3.1.3. Total power consumption

The total dynamic power consumption depends on the short circuit energy, switching energy and the respective switching activity f_{switch}. The total static power consumption P_s is a given for every logic gate and part of the FDK. Therefore, the total power dissipation in an IC is given by Eq. (3.3). Reducing the power dissipation usually consists of reducing one of the four parts.

As a program is executed on the processor, the amount of clock cycles it takes is generally fixed, no matter what the clock frequency is. Therefore, a better comparison metric between different designs does not depend on the speed of the clock. Therefore, the used metric is the watt per hertz, or energy, as shown in Eq. (3.4).

$$P = (E_{sc} + E_{sw}) \cdot f_{switch} + P_s \qquad \text{Equation (3.3)}$$

$$E = \frac{P}{f_{clk}} \left(W\,Hz^{-1}\right) \qquad \text{Equation (3.4)}$$

3.2. Trade-off in transistor types

As introduced in Section 2.1.5, this specific process has access to three different transistor types. The effective difference between the transistors is the threshold voltage. Therefore, this determines whether a transistor is open or closed, or how much in between.

A transistor with a higher threshold voltage, will start conducting later. This means during a transition, it is more closed than open, resulting in a lower E_{sc} in the respective logic gate. However, the devices also have a lower driving capability, causing slower transitions. This again causes a higher E_{sc} in the following logic gates. Furthermore, the larger parasitic capacitance might cause an increase in E_{sw}. Therefore, choosing the right threshold voltage is a trade-off in dynamic energy consumption. The main difference can be made in leakage power. The high threshold voltage devices are designed to have a small leakage current, causing P_s to be far lower.

3.2.1. Clock network

The clock network is generally always switching. Therefore, the dynamic power consumption is far greater than the static power consumption. Reducing the leakage current is not very useful and using high threshold voltage devices would still lead to an increase in E_{sc}. Due to the high f_{switch}, it is almost always more beneficial to use the fastest devices possible. For the clock tree only low threshold voltage transistors will be used.

3.3. Trade-off in performance and power consumption

In most designs, power consumption and clock speed are a trade-off. As introduced in Section 3.2, the high threshold voltage transistors can be used to reduce leakage power, as well as dynamic power to some degree. However, their longer transition time make the final circuit slower. In a non-timing critical path it is almost always worth using. By constraining the performance less, by definition the design will have less timing-critical paths. Therefore, it will eventually always result in a lower total power consumption.

Dynamically determining the required performance is an interesting topic in power reduction. For example, the ARM big.LITTLE architecture uses a fast but power hungry ‘big’ processor in combination with a slow and power efficient ‘little’ processor. When the big processing power is not needed, only the little processor is used.

3.3.1. Dynamic voltage scaling

Another technique effective in trading performance for power is called dynamic voltage scaling (DVS). In Eq. (3.3), both E_{sc} and E_{sw} depend quadratically on the power supply voltage VDD. By reducing VDD, significant improvements can be made towards power reduction. However, this limits how 'open' the transistors will be, reducing the total current they can conduct. Therefore, a lower VDD will increase the transition time required for a logic gate.

Therefore, a reduction in VDD reduces power consumption, but also reduces speed. Again, when the predicted work load is low, a decision can be made to do this and to what degree. The resulting effect is what is called DVS. In [16], a VLIW processor very similar to the ρ-VEX, DVS is applied to reduce the total energy required to complete a specific program by more than two times.

Due to its complexity conflicting with a first prototype (**G**.1), it is not used in this work. It will be a very interesting option for future designs. Mostly because a satellite (**G**.5) will usually have plenty of time to perform its calculations.

3.4. Operational modes

Sometimes, it is not necessary for the processor to calculate anything at all. For example, when a satellite is targeting far away planets it can't do anything until the planets are actually reached. In this scenario, the switching activity is zero. The dynamic power consumption is absent, however the static power consumption becomes dominant.

For that purpose, different operational modes should be implemented. Stand-by and power down modes can be helpful in reducing chip-wide power consumption. In other scenarios part of the circuit still has to be active, while other parts can be disabled. This is called partial power down.

3.4.1. Power down

Power down mode can be easily implemented from outside the IC. By removing the connected power supply, no current will flow into the chip. As there is no current, it will not consume any power. A big disadvantage of this method is that the processor usually has to gather some initial configuration information, also called boot. Furthermore, the caches will be empty and the state will be lost. If a previous calculation is needed, it had to be saved to an external memory and be reloaded when powered up again, sometimes taking too much time.

3.4.2. Stand-by

In stand-by, the processor will still be connected to the power supply. As it doesn't have to perform any calculations, clock and thereby all other transitions can be disabled (or gated). The dynamic power consumption will be zero. However, the leakage power now dominates.

To also remove leakage power, power gates can be implemented. A power gate can remove the voltage of parts of the internal circuitry. The designer can choose which parts should be powered down and which parts shouldn't. Usually, all memories and the register remain powered, to retain the processors state, allowing it to quickly continue when necessary. Furthermore, one or a few pads require power to receive an external signal when the processor needs to continue. Otherwise, a small internal circuit like a timer can be used to periodically check.

Like DVS, a stand-by mode requires additional components, the power gates. As only the leakage power is reduced it is not absolutely necessary for an initial prototype (**G**.1). Therefore, it is not yet implemented within the scope of this project.

3.4.3. Partial power down

Power down and stand-by are both modes where the whole chip gets disabled. However, it is also possible to only disable part of the chip. This is especially useful for the ρ-VEX, due to its multiple configuration.

For example, when a program is run that can't really be executed in parallel, it can be configured to use two pipelanes. The other pipelanes are thereby not in use. By gating their power, the full power dissipation of all pipelanes is reduced to a quarter. Furthermore, the ρ-VEX is able to run multiple programs in parallel due to its different contexts. Each of these contexts requires memory to save and restore its state. However, if no program is loaded in the context, the memory is still leaking power. By gating the unused contexts, static power consumption can be reduced.

Partial power down is again a trade-off between performance and power. The many different scenarios have to be analyzed to know whether it is useful to actually implement. Furthermore, changes to the design to allow load prediction and disabling logic have to be implemented. Lastly, the power gates are still a whole new topic, conflicting as a prototype. Therefore, this operational mode is also not implemented. However, especially for the ρ-VEX, it is an extremely interesting and viable option to include in a future design.

3.5. Conclusion

In this chapter, the reader was familiarized with the different energy consumers in an IC. The short-circuit energy and switching energy in combination with a switching activity make up the dynamic power. The total power dissipation furthermore includes the static power. The main goal was to introduce the different ways to reduce power dissipation, as required by (**G**.5b). Dominant components can be predicted for a circuit. Due to its constant switching, the clock tree should be considered separately.

The different threshold voltage for transistors plays a big role in reducing leakage power and result in a trade-off between performance and energy consumption. Another trade-off can be made with Dynamic Voltage Scaling. A known effective technique of reducing overall consumed energy by lowering the voltage. This comes at the cost of a lower clock speed and increased complexity.

The last discussed option for reducing power dissipation are global power down, stand-by or partial power down modes. In global power down, the total IC is turned off, causing a slower reaction time and the requirement of an external memory. Stand-by overcomes these problems by using internal power switches. However, the energy savings are also less.

The most interesting option for the ρ-VEX is partial power down. Its reconfigurability perfectly synchronizes with disabling parts of the circuitry. If the processor load can be effectively predicted, entire pipelanes and contexts can be power gated. The result is a significant reduction in power consumption under partial loads and less timing-critical applications.

As DVS, stand-by and partial power down require changes to the circuitry as well as increase the complexity of the chip design, implementing them would conflict with the main goal of a prototype, (**G**.1). Therefore, they are not implemented in this project and are left for a future design. This work will mainly focus on reducing the power consumption under full load, when no performance trade-off should be made.

4

ASIC design flow

The whole ASIC design flow, and with it most of this project, is concerned with one task: converting an RTL netlist into a GDSII file. The RTL netlist is a generalized hardware description at Register Transfer Level. This description can be written in either of the major languages, VHDL or verilog. The netlist can and should be verified in software through simulation or in hardware on an FPGA. During the flow, the design is simply checked whether it shows the same functionality as prior synthesis. Therefore, if this initial verification is done incorrectly, the final design will be incorrect as well.

As the ρ-VEX is already used in multiple other (FPGA) projects, its design is assumed valid for this project. Hardware verification will not be performed again. Software simulation is still used to show validity of the design, where the initially provided testbench is used as ‘golden’ file. The output is correct and the output of any other simulation should be the same. Only if this is the case is the design considered valid.

Figure 4.1 shows the design steps involved with an ASIC design flow. In the top right the ’RTL netlist’ and ’simulation files’ inputs can be seen. These must be provided for any project. In the bottom left, ’ASIC layout’ is the final design output. This GDSII file will be sent to the IC service or foundry for production.

Every trapezoid in the figure describes a software function to convert several input files to one or more result files. Note that the same function might be performed in different software. The possible software suites are named below every function. For this project, ’Synopsys Design Compiler’ (DC) is used for RTL synthesis, ’Cadence Encounter’ for Place & Route and ’Cadence Tempus’ for signoff analysis. All intermediate files are exchangeable and different combinations of software can be used. The choice for this particular software was made purely on the basis of availability and a different choice might even yield better results.

The rest of this chapter follows the defined software functions. Every trapezoid in synthesis (blue) and Place & Route (red) has a corresponding and equally named section, describing the required inputs and/or provided outputs. Subsequently, Section 4.4 describes several steps to close timing requirements. These can be applied as necessary for a project.

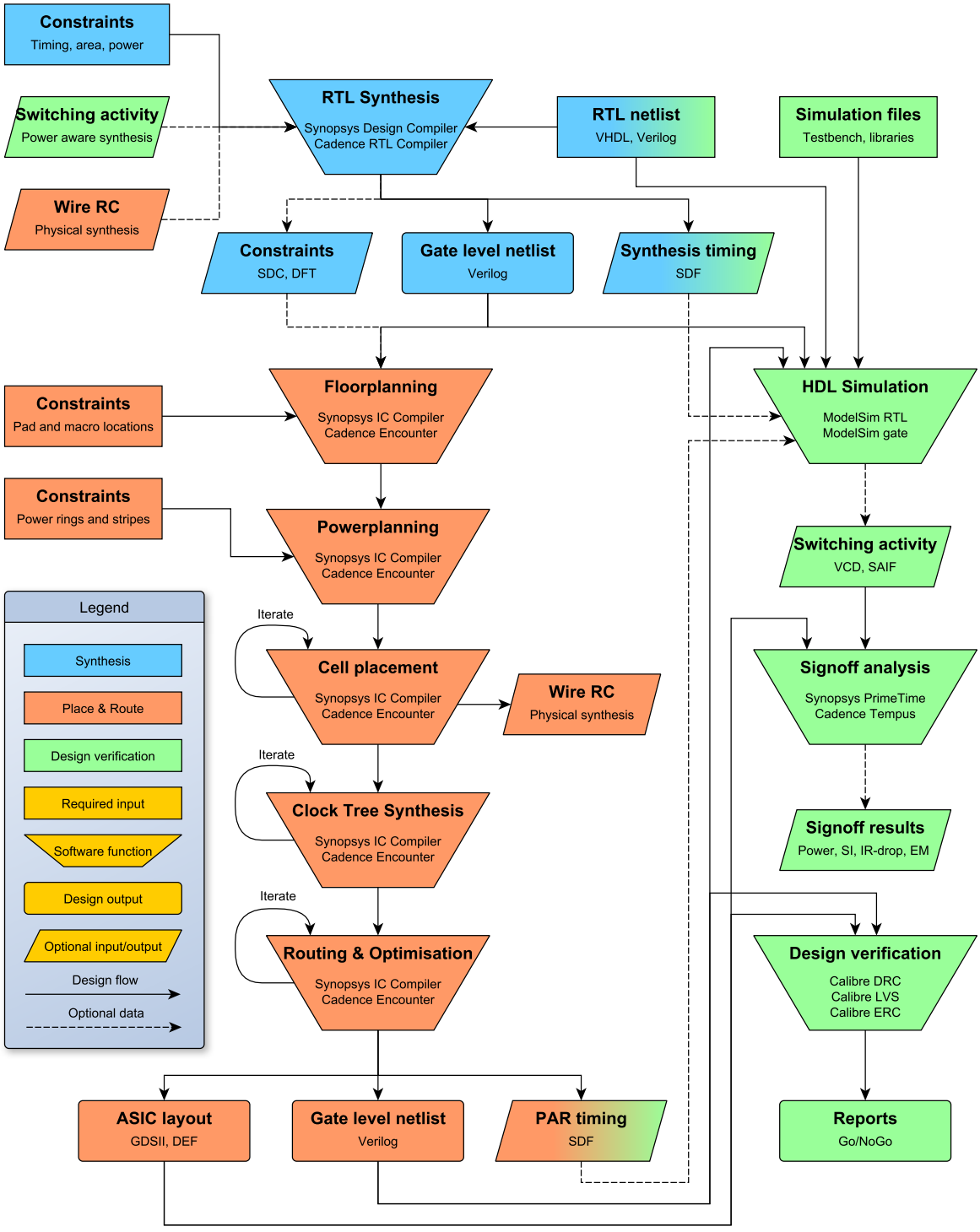


Figure 4.1: General RTL to GDSII design flow overview

4.1. RTL Synthesis

RTL synthesis encompasses the mapping of the described hardware to actual hardware. The main input is an RTL netlist in any HDL language and the output is a gate level netlist. For the most part, there is a one to one translation into library components, in the case of this project to those provided by UMC. The rest of the software is trying to find a 'good enough' mapping. That is, a mapping that meets all constraints specified by the user. Therefore, the most important part about synthesis is setting up the constraints.

The three main constraints that can be optimized during synthesis are timing, area and power. For most designs, reducing the area means more dies fit on a single wafer, reducing their mass-production cost. However, for the MPW run of IMEC, as described in Section 2.2, the total die area is fixed. Therefore, area optimization is not performed and not explained here. Timing and power optimization are explained hereafter.

4.1.1. Timing definitions

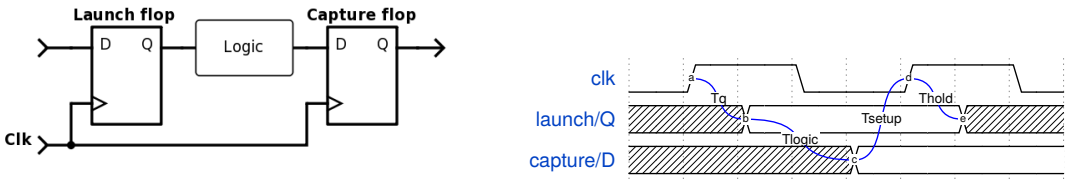
Most designs, and especially a processor like the ρ-VEX, are synchronous designs. This means they operate on the basis of a global clock. All flip-flops are connected to the same clock network. See Fig. 4.2a. All timing paths can be distinguished by a data launch flop, data logic path and a data capture flop. The relevant timing paths are given in Fig. 4.2b.

The four timing paths are defined as follows. T_q is the time it takes for a data transition to appear at the launch flop output Q after a clock edge. T_{logic} is either the best case or worst case time it takes for this transition to travel through the logic. T_{setup} is the time required for the data to be stable on capture flop input D before the next clock edge. T_{hold} is the required time for the data to remain stable after a clock edge on the capture flop. T_q, T_{setup} and T_{hold} are inherent to the design library, while T_{logic} can be influenced by parameters like the amount of logic gates used, the fanout of the logic and the length of wires in between.

Two types of timing violation can be distinguished. Setup and hold violations, respectively violating T_{setup} or T_{hold}. See Eqs. (4.1) and (4.2) for the setup slack δ_{setup} and hold slack δ_{hold}. These must simultaneously be positive for a valid design and mainly depend on the slowest logic path $T_{logic,slow}$ and fastest logic path $T_{logic,fast}$ respectively. δ_{setup} and δ_{hold} are also called the Worst Negative Slack (WNS) and will be denoted as such throughout this document.

$$\delta_{setup} = T_{clk} - T_{setup} - T_{logic,slow} - T_q \quad \text{Equation (4.1)}$$

$$\delta_{hold} = T_q + T_{logic,fast} - T_{hold} \quad \text{Equation (4.2)}$$



(a) Clock timing paths

(b) Data setup and hold timing

Figure 4.2: Static Timing Analysis

4.1.2. Setting up constraints

As already noted, the constraints are probably the most important requirement for correct synthesis. As the tools can optimize timing, area and power, there is not a single 'optimal' mapping. Instead, user limits are relied upon, after which further optimization is not performed. Therefore, constraining the design too little might leave a slow, large and power hungry circuit. On the other hand, constraining the design too much might result in the tool focusing on timing very much, while power consumption rises through the roof. The choice for these constraints is explained further in Section 8.2.

Another important constraint that needs defining are the ICs operating conditions. As explained in Section 2.3.1, the speed of the logic gates, T_{logic}, depends on the actual supply voltage, which may vary. On top of that, the temperature of the chip and variations in the production process changes

the propagation time. Together, these are called Process, Voltage and Temperature (PVT) variations. Calculating timing for every possible combination is possible, but very time intensive and not very useful. Instead, an estimate is made which combination will result in the smallest value for $T_{logic,fast}$ and which combination will result in the largest value of $T_{logic,slow}$. If even these cause positive slack, the design is expected to keep operating under all conditions.

As the process variation is dependent on the foundry and they like to keep the details about fabrication a secret, libraries are provided for the 'worst case' ($T_{logic,slow}$) and 'best case' ($T_{logic,fast}$) process variation. For voltage variations, a deviation of 10% is the industry standard. Lastly, the temperature variation depend on where the IC is going to be used. As this project is focused on space application, the largest temperature range available in the provided libraries is taken. This results in a less 'optimal' design, but allows it to be far more reliable. Smaller temperature ranges allow for tighter constraints, as the slow and fast timing are closer together.

4.1.3. Timing optimization

During synthesis, the placement location of logic gates is still unknown and there are no actual wires yet. This means T_{clk} is the ideal clock period as constrained and might later on improve or deteriorate. However, more importantly, as there are no wire delays, the fastest logic path timing will be far too optimistic and close to zero. Therefore, synthesis does not bother with improving δ_{hold} at all. Instead, this can easily be fixed later on by inserting routing buffers, which will be done in Section 4.2.5.

Therefore, synthesis is focused on decreasing the slowest timing path in the design. (Other designs might focus more on area, but this is not investigated for this project, as the design size is fixed beforehand). As explained in [17], there is no single 'golden' set of rules to optimize this timing. The required steps largely depend on the design. On the other hand, it is also not viable to just try out some optimization and hope the timing gets better. Synthesizing the design is already very time consuming and the amount of possible optimization steps are hundreds. Therefore, it is crucial to understand the design in question and know what can and can't be done. This is in large part just experience. As this was the first design, learning all optimization strategies took the vast majority of time. The most important steps considered will be explained in Section 8.10.

Recently, IBM has been working on an automated synthesis tuning system called SynTunSys [18]. This system claims to achieve 36% timing improvement and 7% power reduction, compared to manual optimization. It is still relatively quick due to its efficient design space pruning algorithm [19]. Even though this seems promising, it is not yet universally applicable. It does show how important synthesis constraints are for achieving an efficient design.

4.1.4. Design For Test

Testing the design is of course the last step of the project, even after production. However, testing a 'black box' can be very difficult and depending on what can go wrong might even be impossible. To ease testing later on, some circuitry is embedded in the design. This is called Design For Test (DFT). As this circuitry might impact timing or design requirements, it is crucial to implement it as early as possible. Therefore, DFT is part of the logic synthesis, or for other designs even part of the RTL netlist. The DFT synthesis steps performed for this particular design are described in Section 9.3.

4.1.5. Synthesis outputs

After synthesis, three results are generated. A gate level netlist, a transposed constraint file and timing information for simulation.

The gate level netlist is the synthesized version of the input RTL netlist. Whereas the RTL netlist only describes the function of an IC, the gate level netlist is actually composed of real logic gates. These gates can all be manufactured using transistors and connected directly using metal wires. The gate level netlist will be the input for the Place & Route phase.

The output constraint file is for the most part a copy of the input constraint file. It is required for the subsequent design stages. Some extra information might be added to the constraints. For example, the DFT performed during synthesis is important for the routing stage. The inserted clock gating cells are of course important for the Clock Tree Synthesis.

Lastly, timing information about every logic gate and every wire in the gate level netlist can be outputted in SDF format. These timing results can be used during simulation to increase the accuracy. It also allows checking for setup and hold violations in external tools.

4.1.6. Optional improvements

There are ways to improve the synthesis results by feeding data from other (later) design steps. These include using simulation data in the form of an SAIF file, or using physical chip result wire delays. The SAIF file can reduce power consumption and the wire delays are used to optimize timing.

The SAIF file contains the switching activity of gates and nets. As synthesis normally doesn't know anything about a design, all nets are expected to switch equally much (which is where power consumption comes from, as explained in Chapter 3). In the real design, however, parts of the circuitry might be idle most of the time while other parts are switching constantly. Of course, the power consumption of the idle parts is mostly static while the power consumption of the active parts is mostly dynamic. Therefore, the optimization requirements are different. This information can be extracted from a simulation, mimicking the real life behaviour of the IC. Note that using the original RTL netlist for simulation allows for annotating most flip-flops in the design. When using the gate level netlist of a previous synthesis run, all flip-flops and almost all gates and nets can be annotated, slightly improving the results.

After placement of the design in Section 4.2.3, physical information is known about the length of wires. The timing delay of these wires is called the RC-delay (as calculated from $\tau = R \cdot C$). This wire RC information might adversely affect design timing, or allows for tighter requirements. Feeding back this information is called 'physically aware synthesis', as explained in Section 4.4.2. Note that because synthesis is only concerned with setup timing, physically aware synthesis is only useful if the final design ends up with setup violations. If hold violations persist, other methods are required, described throughout this chapter.

4.2. Place & Route

After synthesis, a global description of the logic gates required for the described function is available. This gate level netlist also describes the connections between all logic gates, but no physical location and metal wiring between the logic gates is described. This is what the Place & Route (also called P&R or PAR) step is for. This chapter describes the P&R steps towards a physical layout.

4.2.1. Floorplanning

Before placing the logic gates it is important to let the P&R tool know what area it can use for which components. For example, the total design area might be limited to reduce costs. Furthermore, bondwires need to be attached on the outside of the design area. Depending on the connection used (BOAC, flip-chip), the location of these pads must be fixed. For this design, all bond pads must be on the outside of the area, creating an 'IO ring'. Furthermore, some instances might be placed in an external tool and imported as a macro (or black box). The software tools can place these in some smart way, but placing them by hand yields superior results. Finally, it might be beneficial to bound the area of certain groups of logic cells and nets using module constraints or blockages. The IO ring, macro placement, module constraints and blockages are described below.

IO ring

The IO ring will contain bond pads, to make an electrical connection between an internal net and external package pin. All communication with outside the IC will be through the IO pads, including power and ground connections. The exact requirements on the IO ring are described in [20]. The most important considerations are latch-up, ESD protection and different power domains.

Latch-up is caused by parasitic devices when multiple MOSFETs are placed next to each other. When the resistance between the different transistors is very low, a short circuit can occur between power and ground. This can prevent the design from proper function, or even permanently damage it. It is easily prevented by spacing any digital function cell at least $20\,\mu m$ away from an ESD cell, for example power and ground pads [20]. This way the resistance along the parasitic path is large enough to prevent latch-up from occurring.

All power and ground pads have ESD protection built in. This protection discharges sudden large current spikes to the power nets and is required to prevent damage to the input and output pads. For proper ESD protection, the resistance from any pad to power and ground pad must be less than $1.2\,\Omega$. Furthermore, for every power pad a respective ground pad and vice versa must be present.

Finally, different power domains can be created. Digital pads, as well as analog pads are available. Both come in two flavors, core power or IO power. The IO power voltage depends on the used process,

in this case $2.5\,V$. The core power is $1.2\,V$. The different voltages as well as the digital and analog domains can't be mixed. Special breaker cells are provided that need to be placed between the different domains.

Module constraints
Hierarchical modules in the gate level netlist are recognized. Depending on the amount of logic gates within the module, a size is estimated required for placing and routing the module. Depending on the design, it might be needed to increase this size, for example to allow more space for routing or tighter placement for shorter wires. The percentage of assigned chip area actually needed for logic gates is called the Target Utilization (TU). A high TU yields a compact design while a low TU is easier to route.

Besides the TU, it is also possible to assign a certain physical location of the chip to the module. This can be configured using the following constraints: [21, p. 401]

- Guide: a soft constraint which the P&R tool will try to honour, but depending on the needs can be ignored.
- Fence: a hard constraint where all logic gates belonging to the module will be placed within the region, while other logic gates will be placed outside the region.
- Region: like a fence, except other logic gates might be placed within the region.
- Soft guide: a constraint that is not fixed to a physical location.

Blockage
It is possible to force the P&R tool to not place any instances in an area or not route nets. These are called a placement blockage or routing blockage. Blockages can be added and removed during any design step, although during floorplanning is the most common. Blockages can be useful for example to keep some space for special wiring or prevent instances to be placed near or under macros or wires.

Macro placement
Macros are usually an instantiated Intellectual Property (IP), like SRAM cells. To keep the verified characteristics of the macro or prevent giving out information of the IP, these macros are simply an abstraction of all internal transistors. Only the size of the macro and named ports where metal wires should be connected are visible. As these macros can be relatively large and contain many ports, routing might be difficult. Therefore it is useful to manually place the macros relative to module constraints or even to each other. Doing so can significantly reduce the routing required in between instances. Furthermore, a specialized placement blockage called a halo can be added around macros. This halo can be sized relative to the amount of pins on a macro and helps in clearing space for when the macro needs to be routed.

4.2.2. Powerplanning

The power plan is concerned with supplying every transistor on the chip with the correct static voltage. This can be VDD, VSS or any voltage rail required for the chip. For example for this project, there is a separate 'analog' supply rail. This rail also supplies $1.2\,V$, but is split off from the digital parts to separate noise. Another $2.5\,V$ digital and analog rails are used to supply the IO pins.

As already introduced in Sections 2.3.1 and 2.3.2, IR-drop and EM are the main concerns for reliability. The total IR-drop is directly related to how well powerplanning is performed. It can be reduced using 'rings' and 'stripes', as explained below. As the supply rails carry the largest current on the chip, they are also most prone to EM. Starting from the powerplanning phase it is important to calculate the total effect and keep it below threshold.

As the total drawn current is a complex function from the switching activity of every transistor on the chip (which also varies with process variations, temperature, etc.), the total combined effects will be estimated. Many papers describe different formulae to estimate the static and dynamic effects. The smaller and faster the technology node, the more advanced estimations used. In [22], a nice overview is given. For the most part, the software tools can give such an estimate and visually display the IR-drop. Depending on the desired limit, the design can iteratively be improved. This is done in Section 8.5.

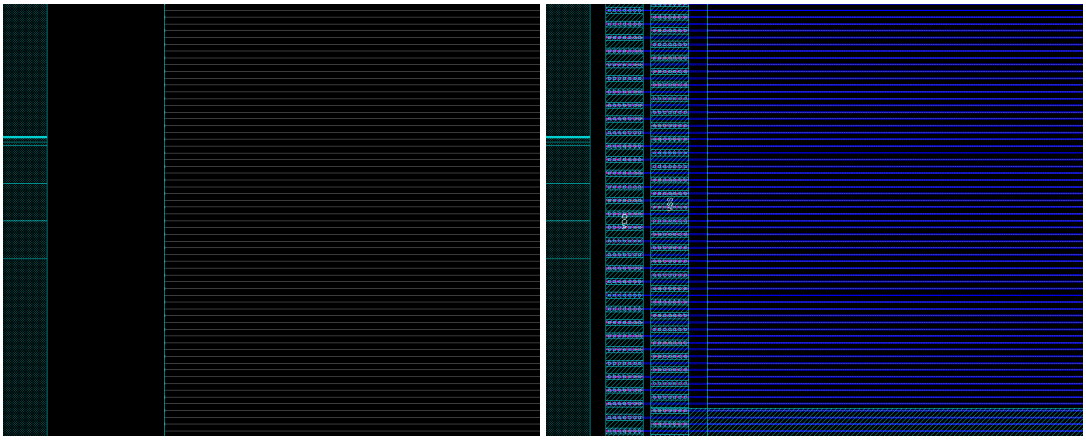
Besides the power rings and stripes, power will be directly routed to all standard cells. Even though placement will only be done hereafter, the locations are fixed and power distribution can already be performed. Lastly, power switches can be implemented to reduce power consumption of the chip.

Power rings
To improve stability of the power supply, every domain (analog, digital) gets surrounded by a ring. This ring will be directly connected to IO pads, supplying it with the correct voltage. As the ring will not interfere with later routing, it can be made relatively wide and possibly on multiple metal layers. This creates a very stable voltage near every side of the chip.

If later on there is are problems in the ring itself, the wires can be widened. Another improvement can be to connect power pins on all four sides of the chip area. This allows shorter current paths within the chip, as long as the pins can be held at a stable voltage. Essentially bringing the problem to outside the package. The final IC will probably be mounted on a PCB, where area and thereby metal wires are cheaper allowing for easier distribution.

Power stripes
After the power ring there might be IR-drop issues towards the centre of the chip. These cannot be solved by widening the ring. Instead, stripes can be added to split up the ring. These stripes allow for wider metal connections on top of the standard cell placement area. This reduces the connection length through thinner wires. However, as these stripes are on top of the standard cells, it also affects the routing congestion. If during routing there are problems near the stripes, it will be important to reduce the number or the size of the stripes. These problems are reflected upon during implementation in Section 8.5.

Standard cell rows
To ease the power routing and placement of cells, every logic gate in the standard cell library is created with a fixed height. This allows the design area to be split up in equal height rows before any cell is placed. Furthermore, every standard cell has power connections on the top side and ground connections on the bottom side (or reversed, depending on the placement orientation). This allows for the power to be routed to all standard cells, before any of them are even placed. See Fig. 4.3b for such a power distribution on an empty area.



(a) Standard cell distribution before placement (b) Power distribution before placement

Figure 4.3: The chip active area is split up in rows of standard height

Power switches
As explained in Section 3.4, it is beneficial for the chip to stop parts that are not in use. If only the logic is stopped, the dynamic power is minimized but due to leakage it will still draw power. Furthermore, it is possible to (temporarily) lower the supply voltage for parts of the chip. This will slow down the devices, but reduces power as well. A viable power reduction technique if the slower devices are acceptable.

Power switches are added to the library for these full or partial power down modes. These power switches allow for dynamically switching on or off the power on parts of the power distribution. Otherwise they allow for crossing two power domains, lowering the voltage on parts of the chip. These power switches are outside the scope of this document, but might be interesting for future work.

4.2.3. Standard cell placement

After the power plan has been made, all standard cells have to be placed. The standard cells depend on the process. These library components are included in the gate level netlist. The height of each of these cells is fixed, as well as the connections between the cells. However, to improve the WNS, it might be beneficial to use larger transistors. Or to reduce power consumption, HVT, RVT or LVT devices can be used. The process of choosing the right library cell is called 'logic gate resizing' or 'footprint swapping'. The former name is used in this document. Note that hold violations could only be fixed after a clock tree is present. As this is still not the case, only setup timing is taken into account. Therefore, resizing needs to be done after clock tree synthesis and in fact again after routing as well.

Logic gate resizing, as its name suggests, actually changes the size of the cell as well. The height is fixed, but the width can vary. To allow for resizing during this and in later stages, the chip area is not totally filled. The utilization is not 100%, as explained below. This can also be beneficial for routing, which can be estimated as a routing congestion.

Well tap cells

Specific to this library, and many others on the same or smaller node size, is the fact that it is a 'tapless' library. This means the standard library cells don't actually have a proper power and ground connection. As the transistors have become very small, a connection to the power rail has a significant area impact. These connections are called power taps. To save on area, the power and ground connections of multiple transistors are shared through an N-well or P-well. For every bunch of transistors a well tap cell is placed. The maximum distance between well tap cells again depends on the IR-drop and can generally be found in the FDK. Due to the fixed distance, well tap cells are placed before any other standard library cell.

Routing congestion

The physical placement of gates has an effect on the total length of the wires connecting them. But if the connections become longer, more physical wires are needed for the connection. As the amount of metal layers and the minimum width of a wire are fixed, only a finite set of wires can later on be routed from one point to the other. If in a certain area there are more connections than there is space for wires, it is called a congested area. The implication of congestion is either a forced detour (leading to longer, slower wiring), or impossibility of routing. Both are only detected during the last design step, routing. To prevent needing to go back and forth between placement and routing, a congestion analysis can be made by the software tool. Depending on the result, the designer needs to decide whether the placement is adequate or needs changes. Recent design papers describe the congestion problem and possible solutions in more detail [23].

Chip utilization

The chip utilization affects logic gate resizing as well as routing congestion and is otherwise known as the placement density. It can be calculated as the ratio of area used by library cells and total chip area. In Encounter, there are three ways to influence the placement density.

The maxDensity parameter defines a global utilization of the chip. During placement, Encounter tries to add a spacing between cells to locally account for this constraint. The maxDensity may be used to allow resizing during placement. Encounter will ignore the constraint if it allows for shorter wires.

Using modulePadding, every cell is effectively made larger by a given percentage. This type of padding keeps cells separated during initial placement, but is ignored in later design steps. It is legal for the software tool to overlap module paddings and is thereby just a soft constraint. This can be effectively used for congestion reduction.

Using cellPadding is essentially the same as a modulePadding. However, the cellPadding is a hard constraint. This means that every cell is made larger by the percentage and Encounter has to keep this extra size available. The padding can be removed by the designer in a later stage when the extra space is required. For example, hold violations can be fixed by forcing an extra delay using special delay cells. The cell padding can be swapped for delays cells during routing.

Multiple iterations

As explained, the placement of cells affects the wire length and with it the designs WNS. However, it may also affect routing congestion later on. Furthermore, the cells need to actually fit on the design

area, not overlapping each other. These conflicting constraints make placement an NP-hard problem. Calculating the most optimal placement might be possible, but takes too much time and computing resources. Therefore, the software tool will simply give a possible solution which it thinks is good. Subsequently, it adapts the solution, improving the slack and congestion. As indicated in Fig. 4.1, running multiple iterations might yield better results. It is the designers responsibility to set up reasonable targets and decide when the solution will yield a viable design. This is done in Section 8.6.

4.2.4. Clock Tree Synthesis

As mentioned in 4.1.1, in a synchronous design the clock defines the timing of an IC. During synthesis, the clock is expected to arrive at every sequential element in the design simultaneously. During Clock Tree Synthesis (CTS), this ideal requirement is implemented in logic. There are a lot of techniques to perform CTS. Here three are described, as used to arrive at the final implementation.

Regular CTS

In regular CTS, the clock is designed to arrive at every location on the chip at the same time. The clock root is connected through multiple buffer levels to all sequential elements. The objective is to make the skew as small as possible, where skew is defined as the difference in arrival time between the latest element and earliest element. The software tool will generate a clock network with skew within constrained bounds. Note that a lower skew requires more elements, leading to a higher power consumption. Therefore, not constraining the clock too much can be beneficial.

Besides skew, slew is an important metric. Slew is the time it takes for the clock net to switch from zero to one, or from one to zero. A bad slew in the clock net propagates as a bad slew to the logic, slowing down the circuit. As the clock net is switching constantly, dynamic power dominates over leakage power. Therefore, to improve slew it is advantageous to use the fast LVT devices, instead of focusing on reducing insignificant leakage power.

CTS using useful skew

With regular CTS, the clock skew was kept as low as possible. In some cases, however, clock skew might be beneficial for timing instead. See Fig. 4.4 for an example. The timing path between flop 1 and flop 2 is $14\,ns$. Using regular CTS, the clock period can be $14\,ns$ at minimum. However, by inserting a delay of $4\,ns$ in the clock of only flop 2, the capture path of flop 1 to flop 2 gets lengthened by this time. This way, even a period of $10\,ns$ lead to zero slack. The implication of this is that the launch path of flop 2 to flop 3 also gets delayed by the $4\,ns$. This means the second logic path can be $6\,ns$ at max. Therefore, a small bit of time is borrowed from the second path to relax the constraint on the first path. This principle is called useful skew and allows for tighter setup and hold timings.

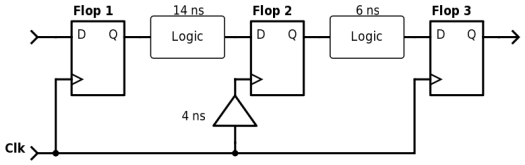


Figure 4.4: Useful clock skew in a circuit

Clock Concurrent Optimization

Clock Concurrent Optimization (CCOpt) is a technique developed by Azuro and acquired by Cadence. Therefore, it is only possible using Encounter. See Fig. 4.5 for a short overview of what it does. the insertion delay is the actual time it takes for a transition in the clock root to arrive at a sequential element.

Essentially, CCOpt is a smart way of integrating useful skew CTS with placement optimizations. The software gathers information about every logic path in the design. Starting from the most timing constrained path, it checks how much skew it can borrow from the following path. This is done for the whole design, finally gathering the minimum and maximum arrival times. For every loop in the design where timing cannot be met, it will start to optimize the logic paths. Reducing or lengthening the transition time depending on whether a flop is constrained by the maximum or minimum arrival time.

After the CCOpt is finished, a clock network is constructed with optimized logic paths and a targeted useful skew for each sequential element in the design. Like useful skew it allows for even tighter timing by utilizing very skewed logic paths.

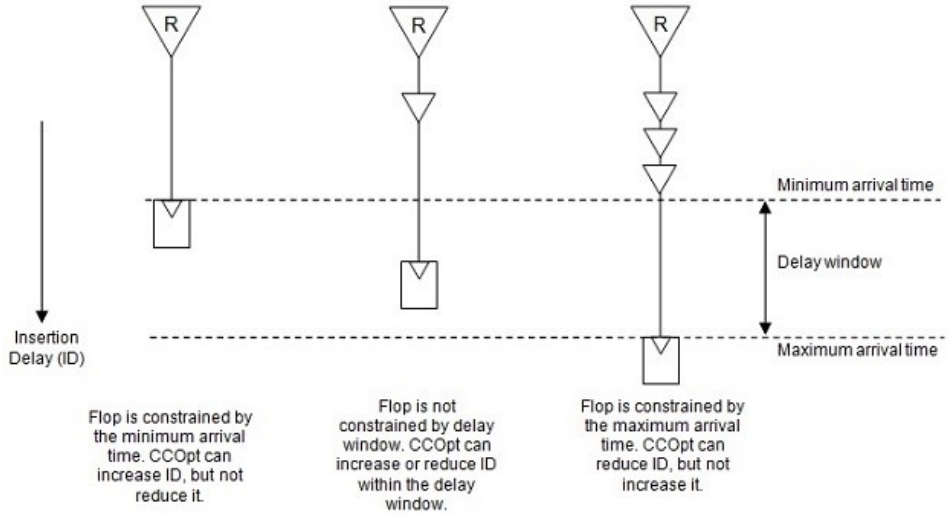


Figure 4.5: Clock concurrent optimization, optimizing the launch, capture and logic path at once

4.2.5. Routing & Optimization

In Encounter there is very much a designer can do to directly influence the routing of the design. By default, routing is performed as a global routing phase, Signal Integrity (SI), Optical Proximity Correction (OPC) and timing aware detailed routing phase and rounded up by antenna violation fixing. Even though the routing itself can't be changed, the results still very much depend on the synthesis, floorplan and standard cell placement. It is very useful to know how a design will be routed prior to starting it, as problems in the routing steps might mean the whole design needs to be redone. General information about the global and detail route can be found in [24], specifically Steiner trees (page 700) and dogleg routing. The document and many others also explain SI end OPC problems and, therefore, will not be repeated here.

A few tricks to influence timing are applied and explained below. Next, fixed routing directions are introduced as a tool to ease routing. Furthermore, multi-cut via insertion and possible antenna violation and routing violation fixing are explained in more detail.

Timing aware routing

By default, the software tries to route the design in two steps. First, the design is routed such that setup WNS is zero. Subsequently, it tries to adapt the routing until hold WNS is zero while maintaining the same setup slack. For most designs, hold WNS is easy to fix. However in the case of the ρ-VEX, fixing hold slack would destroy setup timing and, therefore, more steps had to be taken.

One of these was already introduced during placement and can be utilized now. Before hold violation fixing, the initially added cell padding can be removed such that more delay cells can be inserted. This is necessary if the hold WNS couldn't be fixed otherwise.

Another trick is to force Encounter to find a better solution than necessary. Instead of routing for a setup WNS of zero, a slightly larger slack is built in. Subsequently, for hold violation fixing this constraint is relaxed again. This way Encounter puts more effort in timing during routing. It is only possible to do so if the design is not restricted by setup timing, but by hold timing alone.

Fixed routing direction

Like placement, routing is again an NP-hard problem. Solving it in the ideal way is near impossible (yet). Therefore, every routing software will employ a trick to make it quite a bit easier. Instead of allowing free routing, all even metal layers (layer 2, 4 etc..) are used for vertical wires. All uneven

metal layers are used for horizontal wires. When a connection has to be made diagonally, it is partially routed on an even layer and partially on an uneven layer.

Doing so significantly improves the density of routing, without impacting results too much. As, for example, metal 2 is used only for vertical connections, all wires routed on this layer can be laid out directly next to each other. No space has to be left for horizontal wires.

The downside is that some wires are still preferred on a specific layer, preventing this methodology almost completely. For example, two logic gates very close to each other can be efficiently routed directly on the lowest metal, metal 1. If this would make a vertical connection, all wires running on the same space would have to be rerouted. Sometimes this would lead to a more efficient design. However, Encounter is very rigid in its routing directions and will complain if forced to violate it. This is a design choice that probably leads to a better design in most cases.

Multi-cut via insertion

In regular routing, a via will connect two metal layers where necessary. As a result of imperfect manufacturing, it is possible for a via to have a far higher resistance than normal, or not exist at all, leading to a defect IC (yield loss). A common solution to this is to simply add two vias, leading to a multi-cut via. The vias are redundant, meaning if one of them breaks, the IC will still function correctly. In Section 8.8 the result of this option is shown. Note that as multi-cut vias require at least twice to three times the space, this might be a difficult task for very congested designs.

Antenna violations

An antenna violation is a problem during manufacturing that might damage the IC. See Fig. 4.6 for an example. After metal 2 is created and before metal 3 is added, a large piece of metal might not be connected to a diffusion area, but only to a transistor's gate. The piece of metal will function like an antenna, getting charged by ions used in manufacturing. It will discharge through the only connected path, the transistors gate. If the antenna is large enough, the built up charge can damage the gate.

The definition 'large enough' antenna is a function of the design process. Constraints can be found in the TLR. Encounter is smart enough to recognize antennas and will try to fix them. By default, it will only prevent antenna violations during routing. If a situation occurs where the large metal is required, it tries to connect to a gate only through a higher metal layer, so called jumper insertion [25]. This way the discharge path is only directly to the substrate. Sometimes this isn't possible, generating errors in the design detected during verification (Section 9.5.1).

Alternatively, the designer can choose to add antenna cells. These cells are diodes placed near the gate connected to a violating wire. The diode allows a discharge directly to the substrate, protecting the gate from damage. The downside of such an antenna diode is the extra required area and some leakage. Therefore, Encounter will not automatically add them.

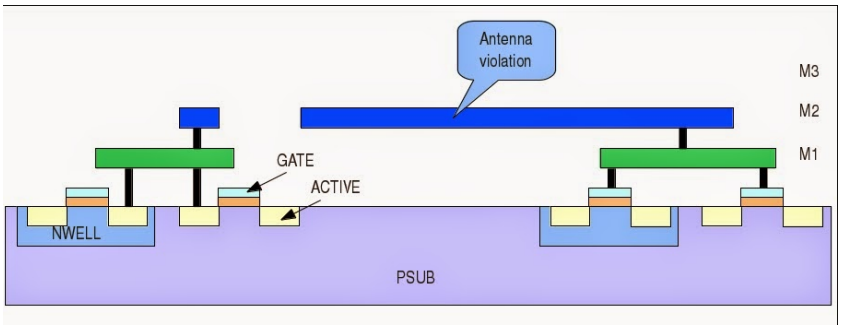


Figure 4.6: Antenna violation due to long wire

Routing violations

Besides antennae, routing violations might include shorts and spacings. Due to a high routing congestion the software tool was forced to draw two wires in the same location, obviously creating an invalid design. Encounter will happily do so and finish routing the rest of the design, leaving the violations. There are two methods of repairing these sorts of errors, reroute or replace.

Rerouting involves deleting all violating nets and possibly nearby nets. The software tool can be used to route these nets again, trying to find a better solution. If at all possible, violations are reduced. This may be iterated multiple times until no improvement is made.

Replacing is a more rigorous and possibly more correct solution. Apparently, the design was still too congested after standard cell placement. The designer can choose to redo placement, or even add more constraints to the floorplan. Congestion analysis should be rerun to see if the new results are satisfactory. This will need a rerun on the whole tool flow. Therefore, it is an unwanted solution, but necessary if all else fails.

P&R results

After routing is finished, the final results can be written out. As indicated in Fig. 4.1, this can be done in three parts. The layout itself can be exported in many different formats, useful for different tools. The most common format is GDSII. This file contains a drawing of every transistor, metal wire and via of the IC. The GDSII can be opened in Virtuoso for visual inspection or used for analysis and verification. Furthermore, a GDSII file is required for fabrication.

Besides the layout itself, another gate level netlist and timing SDF file can optionally be exported. As explained in Section 4.3.3, the netlist is required for verification. Furthermore, the netlist together with optional timing information are needed for functional simulation in Section 4.3.1.

4.3. Design verification

In Fig. 4.1, design verification is indicated in green. It involves checking whether the design functions, initial assumptions were correct, the design is manufacturable and, quite importantly, if the implemented design is actually what was intended.

Verification is split up in three major steps. First, an HDL simulation is run using a netlist to check for functional mistakes. This simulation can be run at different stages of the design. Subsequently, the final P&R result is used for signoff analysis, checking whether initial constraints and final results are adequate. If the design is satisfactory, it is verified. Any notable error in design verification leads to a go or nogo result, deciding whether the final layout will actually function after manufacturing.

4.3.1. HDL Simulation

Using ModelSim and a testbench, the design can be functionally simulated. There are three design stages at which simulation is useful. The initial RTL netlist should always be simulated. If it is not correct, the whole design can still be synthesized, P&R'ed and verified correctly. However, functionally the chip might not work. Furthermore, this initial simulation is useful as golden file. The output can be saved and compared to the output of the other simulations. If at any point the simulations fail, it can be compared with the golden file to see where in the design is a mistake. The initial simulation is at register transfer level and, therefore, only behaviourally. Design IP can be introduced. For example, a behavioural memory can be replaced by a (simulated) physical SRAM cell. These kind of IP is still just a simplification of real hardware, but verified by the manufacturer and should be correct.

It is very important to note that the simulations are only as good as the testbench is. Any case tested can be verified. However, extensively testing every different state of the design is impossible. Therefore, a correct testbench is a requirement, but correct output does not guarantee full functionality of the design!

After synthesis, a gate level netlist can be exported. This gate level netlist contains actual hardware. Note that it is only the logic gates that are exported (not the actual transistors) and connections between them (no physical wires). The netlist can be simulated with the same testbench and the result should be the same as the RTL simulation. This can be used to verify whether the implemented logic functions are still correct. Optionally, timing delay information can be added to the design through an SDF file, but as the clock net is still ideal and hold violations are not fixed, this is still not very reliable. The downside of this simulation is that it can almost only be used for verification. Debugging is extremely difficult, as the names of the original netlist are mangled or not available anymore. A better method is to use 'formal equivalence checking'. This is an alternative solution to simulation. Every element in the synthesized design is mapped to its counterpart in the original RTL netlist. Subsequently, the designs can be proven to be functionally equivalent. If there are no differences, the design can be assured to be correct. As the software was not available during this project, debugging is still done using simulation. For a future project it might be interesting to take a better look into formal equivalence checking.

After P&R, a comparable simulation with timing information can be run. This time, as opposed to post synthesis, all violations should be fixed and the full simulation must run correctly with timing. If this is not the case, there is either a problem with the RTL netlist or any of the constraints was specified incorrectly. It is up to the designer to debug which lead to a discrepancy between RTL and gate level simulations.

From all three types of simulations switching activity can be extracted. The initial RTL netlist or post synthesis gate level netlist simulation activity can be used to improve synthesis results. By inserting one or more real life applications, the synthesis tool can optimize with a better target. The post P&R simulation switching activity can optionally be used for improved signoff analysis, again calculating real life power numbers instead of using an estimate. More details are in the respective sections.

4.3.2. Signoff analysis

Signoff analysis consists of two parts, constraint verification and result extraction. Initially, in Section 4.1.2, timing analysis was set up to use only the best case and worst case PVT corners. This was a simplification to speed up calculations. In reality there are a lot more possibilities for corners. As signoff check, timing and SI is calculated once with the highest possible accuracy for every PVT corner. If the initial best case and wort case assumptions were correct, there shouldn't be any problems. If there are still problems, it is the designers responsibility to detect where the discrepancy resulted from and adapt the best case or worst case corner. If this is the case, the whole design needs to be redone.

As result a lot of metrics can be extracted. Think of design area, maximum or minimum achieved clock speed, transistor counts etc. In this particular design, a goal was to construct a low power IC. Therefore, a high precision power analysis is done. This can be done at transistor level and including real life application switching activity, extracted from simulation. Furthermore, a transistor level dynamic IR-drop and EM analysis are possible. Anything to check if the design is good enough, or should be improved upon.

4.3.3. Design verification

Finally, the design has to be verified. This verification is done in the standalone program Calibre using the actual layout. This results in a higher precision check than the P&R tool can do. Therefore, antennae are checked again. Furthermore, as explained below, three additional checks are performed, DRC, ERC and LVS.

Design Rule Check

The Design Rule Check (DRC) is used to verify the TLR as stated by the foundry. Checks include minimum and maximum wire width, minimum distance between wires, metal density and a lot more. The more mature a process is, the more extensive the checks become. A design with zero Design Rule Violations (DRVs) is more reliable and will likely result in a higher yield. A foundry will usually reject a design that does not conform to the DRC. Only under certain circumstances is it alright to waiver checks, but will likely still have implications.

Electrical Rule Check

The Electrical Rule Check (ERC) simply checks if every transistor is connected to the right supply (VDD or VSS). If for some reason there is a mistake in the power plan, a power ring or stripe is not defined correctly, this will be caught using the ERC. Another mistake might be the absence of well tap cells, while the library requires them.

Layout Versus Schematic

The Layout Versus Schematic (LVS) check compares the functionality of the layout. After P&R two files were created, a layout in GDSII format and a gate level netlist. Only the gate level netlist was simulated and verified. However, the layout file will be sent to the manufacturer. This is where the LVS comes into play, it checks whether the written layout is still equivalent to the verified netlist. This is done by reverse engineering the layout. Every drawn transistor is converted back to an actual device and every drawn metal layer is converted back to wires. Subsequently, these devices and connections are compared to the devices and connections as described in the netlist. Any discrepancy has to be debugged by the designer. The LVS is as opposed to DRC not obligated by the foundry, but helps in preventing creating a functionally useless IC.

4.4. Timing closure

As introduced in Section 4.1.1, all the tools are designed to close timing. However, it might be the case that even after the software's biggest effort, this is still not possible. This section describes two additional techniques that might help to achieve timing closure, called Engineering Change Order and Physically aware synthesis.

4.4.1. Engineering Change Order

The regular design flow as described in the previous sections is very linear. Step by step the design is converted to a final layout. However, if during standard cell placement it is detected two macro cells are spaced too far apart, or during routing it is impossible to achieve timing closure because a standard cell is placed inefficiently, the whole flow needs to be redone. The newly acquired information can be fed back into the initial constraints to get a better design. It might be required to do this several times, consuming a lot of time in the process.

This is where Engineering Change Order (ECO) comes into play. It is a collective of several techniques that prevent having to redo the whole design. It allows for example during routing to move a few standard cells around. The software will detect this change and redo the placement, clock tree synthesis and routing for just these cells, saving a lot of time. Other options include improving power routing if the final design shows a too large IR-drop [26] or even adding spare cells to the design that can be used for design improvements after routing [27].

4.4.2. Physically aware synthesis

After the standard cell placement it might already be impossible to close setup timing. Even though after synthesis setup timing was met, the physical placement of cells might cause longer delays than expected. See Fig. 4.7. In some cases, optimizing standard cell placement (using ECO) improves the result, but sometimes the added delay is too much. If a design has this problem it is possible to apply physically aware synthesis.

As can be seen in Fig. 4.1, after standard cell placement the wire RC can be extracted. This is the resistance and capacitance information, causing delay on a net. On a new synthesis run, the estimated values can be replaced by this physical information such that more accurate delay calculations are performed. All steps before and including cell placements have to be redone.

It is important to note that physically aware synthesis is performed using all steps prior to CTS. Therefore, it is only useful to resolve setup violations. Furthermore, newer tools by both Synopsys and Cadence are able to integrate physically aware synthesis from the first run, removing the overhead of redoing synthesis and cell placement. This project does not implement physically aware synthesis, as these newer tools were not readily available and setup timing was easily met. For future reference it might be interesting to keep these improvements in mind.

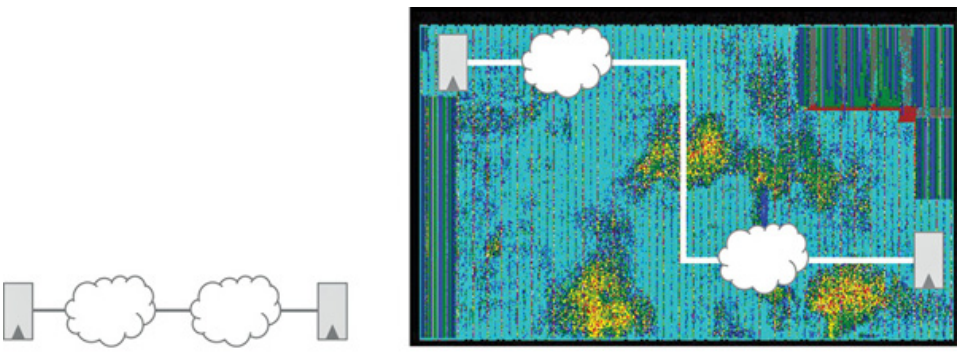


Figure 4.7: Physical delays can be large due to physical placement [28]

4.5. Conclusion

An overview is given of all design steps necessary for converting an RTL netlist into a verified and manufacturable layout. The flow includes synthesis for mapping a design description onto logical gates and Place & Route for mapping logical gates on a physical chip. The flow is a global overview of how the ρ-VEX was optimized, but can be applied to any other design. Two additional improvements are touched upon, one to reduce power dissipation and one to achieve tighter timing closure.

Should it be necessary to reduce the static power dissipation, then it is possible to divide the IC into different power domains. Using power switches provided in the library it is possible to locally lower the voltage or power down the whole chip. In this work, power switches are not yet implemented. The main focus for this project was laid on reducing dynamic power dissipation, whereas power switches mostly reduce leakage power.

To achieve a tighter timing closure, physically aware synthesis was introduced. This improvement can be helpful when future designs focus on clock speed. Again, this is not yet applied as the main focus laid on reduced power dissipation. To implement physically aware synthesis, the design tools as well as the software scripts need some changes.

5

Source-Synchronous Interface

Following an initial design constraint and (**G**.2), there is a need for off-chip memory and a communication interface with it. Therefore, the Source-Synchronous Interface (SSI) is devised. The SSI is implemented in a separate project, parallel to this project. Both projects were developed in close collaboration such that the interface can be optimally integrated into the ρ-VEX. What the SSI exactly does and how it is implemented can be read in a separate document. Only the integration specific details are described in this chapter.

Starting with Section 5.1, the choice for off-chip memory is motivated. In Section 5.2, the SSI is described as an IP block. In Section 5.3, the important interface connections and considerations are described. Finally, Section 5.4 describes the files necessary for abstraction of the IP block, including simulation files.

5.1. Motivation

In any IC or processor design, memory is very expensive. Modern processors implement up to three or four different cache levels, each level decreasing in cost and performance. As this is only a prototype chip, the cost should be reduced even further. Manufacturing cost is directly dependant on the IC area, a comparison is made with the theoretical maximum memory this chip can hold.

As an initial constraint to this design, the total chip area available is $1875\,\mu m \cdot 1875\,\mu m$ (**C**.5). For the bond pads in the IO ring, about $120\,\mu m$ will be used on all four sides, which is a technology limitation. As an example, one SRAM implementation of 256 words by 128 bits has size $299\,\mu m \cdot 101.4\,\mu m$. See Eq. (5.3). At a memory density $\rho_{mem} = 1.08\,bit\,\mu m^{-2}$, the total amount of on-chip memory that could possibly fit is $S_{mem} = 353\,KiB$. Which is nowhere near enough for any real application. This calculation even disregards the processor area.

By using a small portion of the IC area for a communication interface, any amount of external memory can be attached. As the external memory is not part of the ASIC, but a generic piece of hardware instead, its cost is significantly reduced. The ρ-VEX can fit on the IC area and costs are reduced, as required for (**G**.1).

$$\rho_{mem} = \frac{256 \cdot 128}{299 \cdot 101.4} \qquad \text{Equation (5.1)}$$

$$\approx 1.08\,bit\,\mu m^{-2} \qquad \text{Equation (5.2)}$$

$$S_{mem} = \rho_{mem} \cdot (1875 - 2 \cdot 120)^2 \qquad \text{Equation (5.3)}$$

$$S_{mem} < 353\,KiB \qquad \text{Equation (5.4)}$$

5.2. IP block

Usually, parts of IP in a design are implemented as a black box. This allows a company to sell their product, without giving away details about the internals. Even though the SSI is developed in collaboration and the way it works and is implemented is publicly available, it is still useful to implement it as an IP block. The main reasons for this are its analog behaviour and to simplify integration, as explained below.

5.2.1. Analog parts

The SSI is implemented for communication using Low Voltage Differential Signalling (LVDS). This is a protocol based on current signals. Furthermore, it works at a lower voltage than the digital circuit. This makes it a mostly analog part (with some digital circuitry as well) [29].

Encounter is a tool used strictly for digital purposes. It doesn't deal well with analog parts and by default it doesn't even know what a transistor is. When timing becomes important the calculations performed by Encounter will be horribly wrong and can't be relied upon. Therefore, the SSI is developed using different tools and imported as a black box.

5.2.2. Integration simplicity

Another advantage of an IP block is its agnostic nature. Both the ASIC developed in this project and the separately developed SSI didn't know the others characteristics when the projects started. Some targets were set, but the exact details were, of course, unknown. Therefore, it is useful to fix the interface in a black box. The ASIC can be developed around the black box, while the innards of the black box can change simultaneously, both not influencing one another. The SSI could have been implemented in any other project just as easily.

5.3. Interface

The SSI will be implemented externally. The application note as provided is attached in appendix A. An overview of the IP block and its integration on the ASIC is given in Fig. 5.1. Here, the leftmost connections are analog pins as will be added to the IO ring. The ρ-VEX core is provided as RTL netlist and will be implemented in Chapter 8. Following this section, the clock divider and DDR units are explained.

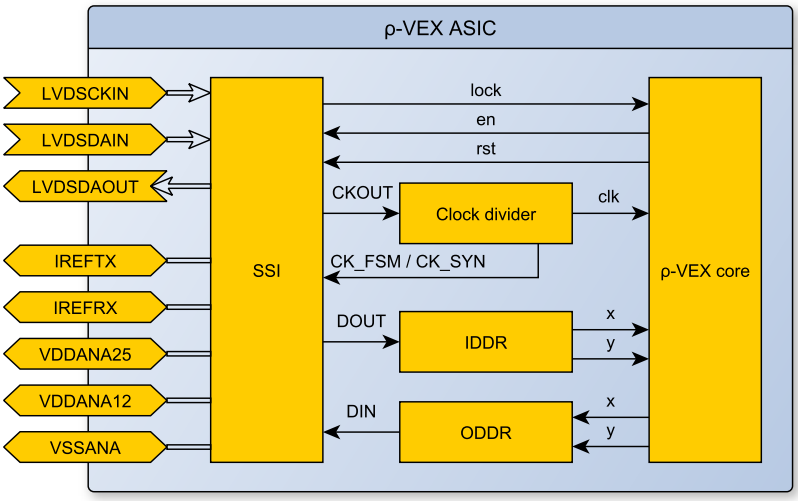


Figure 5.1: SSI as implemented on ASIC

5.3.1. Clock divider

To provide maximum data throughput, the SSI is designed to run at $400\,MHz$. Over the course of both projects, the actual achievable speed of the SSI was not known. Therefore, a fallback option is also implemented to run at $200\,MHz$. However, as will become clear in Chapter 8, the ρ-VEX core can only run at $100\,MHz$. This means the clock as will be recovered by the SSI has to be divided either four or two times, depending on the operational mode. This is why a clock divider needs to be implemented between the SSI and the ρ-VEX core.

Implementing a clock divider is relatively simple. The circuit as will be used is given in Fig. 5.2. The ratio determines whether two times division (for SSI running at $200\,MHz$) or four times division (for SSI running at $400\,MHz$) is used. Note that although clock division is easy, problems arise when crossing clock domains. Implementation and verification of the issues associated with such a boundary is reported in Section 8.2.4 and Section 9.1.3 respectively.

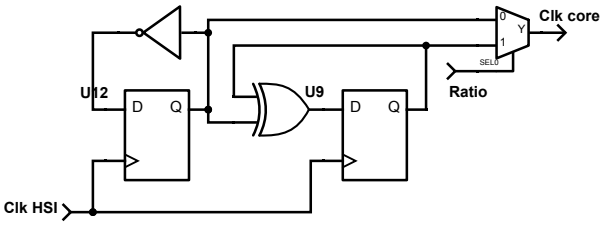
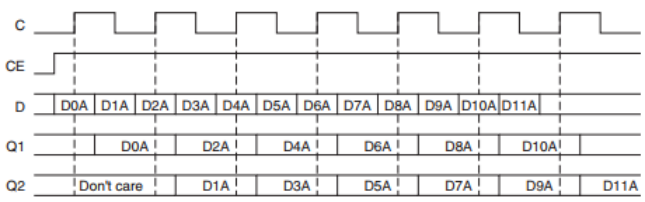


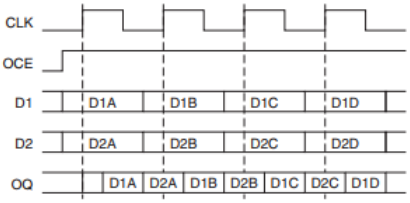
Figure 5.2: Clock divider circuit

5.3.2. DDR receiver and transmitter

Part of the LVDS protocol as implemented in the SSI is that it runs at a Double Data Rate (DDR). This means that data will be sent and received at both the rising and falling edge of the clock, instead of only at the rising edge. Allowing for double the data being sent without doubling the maximum signal frequency. The implication of this is, however, that the ρ-VEX core single data rate signals have to be converted to DDR. The protocol for doing this will be determined by the FPGA external to the ρ-VEX, sending the LVDS data. More specifically, this is a Xilinx Virtex-6 FPGA with Input DDR receiver and Output DDR transmitter buffers, both running in 'SAME_EDGE' mode. An overview of these units can be found in the respective datasheet [30, p. 94,122], see Fig. 5.3.



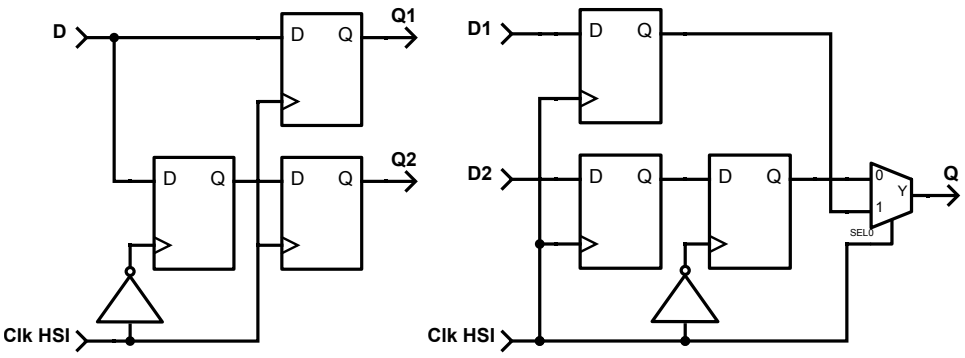
(a) IDDR timing graph [30, p. 94]



(b) ODDR timing graph [30, p. 122]

Figure 5.3: Timing overview of the Xilinx Virtex-6 DDR buffers in SAME_EDGE mode

To make sure the data sent by the FPGA is received correctly by the ρ-VEX and the other way around, this exact same protocol is implemented on the ASIC. See Fig. 5.4 for the circuits implementing this protocol. Implementing them directly as transistors can significantly improve the circuitry. However, the current implementation is good enough and the added complexity of a transistor circuit is not worth it for this project.



(a) IDDR logic circuit (b) ODDR logic circuit

Figure 5.4: Circuits implementing DDR protocol for the ASIC

5.4. SSI abstraction

As of the start of this project, the SSI was not available. To still allow both DC and Encounter to understand what the black box will do and how to work with it, two files are required. A lib file as explained in Section 5.4 will be required for connectivity and timing. A LEF file as explained in Section 5.4.2 is required for physical placement and routing. Furthermore, to ensure correctness of these two files, a simulation is written and explained in Section 5.4.3.

5.4.1. Liberty file

The lib file contains all information related to power and timing. This includes all pins operating voltage, capacitance of input pins and load dependant output pin transition tables. Furthermore, relations between pins are described. This is important for example because a data signal will only start to transition after a clock edge.

Usually, a lib file can be generated using the Cadence Liberate AMS Characterization software. This software can take the IP layout and extract all necessary information. For some reason, however, it was too much effort to include this in the project running in parallel.

The SSI is expected to run at $400\,MHz$, making it the most timing critical interface in the whole ASIC. On top of that, it will generate the clock for the ρ-VEX core. Assuming the clock generation to be ideal (meaning it has zero transition time), propagates this ideality to the whole design. A gate with ideal input will also have ideal output, effectively having zero delay as well. This would result in a clock network that has a zero delay from input to output, effectively making the whole CTS step in the design flow useless. On top of that, the actual delay of the clock network (with 27 stages being about $2\,ns$ in the final design) will be disregarded. Making all software about 20% more optimistic than the real design would! This is of course unacceptable and would result in an ASIC that does nothing at all.

Therefore, against all advice from more experienced designers, the lib file was fully written by hand. All information included in the lib file is summarized in Tables 5.1 and 5.2. The equivalent inverter sizes were determined by comparing the SSI waveforms with those of UMC65 RVT library components. The transition time tables were taken from the respective components and inserted in the lib file, resulting in a somewhat accurate definition.

Table 5.1: Output port maximum constraints

Port name	Description	Max.
CKOUT	Load capacitance	$5\,fF$
DOUT	Load capacitance	$20\,fF$
CK_SYN	Input transition time	$50\,ps$

For every constraint (either in the worst case lib or best case lib), the most pessimistic value was taken. This makes sure the final design will be within bounds and the ASIC will function properly. This rule has two exceptions, the CKOUT and CK_SYN ports as described in Table 5.1. A single element with input capacitance of less than $5\,fF$ simply does not exist and, therefore, this value is doubled.

Table 5.2: SSI port definitions and details

Port name	Direction	Voltage (V)	Equivalent inverter / Capacitance (fF)
CK_FSM	in	1.2	24.8
CK_SYN	in	1.2	17.7
DIN	in	1.2	7.9
EN	in	1.2	5.8
RST	in	1.2	25.8
LVDSCKINP/N	in	-	INVM2R
LVDSDAOUTP/N	out	-	INVM10R
CKOUT	out	1.2	INVM2R
DOUT	out	1.2	INVM2R
LOCK	out	1.2	INVM4R
IREFTX	inout	-	-
IREFRX	inout	-	-
VDDA25	inout	2.5	-
VDDA12	inout	1.2	-
GNDA	inout	0.0	-

Furthermore, the input transition time of $50\,ps$ is so extremely optimistic that DC and Encounter have to put all effort into achieving this. The result are blown up power numbers and an overall badly optimized circuit. After deliberation this constraint is also doubled. The implication is that the SSI will probably not achieve the stated bit rate of $1.25\,Gbit\,s^{-1}$, but that is not at all required anyways.

All available information and added pin relations were implemented in a best case and worst case timing file, required for Encounter. Information on structuring the files was obtained from the Synopsys Liberty User Guide [31]. As information about power consumption is totally absent, this is not estimated either and final power calculations will not include the SSI.

5.4.2. Library Exchange Format

The Library Exchange Format (LEF), is a text file containing an abstraction of relevant physical parameters. From the Cadence reference manual: “LEF defines the elements of an IC process technology and associated library of cell models.” [32]. It defines the size and orientation of a macro for placement, all input, output and inout pins for connectivity and the metal layers used to connect to these pins. Additionally, obstructions of other metal layers can be defined such that some wires can be routed over top the macro (or not at all, if all layers are defined as obstruction). Finally, for each macro pin, gate and diffusion areas can be defined. These areas are important to recognize antenna violations without needing the whole layout.

As the LEF file contains purely physical information, it is very difficult to implement by hand. Initially, an empty layout was used with only a size and dummy pins, slightly representing the final layout. This was used for the floorplanning and placement phases of implementation. CTS and routing were postponed until a first version of the actual SSI layout was ready and exported.

Even in the final version of the LEF file, antenna related information was not included. This means antenna violations can’t be detected and automatically fixed during P&R. Instead, they are only detected in the final layout during verification in Section 9.5.1. Any violations detected there have to be manually repaired. For future versions, it would be beneficial to add this information.

5.4.3. SSI simulation

A description of the SSI in VHDL was also created to allow for full simulation of the interface. The implemented LVDS receivers and drivers are of course analog and ModelSim is actually only capable of simulating digital signals, therefore, the simulation is only an approximation of what will really happen. However, the finite state machine dealing with reset, lock and enable signals (as described in the application note, appendix A) can be tested. This was also used to clear inconsistencies in synchronous versus asynchronous reset and active high or active low signalling.

Besides a digital representation of the SSI, the simulation file also contains VITAL information. VITAL is a standard for annotating timing information on VHDL models for ASIC libraries [33]. As was introduced in Fig. 4.1, it is responsible for adding internal delays to macro pins from an SDF file as can be generated by DC or Encounter. Two types of delay are implemented, internal path delay and propagation delay.

Internal path delay

The internal path delay will be annotated on every input pin and is a delay that is caused by the pins input capacitance and the connected wires capacitance and resistance. See the listing below for an example of the VITAL standard for implementing this delay for port DIN. The delay as exported to the SDF file will be called 'tipd_din', for an internal path delay on port DIN (VHDL is case insensitive).

```
1 WireDelay : block
2 begin
3   VitalWireDelay(DIN_ipd, DIN, tipd_din);
4 end block;
```

Propagation delay

The propagation delay is a delay that will be generated for every input/output relation as described in the lib file. See the listing below for an example. Port DOUT will only switch after a transition on CK_SYN. Two timings are given, 'tpd_ck_syn_dout_posedge' and 'tpd_ck_syn_dout_negedge', for a transition on DOUT after a rising or falling transition on CK_SYN respectively. The posedge timing is taken for a transition on the input delayed version of CK_SYN (CK_SYN_ipd'last_event) after which it has become positive (CK_SYN_ipd = '1'). The negedge for when CK_SYN has become negative. Note that this path delay has a very peculiar addition. A delay of $625\,ps$ has been added to the transition. This delay simulates the $90°$ phase difference incurred by the internal data recovery circuitry, as described in the SSI application note.

```
1  ------------------------
2  --  Path Delay Section
3  ------------------------
4  VitalPathDelay01 (
5    OutSignal     => DOUT,
6    GlitchData    => DOUT_GlitchData,
7    OutSignalName => "DOUT",
8    OutTemp       => DOUT_zd,
9    Paths         => (0 => ((CK_SYN_ipd'last_event / 1 ps + 625) * 1 ps,
       ↪                    tpd_ck_syn_dout_posedge, CK_SYN_ipd = '1'),
10                     1 => ((CK_SYN_ipd'last_event / 1 ps + 625) * 1 ps,
                         ↪ tpd_ck_syn_dout_negedge, CK_SYN_ipd = '0')),
11   Mode          => OnEvent,
12   Xon           => Xon,
13   MsgOn         => MsgOn,
14   MsgSeverity   => WARNING
15 );
```

5.5. Conclusion

The SSI is an externally developed analog IP block, allowing it to be included in the digital design flows of DC and Encounter. It is an interface running at two times or four times the clock frequency of the ρ-VEX core and sends DDR signals over LVDS signal pairs. This allows the main memory of the ρ-VEX to reside on an FPGA external to the ASIC, leaving more chip area for the actual core while still allowing for useful memory sizes.

To use the SSI IP block, a clock divider, a DDR input buffer and a DDR output buffer are constructed. Furthermore, all timing information was manually gathered in the liberate format to be used in DC and Encounter. In a future, more timing critical, version of the ρ-VEX ASIC, it is crucial to generate this lib file with the provided software. It will allow for more accurate timings and actual power estimation.

For physical information needed during placement and routing, a LEF file was provided and explained. Missing from the LEF file, however, was gate and diffusion area size, required for antenna violation detection. Therefore, it is required to separately verify antennae on the final layout and manually correct any detected violation. For automation of this process, a future version of the LEF file should include this information.

Finally, a VITAL file was written for digitally simulating the SSI. The VITAL constructs for internal path delay and propagation delay allow the simulation to be annotated using an SDF file. Therefore, the simulation can accurately detect setup and hold violations.

6

ρ-VEX configuration

One of the highlighted features of the ρ-VEX is its design time configuration. The whole VHDL code is structured, such that minor changes like a stall signal throughout the core can be implemented by changing a single switch. But also major changes as the number of pipelanes available, configuring the amount of instructions that can be executed in parallel. Every such option has an impact on the resulting maximum clock speed, power consumption and/or chip area. Every possible configuration has already been extensively described and, therefore, will not be fully recited here [34, p. 50].

As these are design time configurations, they will also influence the ASIC implementation. Changing any of the parameters requires the whole design flow to be re-run. Therefore, they are discussed before any major design is created and decided upon before doing a full implementation. This chapter will structurally introduce the relevant configuration parameters, give a thorough discussion of its impact on the clock speed, power consumption and chip area and explain the final design choice.

Starting off this chapter is Section 6.1, where the methodology of testing different configurations is explained. Subsequently, the discussed configurations are summarized in Section 6.2. An initial run and the expected impact of every change is listed in Section 6.3, after which the actual results are listed in Section 6.4. Finally, the configurations are considered and a configuration is chosen for implementation in Section 6.5.

6.1. Methodology

There is a big difference in implementing the ρ-VEX on an FPGA or on an ASIC. Therefore, it is not really possible to use any existing quantification readily available from the ρ-VEX. Instead, the initial step of the design flow is set up for ASIC specific quantification. From the design flow, only the synthesis step is performed. The reports generated will have some limitations and are not entirely accurate, but will give a nice indication of what is possible in the final design. Furthermore, a simple synthesis run without thorough optimizations can be run within about two hours and, therefore, allows for relatively quick validation of a hypothesis.

The used synthesis actually already contains a few of the optimization that will be described in Section 8.10. This was required as the very first run resulted in about fifty times worse performance and would mean no design was viable at all. To still keep synthesis relatively quick, it is all done in a single pass (as opposed to the two-pass run for the final design).

6.1.1. Test memory implementation

the final ASIC implementation will need some on-chip cache memory. Ideally this cache is as big as possible. Increasing the caches will however reduce available area for optimizing the core logic. There are conflicting expectations for the two. Therefore, it is decided to run the configuration tests with caches of minimal size required to operate the ρ-VEX. Ideally, the size is increased until the total chip area is used.

Furthermore, the register file will hugely impact the final design. However, as its final implementation will also depend on the quantification obtained in this chapter, it is only optimized in Section 7.5. To still be able to obtain significant numbers, a very naive mapping was made of the FPGA version. This

means the design is expected to be improved quite significantly and mostly relative comparisons should be made.

6.1.2. Design constraints

For a synthesis run, some initial constraints are required. These include the maximum chip area and a clock speed. The whole purpose of this chapter was to determine what can be achieved for chip area and clock speed. Therefore, the design is somewhat over-constrained. For maximum chip area, zero is given such that the design is fully optimized for minimum size. As target clock speed, $1\,GHz$ (or a clock period of $1\,ns$) is set. This will force DC to hugely optimize for clock period and eventually even fail to do so. As there is no timing slack left, power optimizations will not be performed. Therefore, the reported power is purely theoretical and have only relative significance. The resulting minimum clock period will be calculated as the $1\,ns$ - WNS. For the final design, a slightly larger clock period should be taken to not constrain the design as hard and allow for power optimizations.

6.2. Design configurations

All possible ρ-VEX configuration parameters have been discussed and were evaluated. In accordance with (**G**.1), only a prototype is being developed and, therefore, implementing every possible feature of the ρ-VEX is not relevant. On the other hand, the main feature of the ρ-VEX is its ability to adapt to different program loads at run-time. This functionality will need to be kept in the ASIC as much as possible.

Following these simple rules, the parameters as listed below have been chosen for evaluation. An overview of all configuration designs is given in Section 6.2.9.

6.2.1. Breakpoints

In the full ρ-VEX configuration, there are four breakpoints possible. These breakpoints are useful when debugging software. As this is only the first ASIC prototype of the ρ-VEX, a mistake might happen. Software debugging will help in finding exactly where the mistake is made, allowing for correction conform (**G**.4b). Although the ρ-VEX can house four software breakpoints, only one is strictly necessary and practically used. Therefore, the first design prune will involve reduction of the number of breakpoints from four to one.

6.2.2. Trace unit

The trace unit is another tool for debugging. It allows the ρ-VEX to report information about its program execution state, registers and caches. Later on in Section 9.3, a different verification method is described specifically useful in an ASIC. This method is more flexible in its use (although probably slower). Therefore, all information obtained from the trace unit will be somewhat obsolete and it can be left out if required.

6.2.3. Performance counters

The performance counters are a set of registers dedicated to a specific task. For example, cycle counters and cache miss counters exist. The performance counters are to be read by software running on the ρ-VEX. To save hardware, the size of these counters is configurable, or they can be disabled all together. In the ‘maximum’ configuration, this configuration is set to 4 (even though the maximum is 7), resulting in 32-bit wide counters. If left out altogether, some hardware debugging using software cannot be performed.

6.2.4. Bundle alignment

As the ρ-VEX issues long instruction words, every bundle of instructions has to be of a specific length (filling every pipelane). If less instructions are to be executed, the bundle read from the instruction cache has to be properly aligned, incurring some extra hardware. This hardware can be saved by forcing the software to be aligned in the compiler (using, for example, NOP instructions). Therefore, this is a trade-off between software program size and hardware size.

6.2.5. Lane multipliers
By default, every pipelane has its own multiplier. This means that a configuration with eight pipelanes has eight pieces of hardware capable of performing the same operation. Leaving out some multipliers doesn't necessarily decrease functionality (as there are still some multipliers), but forces instructions to be only able to execute in specific pipelanes. This restriction will only slightly decrease the performance of the ρ-VEX.

6.2.6. Hardware contexts
A context is the full state of a thread running on the ρ-VEX. By default, four contexts are configured. This means four programs can be executed in parallel. By halving the amount of contexts, only two programs can be executed. This limits the functionality and, therefore, is to be avoided if possible.

6.2.7. Lane groups
Lane groups are the part of the processor that allows for reconfiguration. The amount of pipelanes divided by the number of lane groups defines the minimum configuration size. Therefore, reducing the number of lane groups severely limits the entire purpose of the ρ-VEX. Changing this parameter would essentially mean fabricating the ρ-VEX ASIC will be useless.

6.2.8. Pipelanes
If the number of lane groups is reduced, the number of pipelanes (instruction execution units) can be reduced as well. This only reduces the performance, but is not possible without also reducing the lane groups. Therefore, it is the least favourable pruning method.

6.2.9. Configuration overview
All possible design prunes have been discussed in order of how favourable they are. All prune steps are listed in Table 6.1.

Table 6.1: Possible ρ-VEX design configurations

Design	Brkpts	Trace	Perf. cntr.	Bundle align	Multipliers	Contexts	Groups	Pipelanes
Maximized	4	yes	4	yes	8	4	4	8
Prune1	1	yes	4	yes	8	4	4	8
Prune2	1	no	4	yes	8	4	4	8
Prune3	1	no	0	yes	8	4	4	8
Prune4	1	no	0	no	8	4	4	8
Prune5	1	no	0	no	4	4	4	8
Prune6	1	no	0	no	4	2	4	8
Prune7	1	no	0	no	4	2	2	8
Prune8	1	no	0	no	4	2	2	4
Minimized	1	no	0	no	2	2	2	4

6.3. Expected impact
To estimate the impact of every pruning method, an initial synthesis is performed on the maximized design. Using an auxiliary script (Appendix E.1), all library cell references are hierarchically extracted and total area is calculated. Only the largest areas are listed, many small elements are ignored. See Table 6.2 for the full design and Table 6.3 for a detail overview of element 'core_pipelane_1_1_0_0_1_1'.

Note that all SRAM cells used for the register file and caches are specifically separated, as they are an order of magnitude larger than all other logic. Furthermore, the register file (GPRF and gpRegs), is the single largest instance in the design. The inefficient layout makes use of 128 (=16 read ports · 8 write ports) SRAM cells. This will later be somewhat improved, in Section 7.5. Therefore, reducing the number of read and write ports will have a significant impact. Prune6 will half the number of contexts and with it the number of registers. It is expected this will have about $\frac{1}{2} \cdot 69.01 = 34.5\%$ reduction in area. Similarly, prune8 halves the number of read ports and and write ports, meaning only $8 \cdot 4 = 32$

GPRF instances are needed. This will reduce the total area even twice as much.

Next, the caches are quite large. One instance is needed per lane group. Therefore, when implementing prune7, halving the number of lanegroups will half the total cache size. Therefore, this will have about a 7% area impact, already becoming a lot less significant.

From Table 6.2, we can easily see that the trace unit is 0.86% of the total area, removing it in prune2 will save about that much as well. Furthermore, in Table 6.3 we can see that a multiplier is about 0.15% of the chip area. Therefore, reducing the amount of multipliers by four will only have about a 0.6% impact on area.

Table 6.2: Initial synthesis of maximized ρ-VEX, size broken up into elements

Design element	Element area (μm^2)	Count	Total area (μm^2)	Relative area
GPRF	18578.04	128	2377988.86	69.01%
Instruction cache	16407.19	4	65628.77	9.44%
Data cache	53041.94	4	212167.74	6.16%
core_gpRegs_	96858.72	1	96858.72	2.81%
core_contextRegLogic_	92390.04	1	92390.04	2.68%
Cache logic	43500.25	1	43500.25	1.26%
core_pipelane_3_1_0_0_1_1	37103.04	1	37103.04	1.08%
core_pipelane_5_1_0_0_1_1	36919.44	1	36919.44	1.07%
core_pipelane_7_1_0_0_1_1	35211.96	1	35211.96	1.02%
core_pipelane_1_1_0_0_1_1	34657.20	1	34657.20	1.01%
core_trace_	29755.08	1	29755.08	0.86%
core_pipelane_6_1_1_1_0_1	21916.08	1	21916.08	0.64%
core_pipelane_4_1_1_1_0_1	21889.44	1	21889.44	0.64%
core_pipelane_2_1_1_1_0_1	21887.64	1	21887.64	0.64%
core_pipelane_0_1_1_1_0_1	21816.36	1	21816.36	0.63%
core_contextPipelaneIFace_	19213.20	1	19213.20	0.56%
core_instructionBuffer_	6531.12	1	6531.12	0.19%
core_globalRegLogic_	4734.00	1	4734.00	0.14%
Registers (DFQM1RA)	6.84	470	3214.80	0.09%
core_cfgCtrl_	2613.96	1	2613.96	0.08%

Table 6.3: Initial synthesis of maximized ρ-VEX, one pipelane, size broken up into elements

Design element	Element area (μm^2)	Count	Total area (μm^2)	Relative area
DFQM1RA	6.8400	1008	6894.7200	0.20%
core_mulu__1	5055.4801	1	5055.4801	0.15%
OA22M12RA	9.3600	504	4717.4400	0.14%
core_alu__1	4661.6401	1	4661.6401	0.14%
DFQM2RA	6.84	454	3105.3600	0.09%
AOI22B20M8RA	7.2	331	2383.2000	0.07%
AO22M8RA	6.48	293	1898.6400	0.06%
AOI22M1R	2.52	244	614.8800	0.02%
INVM40R	10.08	55	554.4000	0.02%
INVM1R	1.08	372	401.7600	0.01%

6.4. Configuration results

Finally, all designs are synthesized as described. The designs with resulting timing, energy consumption and area are listed in Table 6.4. The improvements with respect to the previous implementation is also listed.

As expected, the most significant area improvements lie in the register file reduction. Prune8 reduces area by 63.5% (expected was 70%) and prune6 by 27.5% (expected was 34.5%). The actual numbers are slightly less due to a routing estimation DC adds to the total area.

On the other hand, removing the trace unit in prune2, results in an area reduction of 1.2%, more than the expected 0.86%. This is because the trace instance as listed in Table 6.2 is only part of the logic. Throughout the whole core, wires are connected to the trace unit, which can subsequently be optimized away. This has the additional side effect of an improvement in clock period of almost 40%, arguably being more important than the area reduction. Similarly, reducing the number of breakpoints in prune1 affects a lot of routing. Both reducing some area and improving the clock a lot.

One interesting side note is the design of prune4. Within the noted significance, there is no difference in design at all. The synthesis tool is not entirely deterministic and probably due to sheer chance it got a very similar result. The most important lesson here is that the bundle alignment hardware is so insignificant that it can be kept in no matter what.

A reduction in area is usually paired with a reduction in energy. This is as expected, reducing the number of logic elements also reduces the wiring. Together reducing static and dynamic power consumption respectively. A reduction in clock period, on the other hand, reduces the static power per Hz (as the leakage power is multiplied by a shorter period) and simultaneously increases dynamic power. The resulting energy difference totally depends on how well DC is able to optimize.

Prune5 and prune7 show a small increase in clock period. Again, this can be entirely due to the non-deterministic nature of the synthesis tool and is a totally reasonable difference. All other results are not significant in choosing the configuration.

Table 6.4: ρ-VEX design synthesis results

Design	Clock (ns)	Energy ($\mu W\, MHz^{-1}$)	Area (mm^2)	Clock diff.	Energy diff	Area diff
Maximized	14.72	2359.00	3.0617	-	-	-
Prune1	11.61	2475.69	3.0036	-21.13%	4.95%	-1.90%
Prune2	7.10	2439.95	2.9679	-38.85%	-1.44%	-1.19%
Prune3	7.03	2418.50	2.9051	-0.99%	-0.88%	-2.11%
Prune4	7.03	2418.50	2.9051	0.00%	0.00%	0.00%
Prune5	7.57	2414.56	2.8823	7.68%	-0.16%	-0.79%
Prune6	2.83	2071.80	2.0885	-62.62%	-14.20%	-27.54%
Prune7	2.88	2061.80	2.0539	1.77%	-0.48%	-1.66%
Prune8	2.67	1415.59	0.7495	-7.29%	-31.34%	-63.51%
Minimized	2.57	1413.26	0.7364	-3.75%	-0.16%	-1.74%

6.5. Final considerations and configuration

From the results obtained, a design was chosen. The four major decision points were the area available, the register file, the trace unit, and number of breakpoints. Furthermore, the targeted speed is discussed.

6.5.1. Available chip area

The total chip area for the used MPW run of IMEC is, as stated before, $1875\,\mu m \cdot 1875\,\mu m$. Removing the area needed for the IO ring resulted in a total logic area of $A = 2.7\,mm^2$. Taking into account the fact that the SSI is not yet calculated in (as the final design was still unfinished prior to these estimations), this area will be even smaller. All designs up to and including prune5 are already larger than this and, therefore, are not expected to fit the chip area. Even though it would limit the functionality, reducing the hardware contexts from four to two would be necessary to fit the design.

Later on in the project the register file got significant improvements (see Section 7.5). This allowed the design including the four hardware contexts to be implemented, reverting the prune.

6.5.2. Trace unit
As was shown, removing the trace unit result in an almost 40% improvement in speed. As most of its functionality will later be replaced by ASIC specific hardware, it is left out of the final design.

6.5.3. Breakpoints
The number of breakpoints was already discussed. In practice, all four breakpoints are never used. One is certainly enough and allows a 20% clock speed improvement. Therefore, the final design will only have one breakpoint.

6.5.4. Target clock speed
After removing the trace unit and reducing the number of breakpoints, the design was able to achieve a clock period of about $7\,ns$. This was, however, with a greatly over-constrained design, as discussed in Section 6.1.2. DC will put all effort in optimizing clock speed, while neglecting the power consumption. This conflicts with the main goal of low power consumption, (**G**.5b). Therefore, the final design will need to be constrained less. Targeting a clock period of $10\,ns$, or $100\,MHz$, allows the design to be maximally optimized for power and leaves some slack for the P&R tools as well.

6.6. Conclusion
The ρ-VEX design is extensively investigated before implementing it as an ASIC, as expected for (**G**.3). One of ρ-VEX's features is its design time configuration flexibility. A test run is set up and used to compare the different options. From these initial reports, a design that will finally be implemented is chosen.

Initially, the design was expected to need some pruning. The best option is reducing the number of hardware contexts from four to two, limiting the amount of threads the ρ-VEX can run in parallel. This would allow the ASIC to be implemented on the smallest and cheapest chip area, as is preferred for (**G**.1). Later on in Section 7.5 the register file will be improved and reduced in area. This allows all four contexts to be implemented.

To significantly improve the ASIC clock speed, the number of software breakpoints has been reduced from four to one. As more breakpoints are rarely used, this is not a problem. Furthermore, the trace unit has been entirely removed, significantly reducing the logic connectivity and improving the clock speed further.

Aiming for (**G**.5b), low power design, the clock speed is constrained at $100\,MHz$. This should be easily achievable during synthesis and allows some slack during P&R. The constraint is very loose, allowing for more focus on reducing power consumption.

7

Design overview

Prior to this project, the ρ-VEX was implemented entirely on an FPGA. Implementing it on an ASIC instead requires some additional changes to the netlist. One such changes is the use of an external memory, as was already discussed in Chapter 5. Section 7.1 will list the changes required for actually implementing it.

As this design is only an initial prototype, the final results are very difficult to predict. However, ensuring at least something works is very important. Therefore, Section 7.2 deals with (**G**.4). As was also discussed, the SSI will be able to operate either in $400\,MHz$ or in $200\,MHz$ mode. In addition to the two operational frequency modes, conform (**G**.4a) and (**G**.4b), two operational modes are added and discussed in Section 7.3.

The FPGA initially used for the ρ-VEX has dedicated memory components for the caches and register file. These components are not available when the ρ-VEX is implemented as an ASIC. Section 7.4 will explain the need for a data and instruction cache and explain how the FPGA specific parts are replaced. The register file is quite complicated to implement due to the need of 24 access ports. Multiple solutions are expanded upon and an implementation is substantiated in Section 7.5.

7.1. Communication interface

Most of the implementation details regarding the communication interface have been discussed during both projects. The relevant information can mostly be found in the appendices, as they are developed outside this project. Details about the SSI are listed in appendix A. How it is implemented in the ρ-VEX, or can be implemented in any project alike, are repeated and discussed here. The communication protocol around the interface, as implemented in the ρ-VEX, is called the High Speed Interface and is listed in Appendix B.

7.1.1. SSI signal connections

Internally, the SSI has a Finite State Machine. This FSM ensures the signals are timed correctly and all data will be properly received. For this FSM, the SSI expects a reset signal prior to any operational mode and an enable signal to start ‘locking’. The locking process is expanded upon in Section 7.1.2.

Both the reset and enable signals are synchronous active high. This is in contrast to the asynchronous active low design of the HSI and almost every other ASIC implementation. As it is so standard to do so, this discrepancy was apparently never really discussed. Nevertheless, it is easily fixed in the netlist. In a future design, it might be beneficial to use the same type, as to prevent these kind of implementation mistakes.

In the final implementation of the HSI, four SSI instances are used. The four instances are used for four data receivers and four data transmitters. However, they also include four clock receivers. As the ρ-VEX ASIC only requires a single clock, the other three clock receivers are disabled by connecting them to a $2.5\,V$ supply, saving two IO pins per instance. Each instance still requires two reference current sources, these can’t be shared.

7.1.2. SSI locking

For correct data recovery, the SSI requires a calibration. This is achieved by enabling the FSM while applying an alternating signal (101010...). After an SSI is calibrated correctly, it outputs a lock signal. Only after all SSI instances are locked, are they allowed to receive data and is the ρ-VEX enabled. The calibration pattern and lock signals are also implemented in the netlist.

7.1.3. HSI pin sharing

Initially during the project the SSI was expected to use a lot of IO pins, more than available. Therefore, a lot of effort was put into reducing the amount of digital pins the ρ-VEX would use itself. This pin sharing scheme is provided in the netlist and an overview is given in Appendix B. Later on, the SSI worked out to be far less demanding, leaving many pins unused. As the pin sharing was already implemented, it is still used, even though it greatly over-complicates the digital communication. In a future project a decision can be made. Either keep the complicated pin sharing and use the remaining IO pins for extra functionality or simplify the design. Extra functionality could include debugging,

7.2. ASIC debugging

One of the primary design goals was to ensure a useful design, (**G**.4). Therefore, some logic was introduced to debug the design, single out the mistake and possibly correct it. Even though the verification is quite extensive (see Chapter 9), an error might still slip through. Therefore, it is essential to know what went wrong such that it can be prevented in a future iteration. Besides a robust design, four implementation changes are thought out to meet these goals.

7.2.1. UART interface

By default, the ρ-VEX includes an UART interface. This is a slow bi-directional communication interface using three digital IO pins. On the ASIC, it is still possible to use this interface. This allows the HSI and the SSI in particular to be bypassed. If there is any problem with that design, it is easy to detect by testing in UART mode. However, other than knowing there is something wrong, the UART does not allow further debugging. It does still allow the ρ-VEX to be used, albeit with a much slower cache.

7.2.2. Scan chains

Scan chains are a very common way of automated IC testing in the industry [7]. See Fig. 7.1 for an example scan flip-flop. Essentially it is nothing more than adding an extra signal Scan Enable (SE) for overriding the normal input. Instead of Data (D), a controlled Scan Data (SD) can be asserted to the flip-flop. These elements are in fact so standard that they are included in just about every standard cell library, including the used UMC FDK.

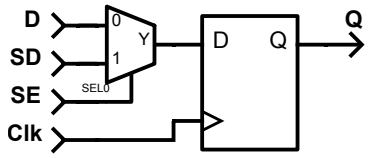


Figure 7.1: Scan flip-flop as basic element in a scan chain

The scan flip-flop can be used to detect faults in very complicated logic. See Fig. 7.2 for an example circuit. Note that when this circuit is produced, only the inputs A, B and C are controllable, while only output D can be observed. A fault can occur in any logic gate, wire or even the input or output can be broken. Normally, it is impossible to know where the fault is. However, by using scan flip-flops, all elements can be connected serially without other logic in between. This creates the data path indicated with a dashed line. As long as there is no fault in the constructed scan chain, each flip-flop can now effectively be used as an input and output. Not being limited to the inputs and outputs A, B, C and D anymore, the whole circuit can easily be verified.

Usually, scan-chains are used for automated and quick testing of an IC. However, they additionally allow for full access to every element in the circuit. With a bit of effort, this allows debugging the IC and even using parts of the circuit circumventing the rest.

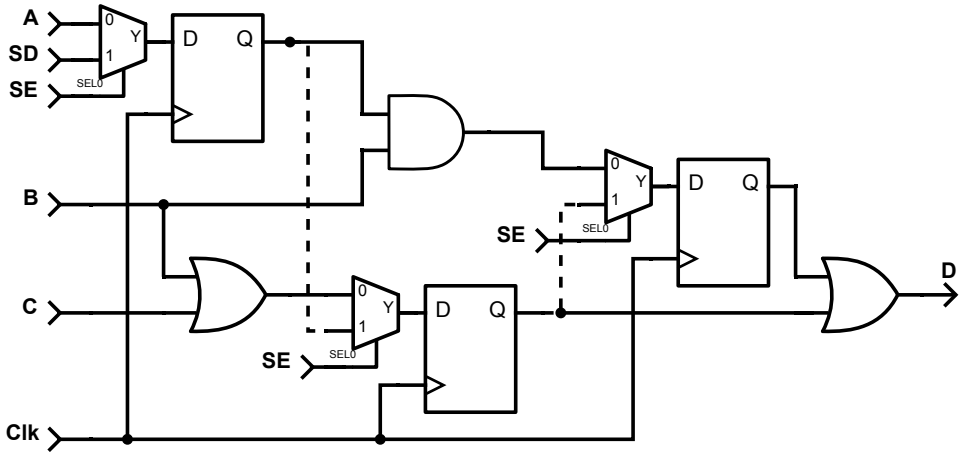


Figure 7.2: Scan chain for serial connection of several flip-flops

To ease debugging, multiple shorter scan-chains are implemented. Each chain can be accessed individually through a multiplexer. This allows changing part of the chip, without affecting the rest. Furthermore, it evades the need of shifting data through every flip-flop in the design, greatly speeding up the process. As each scan flip-flop is still connected to the same clock, the non-accessed elements are connected in a loop. Therefore, all data is shifted round, eventually ending up with the same data as before the shift. This is only possible if all chains are of equal length, putting a small constraint on the multiple chain design.

Some of the chains are also chosen very specific. For example, the whole register file is connected as separate scan chains. For every context, one part of the register file is an individual chain. This allows for quick reading and writing of any register in the design. It can also be used if the normal interface to the ρ-VEX is broken, such that the internal logic can still be verified.

Another option is to add a scan chain for design configuration. All data can be shifted in to some logic controlling flip-flops. Depending on the state of these flip-flops, the functionality can change. This would be used for below two configurations.

7.2.3. Standalone mode

The HSI is used to allow for off-chip memory. As a fallback, the UART mode was introduced. However, both modes still require a communication interface to constantly work and have an FPGA connected with the main memory. It would be nice to allow for a standalone mode, where this dependency on external controls is not necessary.

The ρ-VEX still has some on-chip cache in the form of an SRAM cell with additional logic. However, it would be possible to access this SRAM cell as direct mapped memory, by bypassing the cache logic. This can be as easy as adding a multiplexer and configuring the mode, for example with a scan chain. This removes almost all dependencies on external interfaces. See Fig. 7.3.

7.2.4. ROM

One of the disadvantages of the standalone mode is that a program still has to be loaded into the SRAM. If it is not possible to access the ρ-VEX at all through the regular interfaces, this mode is still useless. Therefore, a ROM can be added in addition. This ROM already contains a predefined program and will be run by default. The result can be stored in the register file, which is already accessible through the scan chain. This removes all interface dependencies and only requires the scan chain to work.

To go even further, a single IO pin can be used to signal a result. Applying power to the ASIC starts the program loaded in ROM and signals its state through an LED. As long as the LED turns on, the ASIC still works. This full standalone mode might be useful in radiation tests (needed for (**G**.5)), such that it is certain only the ASIC can fail.

Both the standalone mode and ROM are not implemented in the current design. For actual Independence of external interfaces, some integral changes have to be made. These changes would be too time-consuming, taking into account the extra testing required. Furthermore, as will be clear in

Chapter 8, there would not be enough space left to implement a ROM and the current SRAM size is too small to load any useful program.

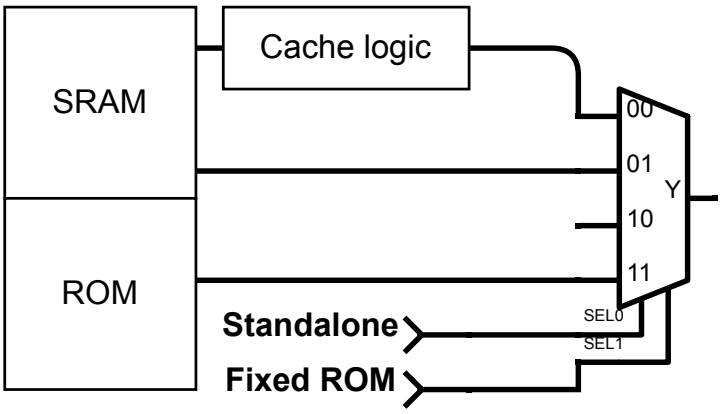


Figure 7.3: Standalone mode, allowing the cache to be used as direct mapped memory

7.3. Mode of operation

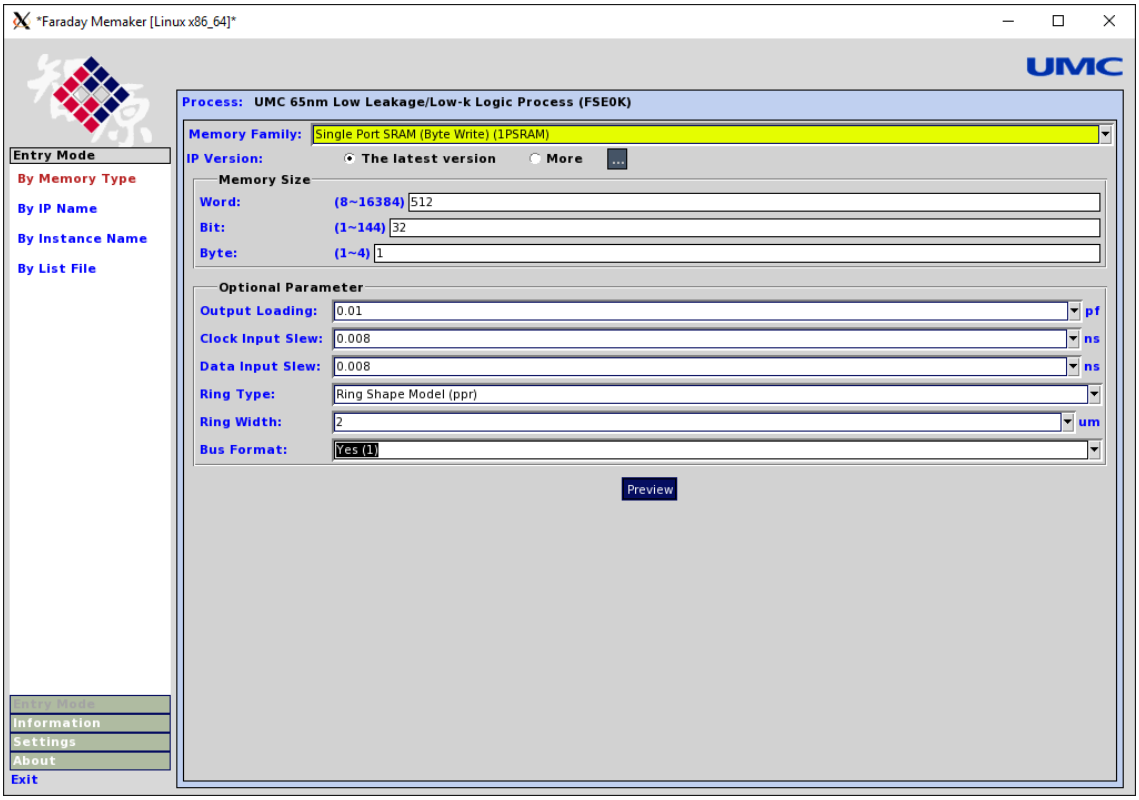
Currently, there are four modes of operation defined. The ρ-VEX can run in normal operational mode for the intended functionality. This can be in fast or in slow mode. The fast mode configures the internal clock divider to divide four times. Allowing the HSI to operate at $400\,MHz$ and the core at the regular $100\,MHz$. In slow mode, the clock divider will divide two times and the HSI runs at a reduced $200\,MHz$. The core will still run at the same speed.

The remaining two modes are for debugging and verification. First, the ρ-VEX can be configured to communicate over UART. Both the UART and core run at the same speed, using a clock applied at a backup pin. See Appendix B for the specification. This mode allows for full bypassing of the HSI. Problems in the SSI, clock division or cross-clock domains are circumvented. If the ρ-VEX still doesn't operate, scan mode can be enabled. In scan mode all flip-flops are connected as serial scan-chains and there is full controllability and observability of every logic element in the design. The HSI is disabled in the same way as UART mode (in fact, both modes are almost identical). One scan chain can be selected through the regular interface, after which data can be shifted in and out through dedicated pins.

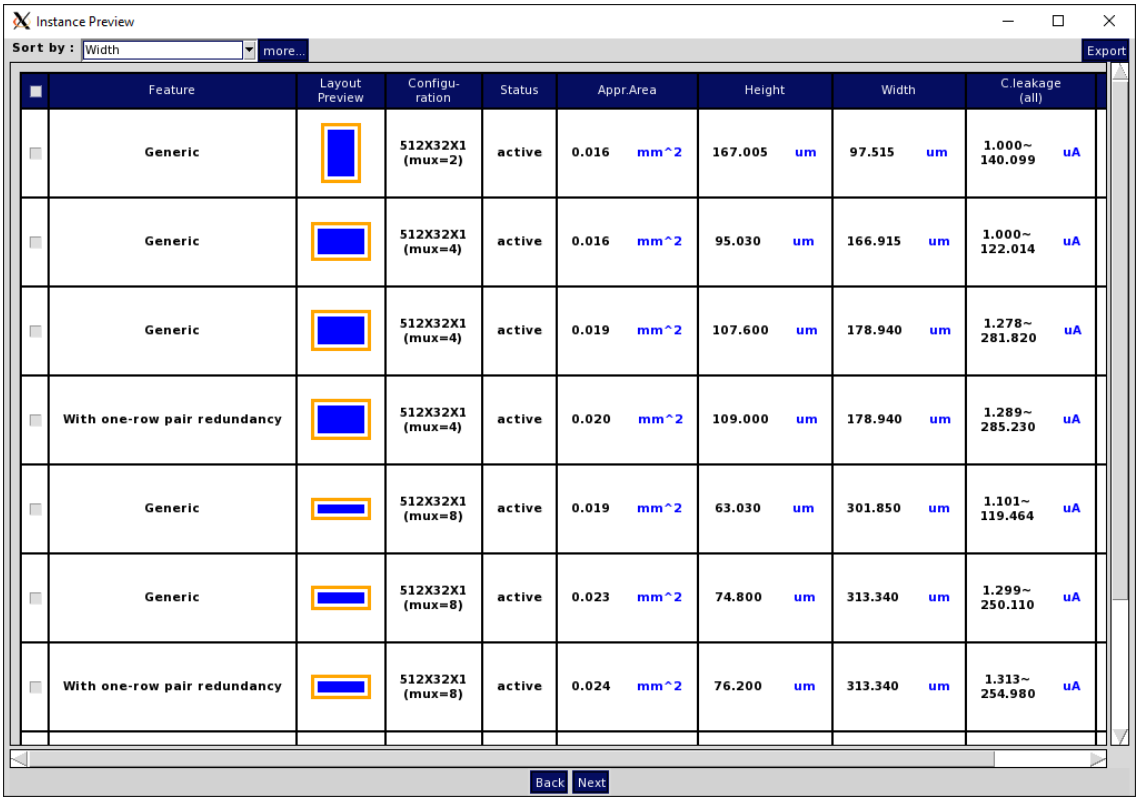
Clearly, each of these modes implement different clock speeds or disable some circuitry. This puts different constraints on (parts of) the design, depending on the operational mode. Therefore, the software tools are configured to recognize this behaviour and perform analysis accordingly. This is done in Section 8.2.5.

7.4. Data and instruction cache

As already introduced, the data and instruction cache are implemented as SRAM cells. These cells can be generated of any desired memory size using the program memaker, see Fig. 7.4. The program is capable of generating single port, two port and dual port memories. A single port memory has a single address decoder, allowing reading or writing to one word at a time. A two port memory has two address decoders. It is capable of writing to one word while simultaneously reading another word. The dual port memory is capable of independently reading from or writing to two different words. A limitation of the memaker is that a word can be at most 144 bits wide.



(a) Settings overview



(b) Different configurations

Figure 7.4: Faraday memaker to generate UMC65 SRAM cells

7.4.1. Common cache parameters

Both the instruction and data cache consist of three parts. A data storage, a tag storage and valid bits. The data storage requires one read/write port, so a single port SRAM is enough to replace it. The tag memory, however, requires one read port and either a read or write on the second port. Therefore, a dual port SRAM is needed. The valid bits require a single cycle reset on every bit, which is impossible using SRAM. Therefore, it is implemented using standard logic cells.

Most of the other options are left to their default, as given in Fig. 7.4a. The second step, in Fig. 7.4b, allows choosing a specific implementation for a certain SRAM cell. Using the mux, it is possible to scale the cell from a square to a few different sized rectangles. This is mostly useful when fitting the element on the floorplan and can be changed later during P&R.

The single port SRAM allows for a row-redundancy implementation [35]. Using a select line, part of the SRAM can be disabled and another part enabled. This allows a broken piece of memory to be 'repaired', without needing a new IC. The dual port SRAM has another option called Build In Self Test (BIST) [36]. The BIST checks for any defects within the SRAM and automatically replaces broken pieces with the built in redundancy. Both options are useful for improving yield (redundancy allows 'broken' chips to still be used) or lifetime improvement (BIST allows 'broken' chips to repair themselves). As these are not needed for a prototype, they are not used. For a future design they might be interesting.

7.4.2. Data cache implementation

The data storage part of the data cache requires read and write ports of 32 bits wide. This is a single word for the ρ-VEX. For the minimal implementation 512 of these words are required. According to these 512 data words, the data tags are 21 bits wide. Both can be easily constructed from the default SRAM.

7.4.3. Instruction cache implementation

From the instruction cache, each pipelane must be able to access an instruction every cycle. There are eight pipelanes and an instruction is 32 bits wide. This means the data storage requires a word size of in total $8 \cdot 32 = 256\,bit$. This is more than the memaker limited maximum word size of 144. Therefore, the ASIC implementation has the data storage of the instruction cache split in two elements, allowing two words of 128 bits to be extracted simultaneously. This doesn't require any other change to the ρ-VEX code. It does, however, have some impact on the final design. As the SRAM cells are totally split, the address decoders need to be duplicated. This has some negative impact on area and power.

For the instruction cache tags, a word size of 19 is needed. This can again be easily implemented using standard memaker configurations.

7.5. 24-port register file

In the configuration as discussed in Chapter 6, this ρ-VEX implementation will have eight pipelanes. From the Instruction Set Architecture (ISA), it is possible to find instructions using two input registers and one output register. This means up to a total of $8 \cdot 2 = 16$ registers need reading and $8 \cdot 1 = 8$ registers need writing. Hence the register file needs 24 access ports.

For most designs the register file is implemented using SRAM cells, like the data and instruction cache. In fact, a special 'register file' can be generated, which is an SRAM cell optimized for the (generally) smaller memory size [37]. However, this register file cell has only a single read/write port. The dual port SRAM has two read/write ports and in fact, that is about the most a generalized SRAM cell will ever go. Not even close to the required 24 ports.

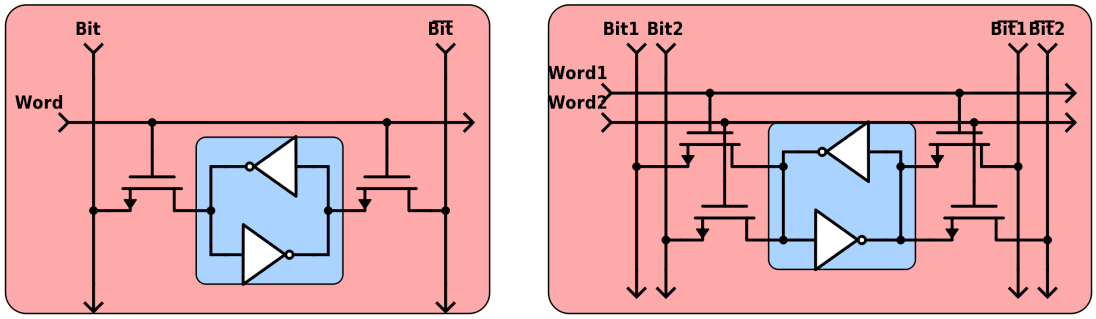
Note that the same problem arises when implementing the ρ-VEX on FPGA. There the problem was solved using the configuration as shown in Section 7.5.3. The naive implementation as used for the area and speed test in Chapter 6 also used this configuration. However, as an ASIC is not limited to any available hardware, a more elaborate solution can be constructed. This section will show different and better alternatives.

7.5.1. N-port memories

Before discussing the different register file architecture, it is useful to generalize the problem to an N-port memory. For the comparison below, the circuitry will be split in read circuitry, write circuitry, bit cell and bit access. As an example, the SRAM cell is taken.

See Fig. 7.5a for the bit cell and access lines. 6T (six transistor) SRAM bit cell (indicated in blue) is composed of two inverters. These two inverters are connected front to back, such that they amplify each others state and any value will be maintained. The other two transistors are activated by a word line, selecting all bit cells in the same horizontal row (word). By taking the correct bit (and inverted version $\overline{\text{bit}}$) lines, the value of the cell can be accessed. Selecting the correct word and bit lines is done using an address decoder. Writing is done using a differential write driver and reading is done using a differential sense amplifier. These last three parts are unimportant for this work, but more information can easily be found [38].

The described SRAM is for a single access port. To create a two port SRAM, only the bit access circuitry needs to be duplicated. See Fig. 7.5b. Therefore, this can be added without changing the bit cell. Note that when a dual port SRAM is required (with simultaneous read or write on both ports), the read and write circuitry needs to be doubled as well.



(a) Single port SRAM bitcell

(b) Two port SRAM bitcell

Figure 7.5: The difference an access port makes in an SRAM cell

The following six sections describe different implementations of an N-port memory. Each of the implementations is quantified by its read and write circuitry, bit cell and bit access. Finally, Section 7.5.8 gives a comparison of the different implementations.

7.5.2. Banked memory

Instead of using one SRAM cell for all registers, the registers can be split among one or more cells. When a specific register is requested, some logic determines in which cell it is stored, such that the correct data is accessed. The split SRAM cells are called banks, hence the banked memory. See Fig. 7.6.

Each of the banks requires only one access port, while the total memory can have N access ports. As long as each port requests a register from a different bank, no problems occur. However, when one bank gets multiple accesses, the arbitration logic has to resolve the conflict somehow. [39] gives a generalized overview of a banked register file and arbitration logic.

A big downside of the banked memory is that a conflict generally results in a pipeline stall or flush, degrading the processors performance. Other work tries to predict and prevent conflicts [40] or resolves write conflicts using register renaming [41]. In [40] the pipeline is flushed to achieve a lower power consumption.

The banked memory is very common in superscalar processors. Although the architecture is different from a VLIW processor like the ρ-VEX, the problem is the same. Multiple instructions get executed in the same clock cycle, therefore, a multiported register file is needed. Generally, the banked register files go up to 12 access ports, but 24 access ports should be possible as well.

The results of a banked memory greatly depend on the amount of conflicts. This again depends on the results of both the bank organization (amount of banks, which register is in which bank) and the software compiler effort, preventing any conflicts. The banked memory is somewhat more complex to make, up to very complex for efficient designs.

However, if an efficient design is implemented, the speed can be good (good enough for modern superscalar processors). Generally, the arbitration logic is implemented as pipeline stage, which is no problem for deeply pipelined designs. As the ρ-VEX has a very shallow pipeline, performance impact will be somewhat worse. The area of the bit cell is the same as for a standard SRAM, as no extra storage is required. There is some overhead in duplicated access logic and the extra required arbitration logic. Power consumption is very low due to the small extra logic and ability to power down individual banks that are unused. All in all good results really depend on the effort, making the design somewhat difficult to scale.

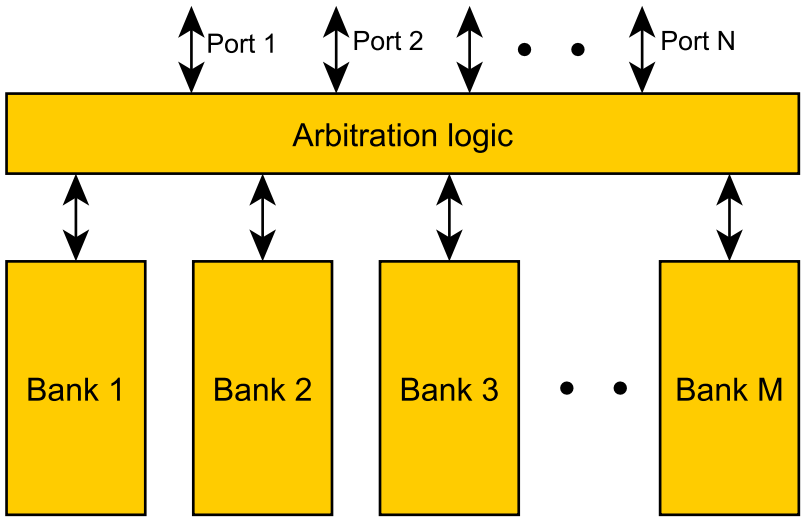


Figure 7.6: Banked memory architecture for N-port memory

7.5.3. Replication

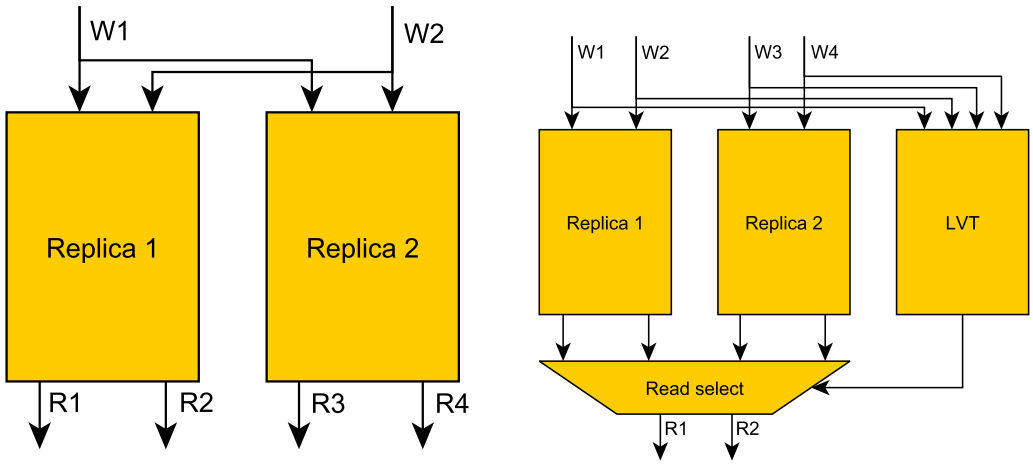
Memory replication is the method implemented for the FPGA version of the ρ-VEX and many FPGA processors alike [42][43][44][45]. Two separate methods are used for increasing the amount of read ports and increasing the amount of write ports. See Fig. 7.7.

For multiple read ports, the whole register file is simply replicated, Fig. 7.7a. A write on port 1 (W1), is written to both SRAM cells. The same happens for a write on port 2 (W2). A read can now happen on any of the four read ports. As the data in both SRAM cells is the same, the read result is the same, no matter from which replica it came. Effectively, the amount of read ports is doubled.

For multiple write ports, a Live Value Table is required, Fig. 7.7b. When a write happens on replica 1, a bit is stored in the LVT indicating the latest write to a register happened in this replica. The same is done for every write port for all replicas. When a read happens, the LVT indicates which replica holds the latest (live) value, such that the correct value is read. The LVT can easily be constructed from standard cells, as it is relatively small compared to the SRAM.

Note that both methods require a doubling of the full storage when a doubling of either read or write ports is necessary. This doubling increases both the bit cell area and the access lines. For an increase in write ports, the LVT makes the area grow even more. Furthermore, for many access ports quite a lot of wires are required. Unlike the banked memory, SRAM cells can't be powered down as effectively, as most are written simultaneously. The total wiring plus SRAM cells lead to a huge power consumption, as well as very slow operation. On the other hand, this implementation requires almost no hardware other than (many replicas of) an SRAM cell. Making it very easy to implement (and hence very popular on FPGA).

A lot of effort in other work is focused on reducing area by improving or replacing the LVT [43][46][47] or by investigating access patterns [48]. The register files speed can be improved by pipelining the added wiring and logic [45]. However, all improvements are mediocre and increase the complexity.



(a) Replicated read port

(b) Replicated write port

Figure 7.7: Replicating memory to create more access ports

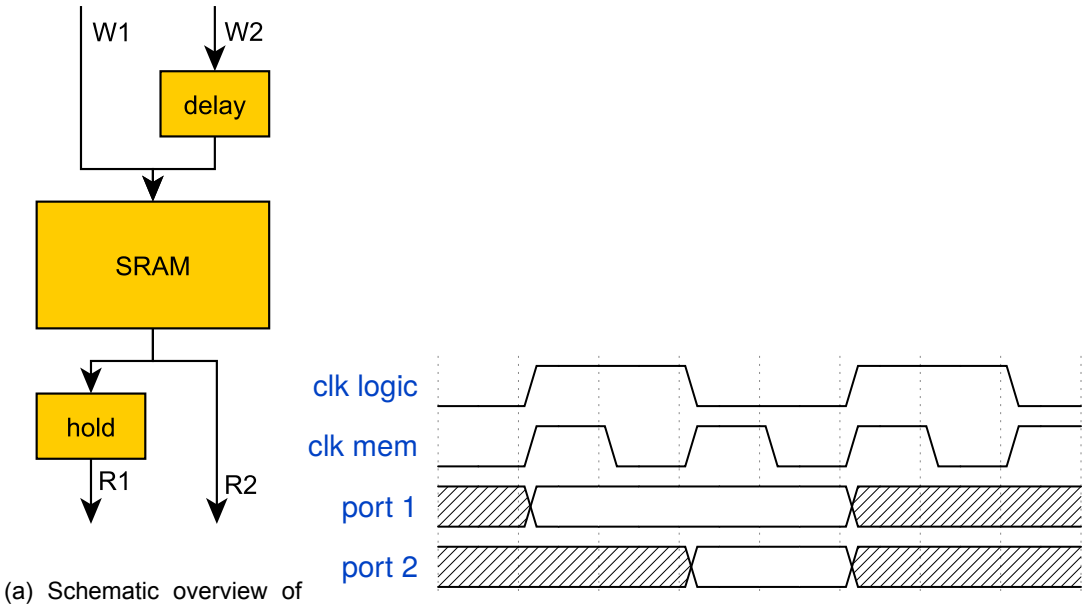
7.5.4. Multi-pumped memory

Multi-pumping is a design methodology to implement resource sharing. By running part of the logic at a multiple of the global clock frequency, that logic part can execute multiple operations in one global clock cycle [49]. The same methodology can be used to construct a multi-ported register file [50].

See Fig. 7.8 for an overview of a multi-pumped memory. At the global clock logic level, two write and two read ports are available. At the local, double, frequency, one write access is fed directly to the SRAM. The other write is delayed by one clock cycle. The SRAM, running at twice the frequency, can thereby handle both writes in one global cycle, albeit a single ported SRAM. In the same way, the first read is kept in a hold register for one cycle. The held value and second read are subsequently presented to the global logic at the same cycle. Figure 7.8b shows a timing diagram of two such reads being serviced in one logic clock cycle.

Increasing the clock frequency x times, allows for an additional $2x$ access ports (both read and write are multiplied). Note that the frequency of an SRAM cell is limited. For example, the single port SRAM used for this project has a limit of about $400\,MHz$. With the added wiring, even eight access ports would be difficult.

If enough access ports can be constructed, the total area is very low. Only a single SRAM cell and a little bit of logic are needed for all. On the other hand, the design is quickly limited by speed and power consumption rises with the number of ports. Some problems like read and write order and clock domain crossings also have to be addressed, increasing the design complexity.



(a) Schematic overview of multiple accesses

(b) Timing diagram of multiple accesses

Figure 7.8: Multi-pumped memory overview

7.5.5. Standard cells

It is also possible to not define an architecture at all. In this case, the synthesis tool will implement the memory using standard cells. The bit cell no longer consists of two inverters, but instead becomes a flip-flop. To write a specific bit cell, a large demultiplexer is used and to read a bit cell, a large multiplexer is used. See Fig. 7.9.

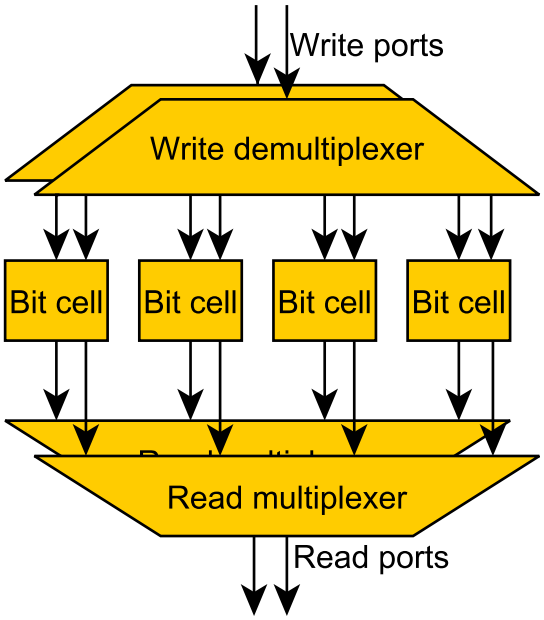
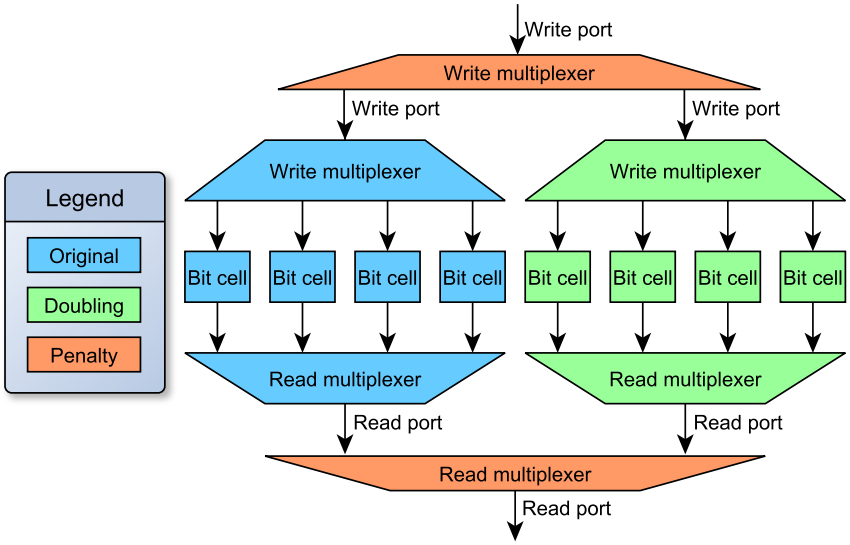


Figure 7.9: Register file layout when implemented using standard cells

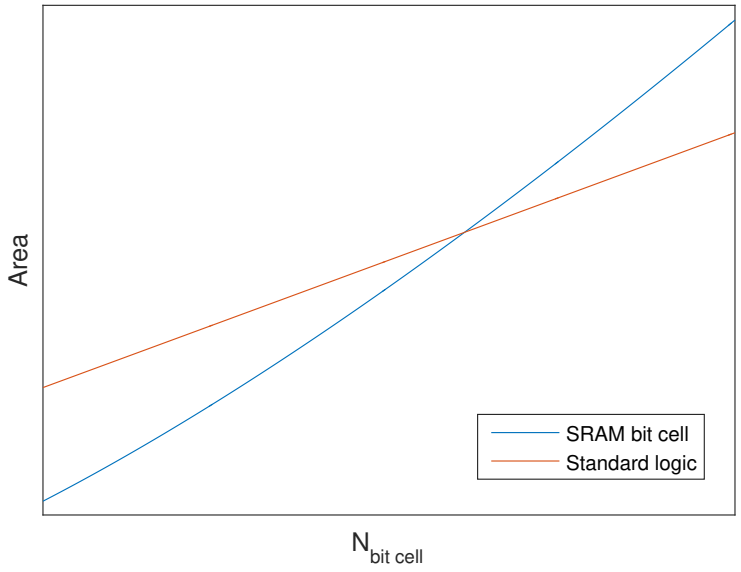
A big disadvantage of this implementation is the size of the bit cells, as well as the size of the access lines (multiplexers). The bit cell, when implemented as a flip-flop, is roughly $6.84\,\mu m^2$ in size. More than three times as much as two inverters of $1.08\,\mu m^2$. Due to an efficient layout, the SRAM bit cell is even smaller at roughly $0.7\,\mu m^2$. Furthermore, doubling the bit cells more than doubles the access logic. See Fig. 7.10a for the incurred penalty and Fig. 7.10b for a comparison of this penalty versus the linear scaling of SRAM.

Furthermore, there is a huge routing overhead in the read and write multiplexers. In [51, p. 56], this overhead is up to 65% of the total area, compared to 25% of regular logic. This was after hand-optimizing and placing the register file, as it was otherwise impossible to route.

The routing overhead in combination of the bad scaling makes the total area very design specific and hard to predict, but generally the worst aspect of this implementation. Even though there is a lot of wiring, most of the logic is not active during a read or write. Therefore, power consumption is relatively low. The access lines can easily be pipelined and performance is generally good. Furthermore, implementing and scaling the standard logic cells requires almost no design effort, making this the least complex implementation.



(a) Extra sizing penalty when doubling bit cells



(b) SRAM sizing vs standard logic sizing

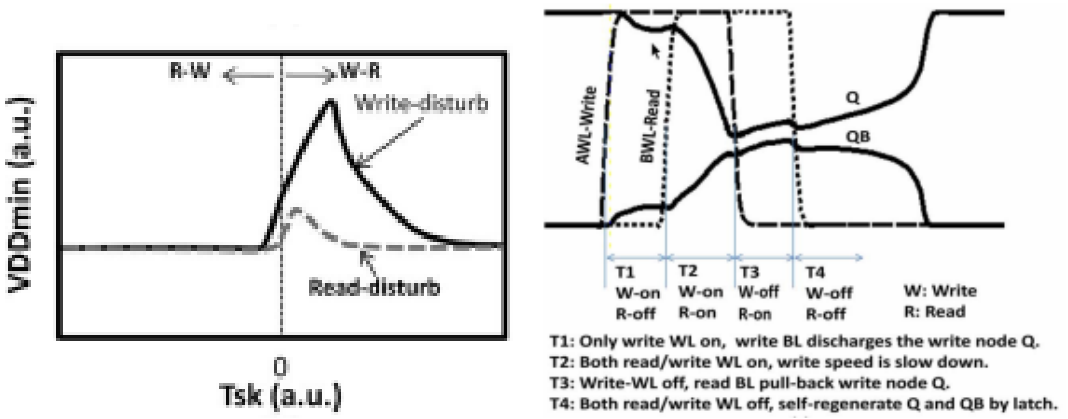
Figure 7.10: Resizing standard cell register file implementation

7.5.6. Full custom transistor level design

For the most competitive register file, it can be designed by hand on transistor level. This has the advantage of small bit cells and good scalability of SRAM, while allowing any number of access ports. Note how adding an access port to the SRAM cell in Fig. 7.5a requires almost no extra area. In reality, however, there is a reason why SRAM cells usually only have one or two access ports.

When a read or write happens on a bit cell, the whole bit line gets charged. All access transistors connected to the same bit line will leak some current. Furthermore, neighbouring bit cells might need to sink current during a read, or leak current during a write. Both will cause the ground voltage to locally bounce above $0\,V$. These read and write disturbances are shown in Fig. 7.11a. After such a disturbance, it is crucial the voltage can regenerate (Fig. 7.11b), or the bit cell can flip and its value is destroyed. This is already a big problem for dual port SRAM [52] and will become even worse when more ports are added.

A lot of work is contributed towards calculating the Static Noise Margin (SNR) and should be used when doing a custom design [53][54][55][56]. There are four port designs [57] and even a 26 access port SRAM implementation [58]. Therefore, a 24 port register file should be possible.



(a) Read or write on neighbouring cell

(b) Regeneration after disturbance

Figure 7.11: Read and write disturbances in SRAM [52]

When a design is custom made, more exotic implementations are possible. The Intel Itanium-2, a commercially available VLIW processor, uses the same 4T storage cell as regular SRAM, see Fig. 7.12. To reduce disturbances, all 12 access ports are simultaneously in read or write mode. By double pumping this memory, 12 read and 12 write accesses can be made every cycle, creating a very fast register file [59].

Instead of improving speed, area and power might be reduced by changing the bit cell design. For example, in [60], the 6T SRAM design is reduced to a 4T design, see Fig. 7.13. The stated main disadvantage is that I_{OFF-P} must be larger than I_{OFF-N}, requiring relatively large access line transistors. However, as there is a need for 24 access ports anyways, this PMOS leakage current can be shared among all ports. Both reducing the bit cell size from four to two transistors and effectively utilizing the leakage current, instead of introducing an inefficiency.

A different 4T design is given in [61], where again the main disadvantage is the requirement of a large leakage current. The design achieves 20% area improvement and 45% dynamic energy improvement compared to the conventional 6T design. In [62] an even smaller, 2T design is proposed, at the cost of a slightly higher power consumption.

Therefore, with a full-custom transistor design, a design can be composed that is faster, smaller and more energy efficient than all the other designs. However, this comes at the great cost of being far more complex to implement. Furthermore, testing the design is challenging without actually fabricating it, making it unsuitable for prototyping the ρ-VEX at the same time. Scaling the implementation when the ρ-VEX' register file changes is also not straightforward, due to the very specific hardware.

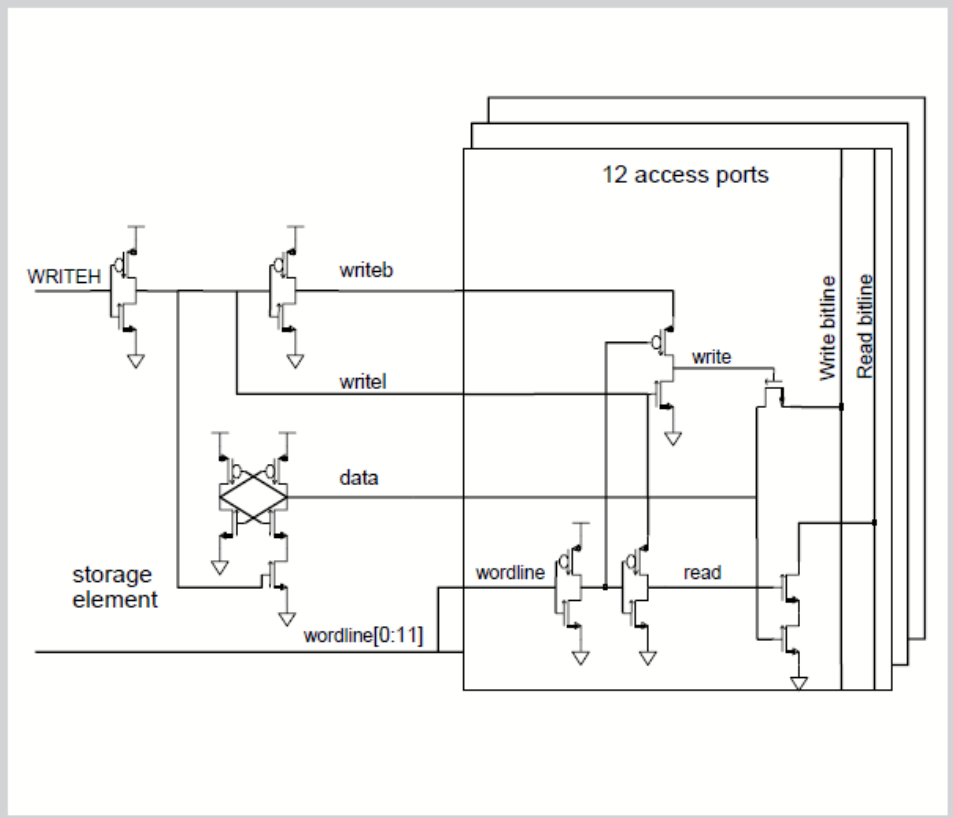


Figure 7.12: Intel Itanium-2 12-port register file implementation [59]

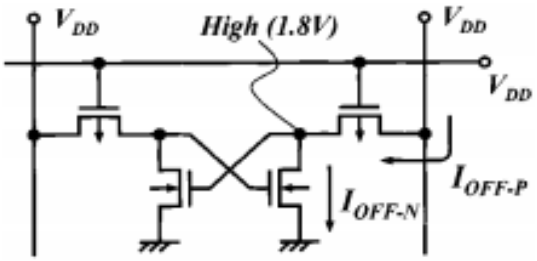


Figure 7.13: Loadless 4T SRAM cell [60]

7.5.7. Clustering

Clustering is not really focused on implementing the register file, but instead simplifies it. The ρ-VEX consists of eight pipelanes, communicating through the register file. However, this register sharing is only used infrequently. In fact, when the ρ-VEX is not configured in eight way mode, but instead in two times four, there is no cross communication between the smaller groups. They operate two independent programs.

This raises the question, is it really necessary to implement all 24 access ports, or can this be reduced? In fact, the VEX, from which the ρ-VEX is derived, already is designed with clustering in mind [63]. Splitting the ρ-VEX in two clusters, with four pipelanes each, significantly reduces the problem. Instead of requiring a register file with sixteen read ports and eight write ports, they can be reduced to two register files of eight read ports and four write ports. Both the total area and incurred wiring are less.

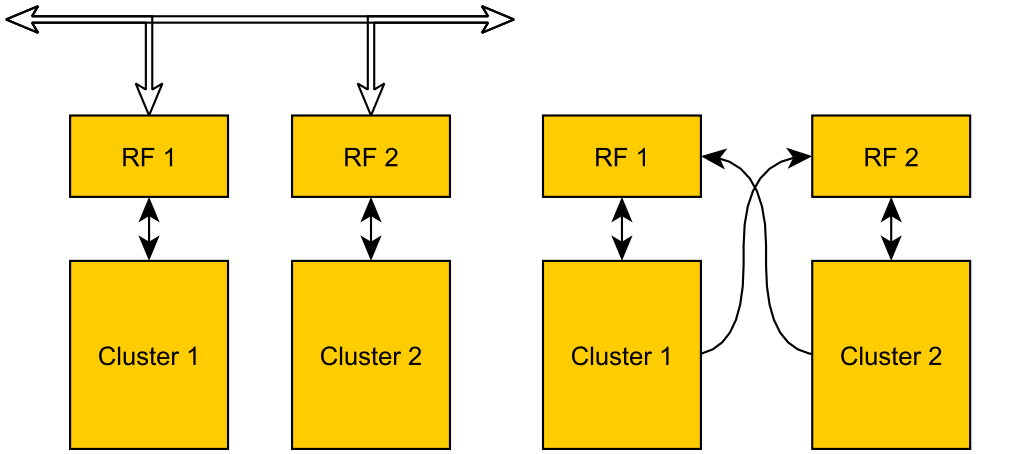
To still allow the ρ-VEX to operate in one times eight mode, utilizing eight pipelanes and thereby both clusters, still requires a way of communication. Roughly two solutions can be described. See Fig. 7.14. Either a bus architecture or dedicated write operands in extended instructions are utilized. All possibilities of clustering are extensively explored and described in [51].

Using a bus architecture as in Fig. 7.14a, a special copy operation needs to be supported in the ISA. This instruction will retrieve a value from another cluster and copy it to a local register file. Therefore, each register file needs an additional read/write port, still needing only a total of thirteen access ports. Due to the bus architecture, it is very easy to expand to more than two clusters. However, for every register accessed in another cluster, a special instruction needs to be executed, thereby having an overhead of one cycle.

By instead extending all instructions with an extra destination operand, results can directly be written to the other clusters' register file. This is shown in Fig. 7.14a. Although each register file still requires thirteen access ports, the extra port is only a write port. Furthermore, due to the destination register being part of the instruction, there is no added latency using this method. When more clusters are needed, there will be more dependencies and cross connections, increasing the complexity for the extended destination registers.

In [51], the best clustering organization reduces area by 45% and energy consumption by 43%. A different organization can improve performance by almost four times. Therefore, if this option is sufficiently explored, it can lead to very significant improvements by architectural design. The implementation of the register file can improve this even further.

Like the banked memory approach, the performance impact of clustering heavily depends on the compiler. However, unlike banked memory, the VEX and, therefore, the compiler toolchain already supports clustering. As the ρ-VEX does not support multiple clusters, this implementation is not further investigated. In a future design it might be very interesting. The total area and power consumption of the smaller register files can be greatly improved, due to reduced inter connectivity. As long as eight times configuration is used infrequently or programs can be efficiently compiled, clustering will also benefit performance. This method will require a re implementation of clustering in the ρ-VEX and is thereby slightly more complex to realize.



(a) Split register file connected by bus (b) Split register file with extended instructions

Figure 7.14: Clustered VLIW organization, with split register file

7.5.8. Comparison

All previously mentioned memory implementations can be compared. Important metrics are the area, speed and power consumption, as they dominate the final ρ-VEX design. Furthermore, as this is just a prototype, the complexity of implementation is considered. Finally, the ρ-VEX and its register file might still change, making scalability of the implementation important. See Table 7.1.

The FPGA version of the ρ-VEX is built using replicated memory, making it very easy to port. As the BRAM are of fixed size, the area is unimportant and power won't dominate. For an ASIC version this can be improved upon with a different implementation, making replication an unattractive option. Due to the requirement of 24 access ports, multi-pumping is severely limited by speed. It might be part of a hybrid solution like the Itanium-2, but due to the high complexity of such an implementation that is deemed not worth the effort. Likewise, a full custom design would yield very good results. However, the high complexity and bad scalability would conflict with the goal of a useful (**G**.4) prototype (**G**.1).

Both banked memory and a clustered design will yield an attractive implementation. They are easily scalable for larger implementations and perform very well. A clustered design might limit the speed of an eight way configuration, whereas a banked memory might need a redesign in amount of banks if the number of access ports changes. However, as both these implementations require changes to the ρ-VEX as well as to the compiler, they are not suitable for an initial prototype. Instead, they should be considered for a future implementation.

Therefore, the final implementation will make use of standard logic cells. This can be easily prototyped and verification tools can ensure correctness of the design. Due to the poor area scaling compared to SRAM and huge wire overhead it is a slightly unattractive option. However, as will be shown in Chapter 8, it is still smaller than a replicated implementation and allows for the full prototype to be implemented, albeit requiring some hand-optimization.

Table 7.1: Comparison of different N-port memory implementations

Implementation	Area	Speed	Power	Complexity	Scalability
Banked	+	+	++	-	+
Replication	- -	-	- -	++	-
Multi-pumping	++	-	o	-	- -
Standard cell	-	+	+	+	o
Full custom	++	++	++	- -	- -
Clustering	++	o	++	-	++

7.6. Conclusion

All important signals required for implementation of the SSI are discussed. The HSI pin sharing scheme is highlighted due to the increased complexity for implementation (**G**.2). Furthermore, four different debugging schemes are discussed, UART, scan, standalone mode and a program ROM. The UART interface allows full functionality without using the SSI (**G**.4). Scan chains will be used to allow full observability and controllability of the ρ-VEX, allowing the design to be debugged to correct any mistake (**G**.4b). Standalone mode and a program ROM would allow the ASIC to operate with reduced or no external input at all, however will not yet be implemented.

The different modes of operation available to the ASIC implementation of the ρ-VEX have been discussed. The operational modes are functional fast, functional slow, UART and scan mode. The different operational modes allow for the design to be useful, even if there is an undetected mistake (**G**.4). Furthermore, the netlist changes required to implement the data and instruction cache and register file have been explained.

All possible register file implementations are thoroughly discussed. A comparison is made between a banked, replicated, multi-pumped, standard cell or full custom memory. In addition, the benefits of clustering the ρ-VEX are discussed (**G**.3). Due to its low complexity and adequate performance, the register file of this prototype will be implemented using standard cells.

For future designs it is suggested to weigh the advantages of a banked or clustered design against the added complexity. A banked register file is extremely popular in superscalar processors. Furthermore, an overall different architecture by clustering the processor, might lead to more than 40% savings in both area and power consumption, or improve processor speed by almost four times.

8

ASIC implementation

The whole RTL to GDSII design flow as described in Chapter 4 will be used for implementing the ρ-VEX ASIC. This chapter will assume all information as described, as well as the used software, to be known. If the reader is not familiar with synthesis in Synopsys DC or P&R in Cadence Encounter, it is encouraged to recap the design flow. This chapter will largely follow every step in the design flow.

First of all, all design steps and optimizations are implemented in a scripted environment, as described in Section 8.1. Subsequently, the constraints as established in Chapter 5 and Chapter 6 are implemented for use in the design flow. Due to its importance in this project, the final register file implementation is once again described in Section 8.3. Section 8.4 through Section 8.8 are concerned with the actual synthesis and P&R design steps.

Any other optimizations performed are explained in Section 8.10. These mostly include global settings towards improving the result. Finally, Section 8.11 includes some important considerations when working with the final implementation. These considerations are of use to the PCB designer implementing the ρ-VEX ASIC, or might answer questions regarding missing steps in the design flow.

Note that the design verification steps are not part of this chapter. Instead, they are split off and put in Chapter 9. This is done due to the importance of signoff analysis for this and any other ASIC design. Also, the amount of bug fixes would overwhelm the chapter.

8.1. Scripting and documentation

One of the main goals for this work, (**G**.6), is to create a well documented and reproducible design. This means that, even when the author leaves, a new student must be able to pick up the work and continue improving without the need for a full year of learning the design again.

Therefore, all steps as performed and described in this chapter and Chapter 9, are fully automated using TCL scripts. By running one (or a few) commands, it is possible to convert the entire ρ-VEX from VHDL to a GDSII ready for tape-out. This also means that, when minor improvements are to be made to the netlist, there is no need to change the scripted flow at all. Furthermore, when larger changes are made, these can be mostly configured using a single configuration file, controlling synthesis, P&R and verification all at once.

To help in understanding and improving the script in the future, this document should be used. In addition, a full user guide has been made and attached in Appendix D to get a new designer started. This user guide has been thoroughly tested in combination with the scripts and will yield an error-free design. For more in-depth changes, the scripts themselves will have to be altered. To aid in understanding this, multiple README files will guide a designer through the project and very verbose comments throughout the TCL scripts should be plenty to understand every single detail.

All in all, expanding on this work should be easy enough for anyone who is at least decently comfortable with scripting. Prior experience with the design tools will help, but learning the flow from Chapter 4 is a quick and good start as well.

8.2. Constraining the design

As already explained throughout this document, the main optimization constraints are area, timing and power. As synthesis only deals with the worst-case constraints, these are laid out in Section 8.2.1. To ensure the cross-clock boundaries to be correct, they have their separate constraints. Continuing on to P&R, all operational modes are added to the constraints. The pad and macro constraints as well as the power constraints of Fig. 4.1 are discussed in their respective Section 8.4 and Section 8.5.

8.2.1. Synthesis area

For most ASIC designs, the area is only limited by its price. However, as this is only a prototype and will be fabricated as part of an MPW by IMEC (see Section 2.2.1), the area is limited. To be exact, the allowed design area is $1875\,\mu m \cdot 1875\,\mu m$. The synthesis design will include IO pads, but exclude power and ground pads. The latter will be added during floorplanning. Furthermore, a typical design will have a cell placement utilization of anywhere between 65% to 80%. A good estimate can only be made with plenty of experience and when the design is known. For the ρ-VEX, an initial P&R is performed. Due to the poor routability of the register file, the achieved logic utilization is only around 60%. About two thirds of the chip is filled with macro's (each of which has 100% utilization), making the chip wide utilization about 90%.

See Eq. (8.1) for the area constraint, with A_{chip} the total chip area, A_{power} the area required for power planning and U the expected chip utilization. The area required by power can be approximated by taking the size of the full IO ring, being $120\,\mu m$ wide, and subtracting the functional pad area. The design has six digital pads, one SSI with eight analog pads and three SSIs with six analog pads, making a total of $N_{pads} = 32$. Each pad is $A_{pad} = 88.8 \cdot 60 = 5328\,\mu m^2$ in size [20]. The final constrained chip area, as calculated in Eq. (8.8), is $2.56\,mm^2$.

$$A_{constr} = (A_{chip} - A_{power}) \cdot U \quad \text{Equation (8.1)}$$

$$A_{chip} = 1.875 \cdot 1.875 \quad \text{Equation (8.2)}$$

$$\approx 3.5\,mm^2 \quad \text{Equation (8.3)}$$

$$A_{IO} = A_{chip} - (1.875 - 2 \cdot 0.120)^2 \quad \text{Equation (8.4)}$$

$$\approx 0.84\,mm^2 \quad \text{Equation (8.5)}$$

$$A_{power} = A_{IO} - N_{pads} \cdot A_{pad} \quad \text{Equation (8.6)}$$

$$\approx 0.67\,mm^2 \quad \text{Equation (8.7)}$$

$$A_{constr} = 2.56\,mm^2 \quad \text{Equation (8.8)}$$

8.2.2. Synthesis timing and power

As shown in Chapter 6, the maximum achievable clock speed of the ρ-VEX ASIC synthesis step is about $141\,MHz$. However, the main goal (**G**.5b), requires a low power consumption. Therefore, the design timing is constrained about 30% lower, at $100\,MHz$. Power is optimized as much as possible. See Table 8.1 for a synthesis run for both clock speeds. Notice how the 30% loosened clock constraint results in an energy reduction of 85%. $100\,MHz$ is still faster than the current ρ-VEX clock speed on FPGA and lowering it more did not show significant power improvements. Therefore, the ρ-VEX clock period is constrained at $10\,ns$.

Note that the clock network is only added during P&R, therefore, it is left out of the comparison.

Table 8.1: Synthesis results for loose constrained clock

Clock frequency (MHz)	Energy ($\mu W\,MHz^{-1}$)
141	2439.95
100	353.23

8.2.3. Clock latency

This implementation will not have an on-chip clock generation. Instead, it will rely on the clock recovery circuit of an SSI instance. The connected FPGA will provide a differential clock signal. This clock will

not be perfect, but instead have some jitter and a non-zero transistion time.

The jitter is random noise, causing the clock to arrive earlier or later. Because it is random, it can't be factored out. Instead, a margin should be taken when calculating any clock constrained timing. In [64] a prediction is made of the frequency dependent jitter. It seems a clock running at more than $100\,MHz$ will have about $0.2\,ns$ of jitter. To be more precise, the data sheet of the used FPGA should be taken. In this case, the ML-605 evaluation board with Virtex-6.

The oscillator on the ML-605 is one of the EG-2121CA [65] series, generating an LVDS clock. The stated phase jitter is $0.05 \cdot 10^{-3}$ UI typ., or 50 ppm. Using Eqs. (8.9) to (8.11) the jitter can be calculated. For the worst case jitter, the HSI runs in $200\,MHz$ mode, therefore, $f = 200\,MHz$. Subsequently, $\delta_f = 10\,kHz$, $T_{pp} = 0.5\,ns$ and the worst case jitter is $0.25\,ns$.

From the Virtex-6 datasheet [30], the transition time of an LVDS driver can be determined to be $0.2\,ns$. Both the transition time and jitter are added as clock uncertainties to the constraints.

$$\delta_f = \frac{f \cdot ppm}{10^6} \qquad \text{Equation (8.9)}$$

$$T_{pp} = \frac{1}{f - \delta_f} - \frac{1}{f + \delta_f} \qquad \text{Equation (8.10)}$$

$$jitter = \frac{T_{pp}}{2} \qquad \text{Equation (8.11)}$$

8.2.4. Cross-clock boundaries

To deal with cross-clock boundaries, the software tools need to know what the relation is of the two clocks. During synthesis, only the worst case scenario is investigated. Therefore, the HSI runs at $400\,MHz$ and contains a clock divider of four times, to generate a core clock of $100\,MHz$. These constraints can simply be set using *create_clock* and *create_generated_clock*. The generated clock command ensures a static relation from HSI clock to core clock. However, as the clock net will not be synthesized until during P&R, the relation also isn't properly timed yet. As will be shown in Section 9.2.4, the clock divider does not properly work yet after synthesis. However, the constraint carries through to P&R, where this problem will be corrected.

8.2.5. Operational modes

After synthesis, all design constraints are collected in a Synopsys Design Constraints (SDC) file. The case as used was functional fast mode. The other operational modes have a small deviation and will get their own SDC file.

Using the *set_case_analysis* command, a zero or one can be forced on a specific pin. This is used to control the different operational modes. More specifically, the pin 'scan_enable' is used for scan testing, 'ratio_reg/Q' defines the clock division ratio (four times division or two times division) and 'ifsel_reg/Q' switches between functional and UART mode. Finally, all clock nets in Synopsys are marked as ideal, meaning they won't contribute to a timing delay. After CTS, a propagated clock constraint is set by loading yet another SDC file. In total eight cases are defined. See Table 8.2.

Table 8.2: Case analysis for each operational mode

Case	HSI clock (MHz)	core clock (MHz)	scan_enable	ratio_reg/Q	ifsel_reg/Q
functional_fast	400	100	0	1	1
functional_slow	200	100	0	0	1
functional_uart	-	100	0	-	0
test	-	10	1	-	0

8.3. Register file

As discussed in Section 7.5, this design will include a register file created entirely from standard logic cells. The implementation is extremely easy and during synthesis it can be optimized. However, there is one big disadvantage: compile time.

The total ρ-VEX ASIC contains about 200,000 logic cells. More than half of these cells are only for the register file. Therefore, the standard cell implementation quite literally doubles the problem size, not even counting the wiring needed to connect these cells. As synthesis, standard cell placement and routing are all NP-hard problems, their execution times scales super-linear with the problem size. In practice, synthesis could complete in about an hour. However, when the register file was replaced, it took more than four times as long. The whole flow takes a total of three days to complete, severely limiting the ability to quickly test changes required for prototyping.

A large effort is put in synthesizing, placing and routing the cells of the register file, even though the register file doesn't change. Therefore, a future design will need to be partitioned in two sub-designs, the core and the register file. By partitioning the design, a change can be made in the core without the need to redo all steps for the register file. For this project the exact characteristics and required timing were entirely unknown, making partitioning almost impossible. However, for a future design it is recommended to either replace the implementation or take a closer look at design partitioning.

8.4. Floorplan

During floorplanning, three interesting design choices have been made. First, the best cache and SSI macro placement have been determined. Subsequently, some module constraints are added and finally the IO ring is discussed.

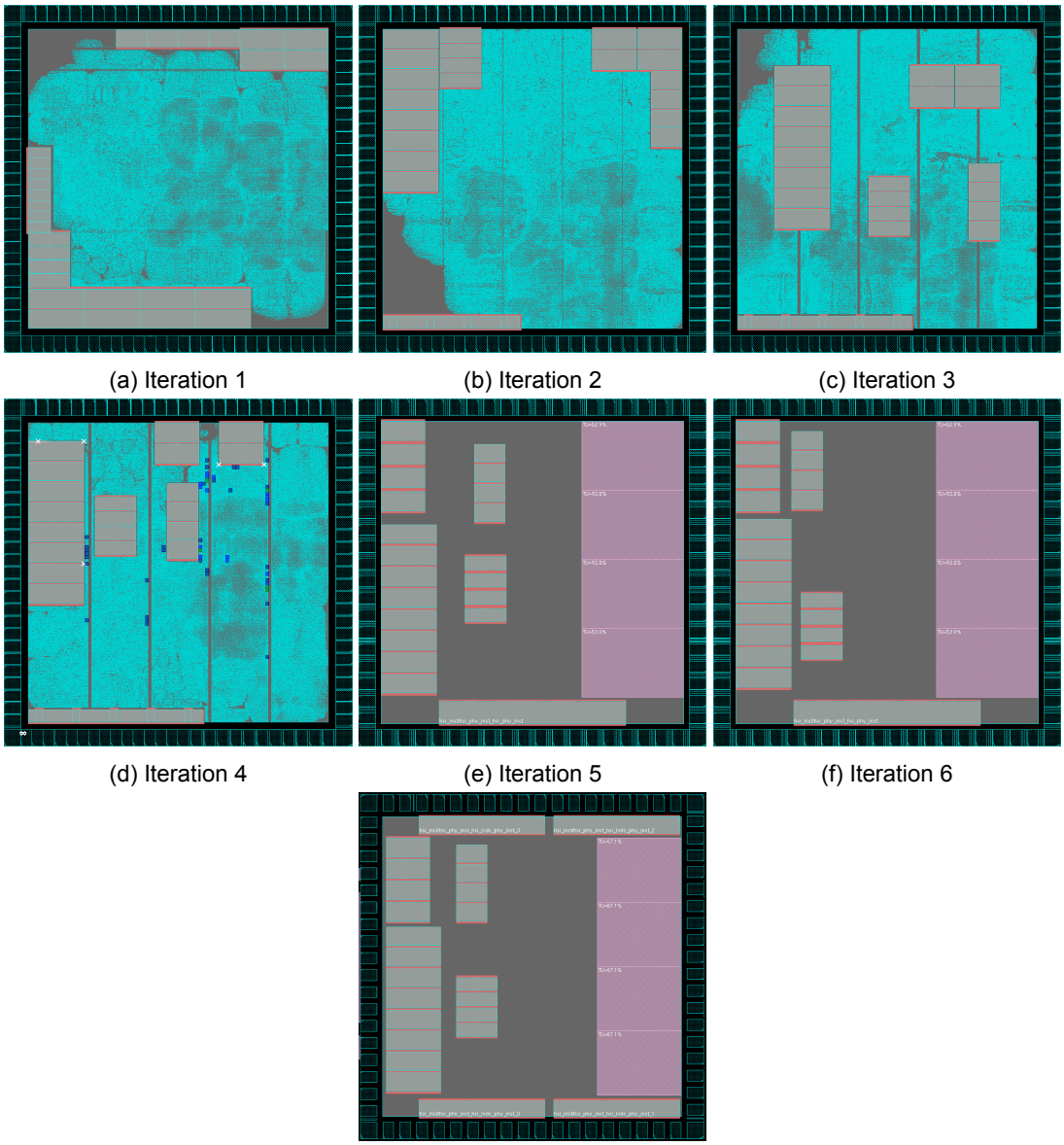
8.4.1. Macro placement

Initially, Encounter is able to calculate a 'good' placement of all macro instances in the design. It does this by calculating the connectivity of each instance to standard cells and ensures there is plenty of space to create a connection to every pin. However, after having run the full P&R flow, timing regarding the cache instances seemed quite bad. Therefore, the initial placement of Fig. 8.1a has been manually refined. The instruction cache instances were seeded in the south-west and the data cache instances in the north-east.

In the second iteration, the caches have all been moved towards the north side of the chip. In this iteration, a single SSI instance was fit in the south-east corner. In the following four iterations it is clearly visible that all caches are moved closer and closer together, towards the west side. This is because the register file will take up the full east side of the chip. The sixth iteration is very close to being able to meet design timing, but still not there yet. Note that in the final iteration, Fig. 8.1g, all four SSI instances can fit on the north and south side of the chip.

Instead of relying on the Encounter seeded placement, inter-connectivity has been manually investigated. Using the flightlines visible in Fig. 8.2, all direct dependencies are checked. In Fig. 8.2e all instances are rotated, moved and some even got a different aspect ratio (using the mux option in the Faraday memaker) such that the flightlines are minimal in size. In the last iteration, Fig. 8.2f, all instances are slightly more spaced apart, as the normal cache logic did not fit in between the instances, causing unnecessarily long wires.

Note that in the final implementation, all instances are horizontally aligned, as can be seen in Fig. 8.3. This allows the power stripes, as discussed in Section 8.5.3, to be easily connected to all instances. Furthermore, all distances are measured exactly to best fit the required logic.

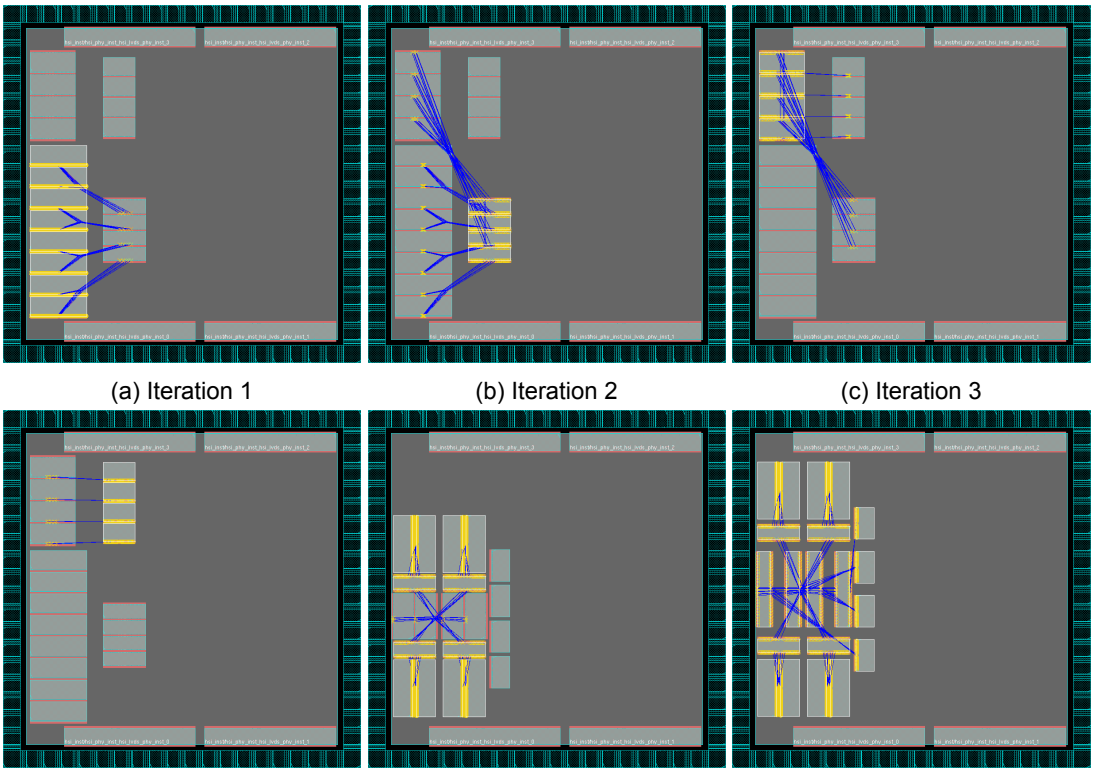


(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

(d) Iteration 4 (e) Iteration 5 (f) Iteration 6

(g) Iteration 7

Figure 8.1: Cache SRAM placement iterations from an initial seed



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

(d) Iteration 4 (e) Iteration 5 (f) Iteration 6

Figure 8.2: Cache SRAM ordering by inter-connectivity

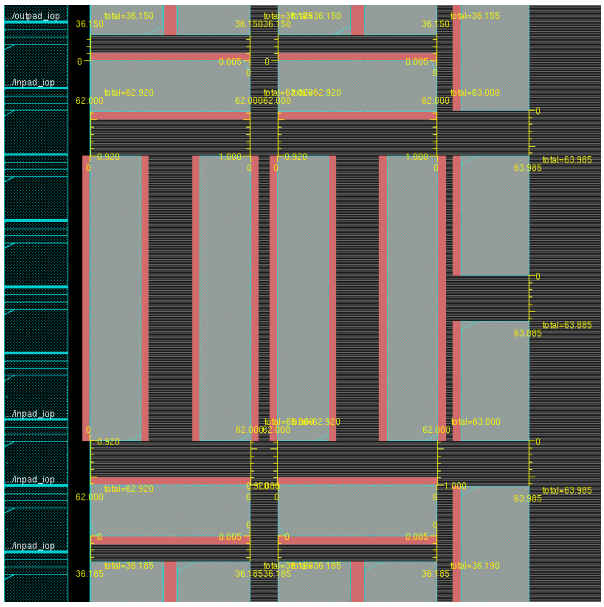


Figure 8.3: Measurements performed to fit all cache macro instances

8.4.2. Placement constraints

In the design, two instances are manually constrained. These are the register file and the HSI. Furthermore, a placement blockage is added for the 'analog' areas. See Fig. 8.4 for the constraint area.

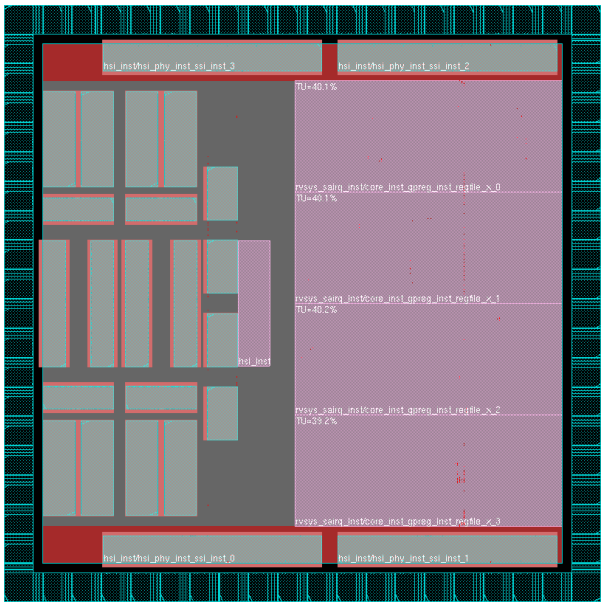


Figure 8.4: Module constraints for the register file and HSI

Register file module constraints

Due to its sheer size, the register file is split in four sub-instances. One register file part per context. These are visible in Fig. 8.4 on the east side. Each part is constrained as a guide, the most permissive constraint. A fence would create very long wires from instances within the register file to instances outside the constraint. Furthermore, the TU is about 40%, allowing 60% of the area to be used for routing. Anything less would cause overcongestion in the routing phase, preventing it from completing.

HSI module constraints

The HSI is concerned with all the SSI instances on the north and south side of the chip. Furthermore, as explained in Section 8.4.3, it is connected with the digital IO pads on the west side. For some reason, Encounter tries to make the connections to these digital pads as short as possible. Instead, the higher clocked SSI instances are much harder to route. Therefore, it is manually constrained such that these wires are shorter and outside the cache area, making them easy to route. Again, a guide is used, to allow Encounter to optimize the placement. The TU is left default with 65%.

Analog placement blockage

In Fig. 8.4, a red area is marked around the SSI instances at the north and south side. In the power plan Section 8.5, these areas will get an analog power ring. No digital power will be available there, therefore, standard cell instances need to be prevented from being placed. A more efficient approach would be to stretch the SSI instances over the full width of the chip. This was discussed, but apparently too difficult. Therefore, some parts of the chip are now unusable. In a future design this can be improved.

8.4.3. IO ring

One of the constraints of IMEC was a minimum IO pad pitch of $90\,\mu m$ [66]. Otherwise they can't bond the chip and have to outsource it, increasing the cost. Therefore, the whole FPGA is designed to fit this minimum pitch, including the SSI. However, as was clear only near the end of the project, it was not the pitch that should be constrained, but the bond pad size. As the bond pad size is technology dependent and can't be changed, bonding still has to be outsourced. In a future design, therefore, this pitch constraint can be ignored, allowing for more IO pads.

Due to the inclusion of the SSI instances, three voltage domains are defined. $1.2\,V$ for digital cells and $1.2\,V$ and $2.5\,V$ for analog cells. To separate these domains, special IO break cells have to be used [20]. These are highlighted in Fig. 8.5. The four larger cells separate digital from analog $1.2\,V$ and the four smaller cells separate the two analog domains. Both analog domains contain two power and two ground pads, required for proper ESD protection. All 26 functional pads required for the SSI instances are placed directly aligned with the corresponding SSI pins, as visualized by the small flightlines.

As the east side of the chip is fully covered with the register file and it has no external connections, all six digital pads are placed on the west side of the IO ring. The remaining pads on both the east and west side are filled with supply pads, allowing for plenty of ESD protection, minimal EM effects and easy power routing.

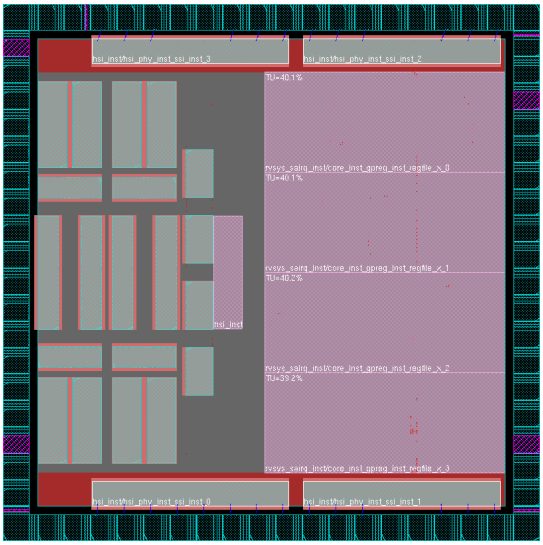


Figure 8.5: IO ring with break cells highlighted

8.5. Powerplan

The digital and analog domains have their separate power rings, as explained below. However, due to the large SSI instances, the digital power ring has a little quirk. Finally, to further reduce routing congestion, the power stripes across the register file have been shrunk.

8.5.1. Analog power rings

See Fig. 8.6 for the full power routing. The north and south side, around the SSI instances, have a triple power ring. The supply is routed as $2.5\,V$, $0\,V$, $1.2\,V$. The VSS wire in the middle reduces noise coupling. The ring is routed on ME3 (horizontal) and ME4 (vertical). This has been discussed to ensure the connections to the SSI are as short as possible. The digital power ring overlaps the analog power ring to make the ‘dead’ zone under the rings as small as possible.

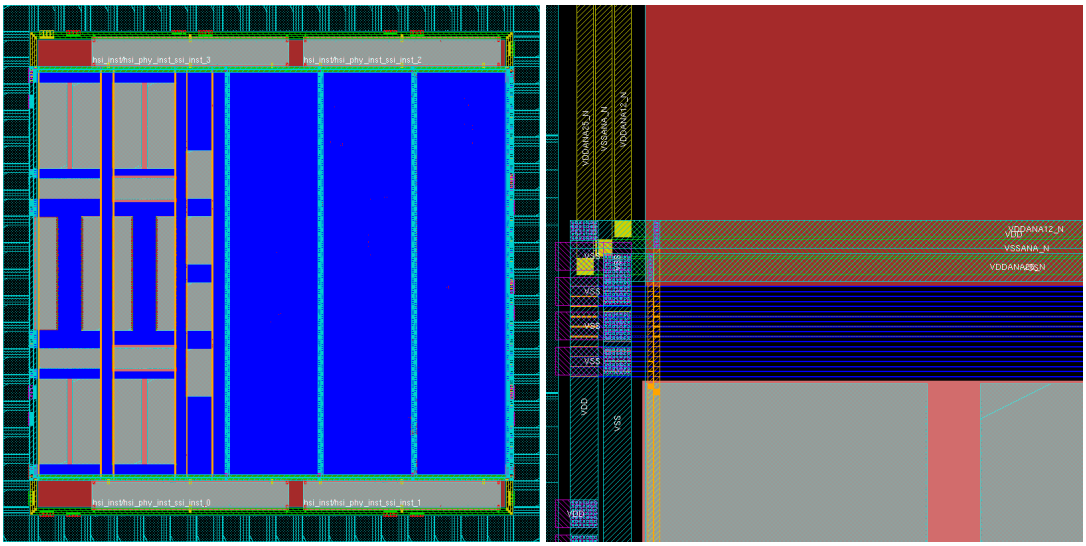
8.5.2. Digital ME8 power ring

As opposed to the analog ring, the digital ring does not have separate horizontal and vertical routing. Instead, only ME8 is used. The reason for this can be found in the TLR [14] and is summarized in Table 8.3. As can be seen, the resistance of ME8 is more than ten times lower than that of ME7 (the listed resistance is another ten times lower, but this is due to the ten times wider wires, which can also be achieved with ME7). The resistance of ME7 is again two times lower than the other metal layers. The reason for this is explained in Section 2.1.4. ME7 is twice the metal thickness. ME8 is eight times the metal thickness and made of a lower resistance material (copper instead of aluminum).

However, ME8 is only supposed to be routed vertically. This is against the ’design rules’, although only suggested. Encounter prints out a lot of warnings and errors, not understanding what is happening. To ensure a valid design, a separate DRC is performed. This is done in Section 9.5.2.

Table 8.3: TLR routing rules, capacitance and resistance

Layer	Min. width (μm)	Max. width (μm)	Direction	Cap / Len. ($pF\,\mu m^{-1}$)	Res / Len. ($\Omega\,\mu m^{-1}$)
ME1	0.09	12	Horizontal	0.000210	2.823166
ME2	0.10	12	Vertical	0.000204	2.176364
ME3	0.10	12	Horizontal	0.000204	2.176364
ME4	0.10	12	Vertical	0.000204	2.176364
ME5	0.10	12	Horizontal	0.000204	2.176364
ME6	0.10	12	Vertical	0.000204	2.176364
ME7	0.20	12	Horizontal	0.000227	0.546765
ME8	2.00	12	Vertical	0.000406	0.003982



(a) Global power routing (b) Enlarged north-west corner

Figure 8.6: Digital and analog power routing

8.5.3. Power stripes

In Fig. 8.6, vertical power stripes are visible on top of the cache instances. As already explained in Section 8.4, all these instances are exactly aligned. This means that the six stripes directly connect to the internal power rings, making this step very easy.

The three stripes across the register file are a bigger problem. Due to the large amount of wires, a high congestion is reported. During the routing stage this will lead to the inability to route the design. Table 8.4 lists the congestion report as outputted by Encounter, after a trial route before CTS. The amount of remaining routing channels in horizontal and vertical direction is listed. A negative 'remain' indicates that there are more wires than routing channels, therefore, it is overcongested.

Clearly quite a large amount of wires can't be routed. Further inspection shows that this is indeed centred on the register file, even after reducing the TU down to 40%. More noteworthy, the congestion is mostly vertical and appears along the power stripes. See Fig. 8.7a, where a red diamond indicates overcongestion.

The ME8 power stripes are $8\,\mu m$ wide, which is necessary to reduce the IR-drop and EM effects. However, Encounter automatically creates connections to ME1 at the same width. ME8 carries the current globally, while the ME1 to ME7 connections are only for local power distribution. Therefore, it is not necessary to create connections of $8\,\mu m$.

Instead, a routing blockage is procedurally aligned with each ME8 power stripe, preventing Encounter from routing the full width on ME1 to ME7. See Fig. 8.7b, where the large slab in the middle is a routing blockage. Visually the connections are already a lot smaller. After power routing, the routing blockage is removed to allow signals being routed in between. The congestion report after applying this power stripe shrinking is shown in Table 8.5. Even though this result is nice, it is still not enough.

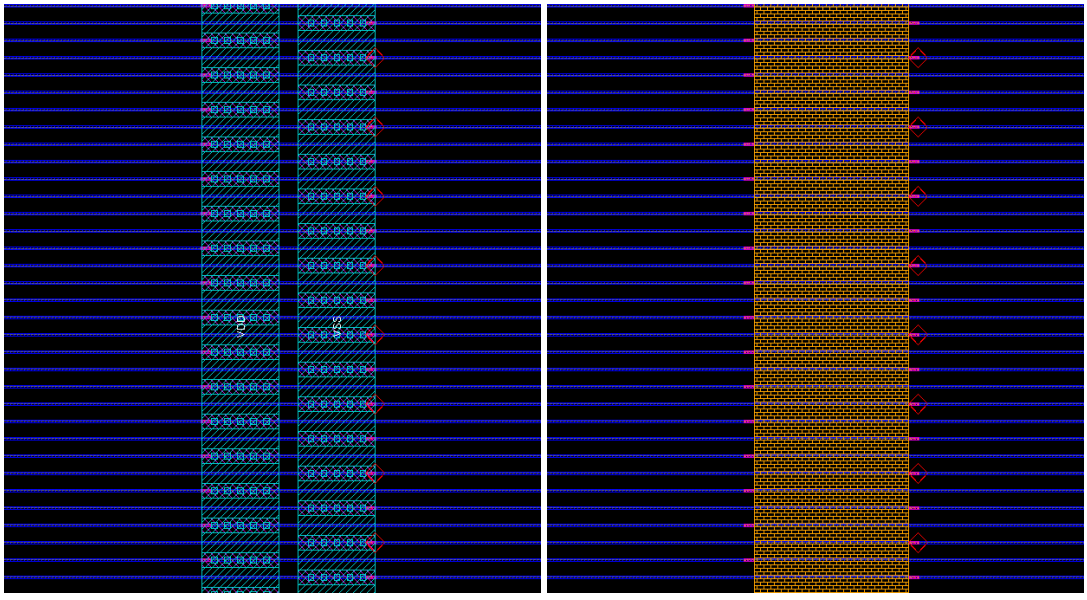
Table 8.4: Congestion report before stripe shrinking, listing the remaining horizontal and vertical routing channels

Remain	cntH		cntV	
-5	101	0.01%	1414	0.10%
-4	425	0.03%	2153	0.16%
-3	1838	0.13%	5959	0.44%
-2	6340	0.45%	17383	1.27%
-1	16209	1.15%	35934	2.62%
0	3478	2.45%	62551	4.57%
1	42001	2.98%	64872	4.74%
2	40986	2.91%	59201	4.32%
3	40070	2.84%	80807	5.90%
4	39544	2.80%	139112	10.16%
5	1187799	84.25%	900045	65.72%

Therefore, placement will apply some tricks to bring it down even further.

Table 8.5: Congestion report after stripe shrinking

Remain	cntH		cntV	
-5	3	0.00%	47	0.00%
-4	46	0.00%	218	0.02%
-3	274	0.02%	1068	0.08%
-2	1809	0.13%	5658	0.40%
-1	6703	0.47%	16995	1.21%
0	20296	1.42%	45873	3.26%
1	32880	2.30%	61844	4.40%
2	38907	2.73%	65980	4.69%
3	41910	2.94%	77414	5.51%
4	42642	2.99%	120170	8.55%
5	1241662	87.00%	1010469	71.88%



(a) Power stripe ME8 (b) Power stripe without ME8

Figure 8.7: Power stripes across the register file, shrunk to minimize congestion

8.6. Placement

As already announced, overcongestion is still a problem. This will be repaired using module padding. Next, hold violations are repaired using cell padding and ECO is used to achieve the final placement.

8.6.1. Module padding

Module padding was already introduced in Section 4.2.3. It effectively widens standard cells during placement. After placement is finished, this module padding is removed and cells can be moved around to reduce congestion. In the final design, a module padding of 60% is used on all cells. This allows a further congestion reduction of 7.5 times, with the congestion report of Table 8.6. This is good enough to allow proper routing later on.

Table 8.6: Congestion report after adding module padding

Remain	cntH		cntV	
-5	2	0.00%	5	0.00%
-4	6	0.00%	16	0.00%
-3	44	0.00%	75	0.01%
-2	217	0.02%	590	0.04%
-1	1206	0.08%	2769	0.20%
0	7133	0.50%	12608	0.90%
1	20381	1.43%	33934	2.41%
2	35289	2.47%	44187	3.14%
3	51517	3.61%	75535	5.36%
4	58860	4.12%	135351	9.61%
5	1252600	87.76%	1102889	78.33%

8.6.2. Cell padding

From an initial run, design routing still failed. The reported error was **info: 147328 net(s): Could not be fixed because of no legal loc*. This is more than 44% of the in total 332,501 nets, not being able to route. The message is somewhat cryptic, but is caused by hold violations. After setup timing is met, Encounter inserts buffers to meet hold timing. However, because it doesn't factor this in (not enough, at least) during placement, there is no chip area to add these buffers. Therefore, this has to be manually corrected.

To do so, cell padding is used, again as explained in Section 4.2.3. As it will be used to repair hold violations, which can happen only at a sequential element, the padding is added to every flip-flop. A size of 11 ME2 pitches is used, adding up to $4.2\,\mu m$. When abutted, this is enough to fit even the largest buffer BUFM48WA, at a width of $8.4\,\mu m$. After CTS the cell padding is removed, allowing all hold violations to be repaired.

8.6.3. ECO iterations

As explained in Section 4.4.1, ECO allows design steps to be reversed. Another option includes repairing placement violations after placement is already performed. An initial placement was not enough, but by optimizing the placement using a single ECO iteration, no violations persist. The overall chip placement utilization is about 62%, including the cell padding. The design is ready for CTS.

8.7. Clock Tree Synthesis

As explained in Section 4.2.4, there are multiple options to perform CTS. For this design, initially a regular synthesis was performed. However, the result was extremely poor. In Fig. 8.8, the clock phase difference is coloured for every sequential element in the design. Note that this is an earlier iteration of the floorplan. The exact timing of every element is not important, but there was a difference of up to $4\,ns$ across the chip. Extremely large compared to the total clock period of $10\,ns$.

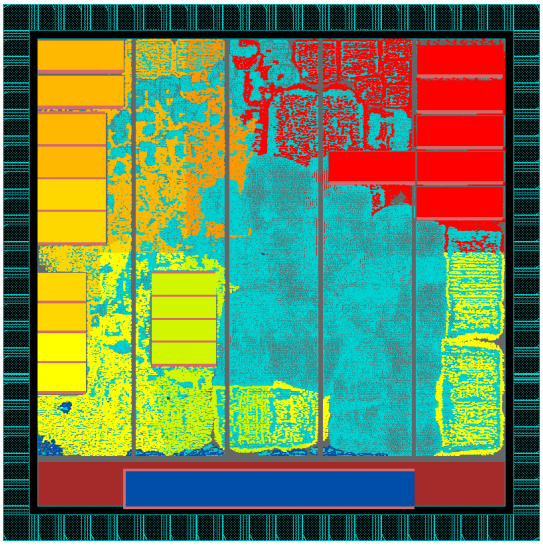


Figure 8.8: Large clock phase difference using regular CTS

8.7.1. Clock Concurrent Optimization

The poor CTS result were the reason to implement CCOpt. In the used version of Encounter (v14.13), CCOpt is not officially supported and requires some hidden functions to be used. For better results it is probably good to update Encounter to a newer version. However, this also requires all scripts to be adapted accordingly.

After CCOpt is performed, a hidden interface appears, the CCOpt debugger. See Fig. 8.9. The debugger shows the insertion delay from every clock root to every sequential element. Different paths can be viewed, as well as different PVT scenarios. It is used for example to check which path groups are failing hold timing and to estimate some clock constraints to be fed back into synthesis for a better result.

There are some things to note about this clock tree. First of all, there are 21 levels of clock buffering. This is quite a lot and explains the poor results of regular clock routing. Furthermore, the minimum and maximum delay to elements in the same path groups can differ by up to $1\,ns$. This is 10% of the clock period. This large difference in register timing is an indication of a very poor pipeline, where some parts drive the clock frequency down while other parts would easily be clocked faster. This is something to look at in a next design iteration.

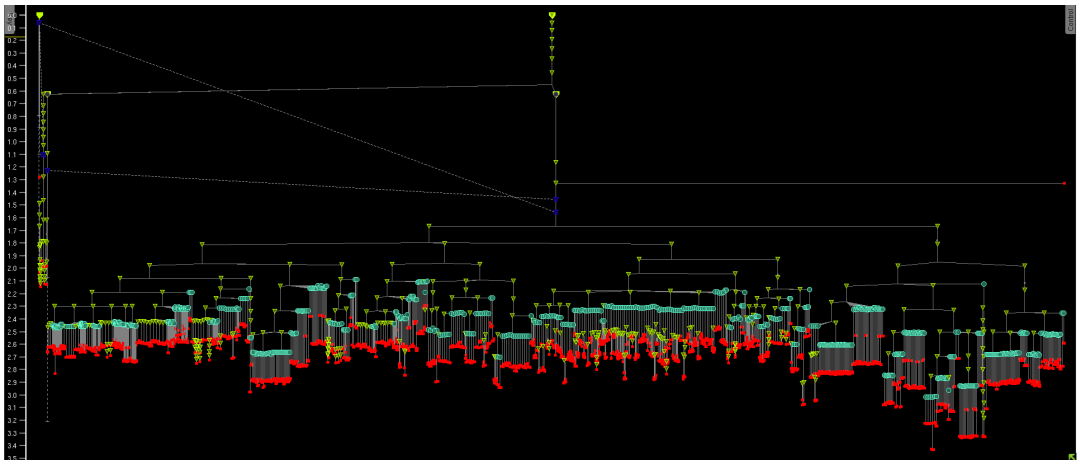


Figure 8.9: CCOpt debugger, showing the insertion delay to every sequential element

8.7.2. ECO iterations

For CTS, ECO is probably even more useful than for placement. Now that the clock tree exists, hold violations can be properly calculated. After routing it is a lot more difficult to repair hold violations. Therefore, doing this directly after CTS has great benefits. This can again be done using ECO, allowing the change of placement as well as clock nets. CCOpt leaves quite a bit of hold violations, as listed in Table 8.7.

Two ECO iterations are performed. This allows a positive setup WNS of $0.073\,ns$ and a positive hold WNS of $0.065\,ns$. Designing with positive slack is useful when later stages don't exactly achieve the predicted timing. Besides WNS, congestion can also be improved quite a bit more, as listed in Table 8.8.

Table 8.7: Hold violations after CCOpt

Hold mode	all
WNS(ns)	-0.489
TNS(ns)	-2332.6
Violating Paths	30642
All paths	71527

Table 8.8: Congestion report after performing ECO

Remain	cntH		cntV	
-5	0	0.00%	1	0.00%
-4	0	0.00%	1	0.00%
-3	0	0.00%	5	0.00%
-2	0	0.00%	16	0.00%
-1	0	0.00%	31	0.00%
0	0	0.00%	321	0.02%
1	75	0.01%	6515	0.46%
2	22	0.00%	740	0.05%
3	6	0.00%	3470	0.25%
4	419	0.03%	1350	0.10%
5	1426733	99.96%	1395477	99.12%

8.8. Routing

Due to the register file, routing is not as straightforward as standard cell placement and CTS. Even though for the most part congestion and hold violations have already been accounted for, clean routing is still difficult to achieve. Therefore, it is performed in several steps. First, non-default nets are routed. Next, regular routing is performed, with via optimization. Finally, several iterations of ECO are performed in four different steps.

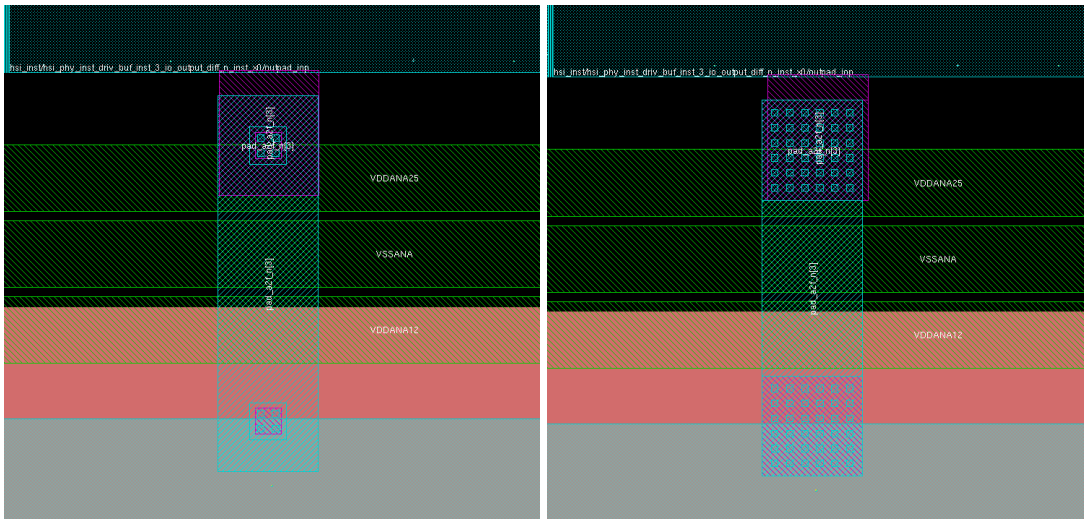
8.8.1. Non-default routing

The SSI instances were floorplanned such that their pin connections are as close to the respective IO pads as possible. However, due to the LVDS drivers, the wiring should have a resistance as low as possible. Like the power routing, this can be done by routing them on ME8 and making them wider. To do this, Encounter has an option called a Non-Default Rule (NDR).

An NDR can be applied to a connection. In this case, every connection from SSI to IO pad, 26 in total. A preferred top and bottom routing layer can be selected. As the IO pins don't have an ME8 connection, these layers are set to ME7 and ME8. Furthermore, the width of the wires is forced to $9\,\mu m$. This is smaller than the maximum width of $12\,\mu m$ as indicated in Table 8.3. In fact, the IO pins only use connections of $9\,\mu m$, therefore, making the wire wider is not an option.

The vias for these wires also were a little bit of a problem. For regular routing, vias of $2\,\mu m$ or $4\,\mu m$ are available, however these are too small for the wires of $9\,\mu m$. See Fig. 8.10a for the default via behaviour of NDR routing. Instead, vias have to be custom generated. This can be done in Encounter

and results in the connection as shown in Fig. 8.10b. The custom generated via ensures it is not the limiting factor, otherwise increasing the resistance significantly.



(a) Regular double cut via (b) Custom multi cut via

Figure 8.10: Custom generation of multi cut vias for NDR routing

As these non-default wires have very specific requirements, they are routed before other nets. In fact, the are routed after standard cell placement, even before CTS. This allows them to be verified without any conflicts, ensuring they are correct. Subsequently, the status of the nets is changed to 'fixed', preventing any optimization or changes to be made. An example of the resulting non-default wiring is shown in Fig. 8.11.

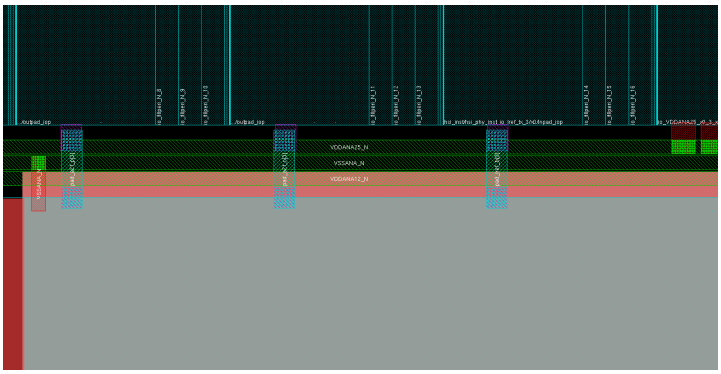


Figure 8.11: Non-default wiring for an SSI instance

8.8.2. Regular routing with multi-cut vias

After NDR routing and CTS, all other signals are routed. This is done exactly as already explained in Section 4.2.5. After regular routing, single-cut vias are replaced by multi-cut vias where enough space is available. This prevents vias from breaking and increases the manufacturing yield. The difference post-route via swapping makes is visible from Table 8.9 and Table 8.10. Not for every via there was enough space to insert multiple vias, which might be necessary for space applications (**G**.5). However, with 71.8% multi-cut vias, quite a good improvement is made.

8.8.3. Four step ECO routing

To repair all routing violations, four steps of ECO are performed. First, setup timing is corrected with a positive WNS of $0.10\,ns$. Subsequently, a special design rule ECO is performed. This repairs design rule violations like maximum load capacitance and maximum wire length. The third step of ECO is

Table 8.9: Via result before post-route via swapping

	single-cut		multi-cut		Total
Metal1	1288151	99.9%	1272	0.1%	1289423
Metal2	946312	99.7%	3124	0.3%	949436
Metal3	451520	99.0%	4524	1.0%	456044
Metal4	278798	100.0%	63	0.0%	278861
Metal5	160365	100.0%	14	0.0%	160379
Metal6	77701	100.0%	3	0.0%	77704
Metal7	7450	99.3%	52	0.7%	7502
	3210297	99.7%	9052	0.3%	3219349

Table 8.10: Via result after post-route via swapping

	single-cut		multi-cut		Total
Metal1	995045	70.7%	412491	29.3%	1407736
Metal2	63713	4.3%	1411715	95.7%	1475428
Metal3	56769	8.0%	656713	92.0%	713482
Metal4	65386	14.8%	376954	85.2%	442340
Metal5	40069	13.5%	257331	86.5%	297400
Metal6	24440	28.6%	61136	71.4%	85576
Metal7	2879	46.5%	3314	53.5%	6193
	1248301	28.2%	3179854	71.8 %	4428155

concerned with repairing hold violations. Encounter will try to do this without degrading setup timing and DRVs, but experience learns that this is not always possible. Here the positive slack is useful. By lowering the target setup WNS to $0.05\,ns$ and setting the target hold WNS the same, some degradation is permitted. This is absolutely required to achieve simultaneous positive slack on both setup and hold timing.

The fourth step considers geometry violations. To make the NP-hard routing problem slightly easier, Encounter will not always follow its own rules. Therefore, some wires can overlap or lie too close to each other and Encounter happily accepts this. A separate geometry check is performed to check for these violations. If any such violation occurs, all concerned wires are removed and rerouted. This will again cause other violations to happen.

In total, these four routing steps have to be iterated four times. Making the routing process consist of a total of 19 steps and being by far the most time consuming. However, the result is an error free design that meets the required timing. The final design utilization is about 61%, meaning that the initial placement had about one percent point of slack in the cell padding.

8.9. Design fixing

Before the chip can actually be fabricated, two more steps are required. These are the placement fill and metal fill.

8.9.1. Placement fill

As explained in Section 4.2.3, well tap cells provide power and ground connections for several standard cells. The connection is made using the N-well and P-well layers, two layers that Encounter doesn't know of. Furthermore, because only 61% of the design is utilized, the rest does not contain these wells, making most standard cells not connected to a well tap cell. To overcome this problem, special filler cells have to be placed.

Somehow, Encounter doesn't know how to deal with the large gaps in the register file due to the low target utilization. Therefore, even after filling, gaps persist in the design, see Fig. 8.12. These will cause violations during design verification. To solve the problem, fillers are added in two phases. First, most of the design is filled with preferred cells. Subsequently, for the remaining gaps, special compound fillers are constructed that exactly fit the remaining width. The final design has a 100% utilization, as required.

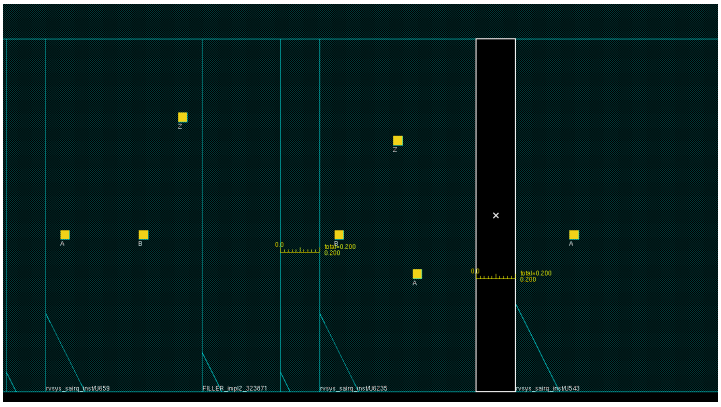


Figure 8.12: $0.2\,\mu m$ gap not filled between standard cells

8.9.2. Metal fill

During fabrication, the metal wires are built on top of each other. However, due to an imperfect design process, there is a difference in height between metal and no metal. This can cause discrepancies on the higher metal layers, causing the wires to break. To resolve this problem, a constraint is to actually fill most of the design with metal. This will cause all layers to be imperfect in the same way, preventing a discrepancy. The correct settings for metal fill can be found as part of the TLR and the default metal fill script of Encounter is good enough. If necessary, the way metal is added could be changed, but that was not required.

8.10. Design optimisations

After an initial run is complete, several optimizations were performed. These include the described set-reset inferring and switching activity annotation. Many more small optimizations were performed, which can be found in the script files. As they are rather small not everything is described here.

8.10.1. Set-reset inferring

During design, all flip-flops appeared to be of the type DF, the default flip-flop. However, when looking at the standard cell library [67], about 34 different types of flip-flop exist. further investigation pointed out the root cause. By default Synopsys does not recognize set and reset signals. Almost all flip-flops in the design should have such a signal, but they were not replaced during synthesis. Instead, these signals were all generated using logic.

The solution was to annotate the netlist. The set and reset signals are generated and an attribute has to be set on the generated signal. By using the 'sync_set_reset' attribute, for a synchronous set/reset signal, the flip-flops were correctly replaced, causing a healthy number of different types to be used. The result is a design that is about 2% smaller and marginally faster and less power consuming as well.

8.10.2. Synthesis switching activity

As discussed in Section 4.1.6, feeding back real life simulation data into the synthesis step can improve the result. See Table 8.11 for the effect of using an SAIF file. The total improvement of using an SAIF file is in this case a noteworthy 14%. It might be possible that this improvement is simply due to a lower switching activity than originally estimated. When the design is not annotated, Synopsys takes a default toggle rate of 20%, which is of course never exactly the case.

When investigating further, the power consumption of the cache instances show interesting results. Their power consumption is actually worsened by 34%, indicating an increase in switching activity. Therefore, it can be assumed the power consumption of the logic is actually successfully improved. Note that this comparison was performed using an intermediate design stage, therefore, power numbers slightly differ from the results presented in Table 8.1.

Table 8.11: Using simulation data to improve synthesis results

Clock frequency (MHz)	SAIF?	Power chip (mW)	Power cache (mW)
100	no	48.25	20.45
100	yes	41.39	27.46

8.11. Chip considerations

Some design steps have been omitted or will be required in the future. For this project they have a clear cause, but this may change. Therefore, the most noteworthy considerations are listed here.

8.11.1. Die seal ring

The die seal ring is an important part for improving IC reliability. It is a ring surrounding the whole chip and consists of two parts. The first part is a full filling of the surrounding, 'sealing' the IC. This prevents moisture from slowly entering the active area and causing damage to the metal. The second ring is a specialized scribe line reinforcement pattern. In short, it reduces the stress from die-sawing and packaging. Without this ring, stress might damage the chip, causing it to fail at some point. More information about the die seal ring can be found in the TLR [68].

The die seal ring has not been implemented for this design. This is because of the used IMEC MPW run. In this run, all designs are combined in a pre-specified model. This model also includes the die seal ring and, therefore, doesn't have to be manually added.

8.11.2. Power-up sequence

During chip power-up, it is recommended to first power the IO pins and subsequently the logic. This will prevent random 'data' on the input pins from causing the internal circuitry to behave unexpectedly. Unwanted effects include large current flows and latch-up. Therefore, when designing a PCB for the ρ-VEX ASIC, VDDIO has to be powered up before VDD, VDDANA12 and VDDANA25. On top of that, the data book defines another rule. "In order not to activate the ESD protection circuit in power up, the rise time of each supply should be at least 100us" [20].

Furthermore, the dual port SRAM used for the caches also defines a power-up sequence. "There are 2 ns of dummy cycles that are required to allow the memory redundant logic to settle after the red registers are loaded" [36]. Therefore, if the redundancy option of the memories is implemented in the future, power-up sequencing becomes important.

8.11.3. Bit sliced register file

In [51], a similar standard logic cell register file implementation was used. This design also had problems during routing. Their solution involved a manual placement of the register file, to aid the software in optimizing the design. Furthermore, the register file was split in four 'bit slices', allowing further optimizations. For a future placement of the ρ-VEX register file this might also be an interesting improvement.

8.12. Conclusion

To implement the RTL to GDSII flow for the ρ-VEX, a set of TCL scripts is created. Every step and optimization is automated, allowing for the results to be easily reproduced, as required for (**G**.6). This also includes verbose comments throughout the code and an extensive user guide to aid new designers in continuing the project. The scripted environment is set up such that it can be easily configured from one file and even allows other projects to be implemented.

All synthesis and P&R constraints are implemented as required for the general flow. Timing is underconstrained by up to 30%, allowing for an energy reduction of 85%. Simulation data is collected in an SAIF file and used to reduce power consumption by another 14%. The final design is low-power (**G**.5b) while still being faster than the FPGA, keeping the design useful (**G**.4).

All steps required to achieve error-free routing and meet timing requirements in P&R are discussed. These include manual optimization of cache instance placement, module constraints for the register file and HSI, module padding for congestion reduction and cell padding for hold violation repair. To improve timing, Clock Concurrent Optimization was required, due to the imbalanced pipeline. Furthermore, an elaborate routing flow consisting of four repair steps was necessary to completely route the congested design. In total, placement requires one ECO iteration, CTS two and routing four iterations.

To improve reliability, as required by (**G**.5a), vias were replaced by multi-cut vias. Furthermore, conform to the UMC design rules, standard cell filling and metal filling are applied.

Finally, some chip considerations are stated. These are important for other designers working with the project (**G**.3c). For the PCB, power-up sequencing is required to prevent unwanted signal switching. A future design might need internal sequencing if the SRAM row redundancy option is implemented. When a next iteration of the ASIC is not fabricated using the MPW service of IMEC, it is important to manually add a die seal ring. Finally, an improvement (**G**.3c) to the placement and routing of the register file might include bit slicing, a technique that improved a similar design [51].

9

Verification & Signoff

After the RTL to GDSII flow is finished, a correct design should be ready. In principle, this GDSII file can be sent to IMEC for production. However, there is no guarantee that the final circuit is performing the same task as was intended with the initial netlist. Errors in the design can cause the software to fail. Otherwise, it might warn about possible mistakes, but still continue and give 'a' design. This can be different from the intended design. Some reports and problems are discussed in Section 9.1.

Even if the software is fully able to understand the code and doesn't warn about anything, the design can still be incorrect. Ambiguous notations can cause the software to implement a different function than required. Or an undetected error in the initial netlist is happily converted to an error in the P&R output. Even though the initial and final netlist implement the same function, this is undesired. Therefore, Section 9.2 will discuss HDL simulation using SDF timing in ModelSim for all different design stages, to verify the functionality is still as intended. It also explains how to debug the design and gives the major improvements made to correct fatal errors.

The final steps that are discussed are called Design for X (DFX). These include augmenting the design with hardware specialized for test (DFT, Section 9.3) and ensuring reliability during the product lifetime (DFR, Section 9.4). Finally, even if the GDSII is correct, it might be difficult or impossible for the foundry to fabricate. Therefore, there is a set of rules that help in manufacturing (DFM, Section 9.5).

Signoff is the collective name of all checks that need to be valid before the design can actually be manufactured. This includes the HDL simulation, DFM steps as well as an extended set of timing analysis. All steps and their results are summarized in Section 9.6.

9.1. Implementation reports

Both Synopsys DC and Candence Encounter generate extremely large log files. These are megabytes in size and can grow up to gigabytes. Sometimes the amount of data becomes overwhelming. Therefore, it is useful to generate separate reports, containing small bits of essential information. In the scripted flow as described in Section 8.1, this report generation is also added. To see all reports, take a look at the project or the software documentation. This section will only describe the reports that actually led to repairing significant mistakes.

9.1.1. Synthesis-ready design

As will be discussed in Section 9.2, ModelSim will be used to verify the correct functionality of the design. This is also done for the initial netlist, as provided for the project. However, a simulation permits extra functionality aiding into debugging. This includes instantaneous signal changes and global transition sensitivity. This is not possible when the design is going to be synthesized. Even worse, different tools, like synthesis for FPGA, can interpret these functions different.

Therefore it is important to separate these simulation commands from the actual netlist. ModelSim is able to report simulation only commands, using the *-check_synth_ready* switch. A helper script is added to the project, which can be called using *check_asic*. It will compile the whole source netlist and print warnings where mismatches might occur between simulation and synthesis results. This script was used to correct flaws in the ρ-VEX design.

Quite a lot of warnings still persist. These have to do with negative ranges, for example *std_logic_vector(0 to -1)*. Clearly this is an empty vector. During synthesis, this will be optimized away, therefore a warning is generated. These negative ranges are intended and caused by the way the ρ-VEX is constructed. Due to its design time flexibility, parts of the circuit can be turned on or off. Depending on the design configuration, these negative ranges used to indicate absence of a component. Therefore, optimizing them away is the correct behaviour and the warnings can be safely ignored.

As will be shown in Section 9.2.4, the idealized instantaneous switching of simulation caused another mismatch. This can't be verified without actually running a correct simulation.

9.1.2. Combinatorial loops

A combinatorial loop is a circuit that loops back to itself, without a sequential element in between. For example, see Fig. 9.1. Depending on input A, this circuit can be unstable. As the synthesis tool does not know the value of A, it does not know when the circuit settles. It might be infinitely toggling, rendering the calculation of setup timing impossible. In some cases, a combinatorial loop is intended, but most of the times it is a design mistake. DC can report all combinatorial loops.

In the ρ-VEX, a combinatorial loop also existed. What DC does is simply break the looping wire during timing calculation and still synthesize the circuit. In this case, however, it would have caused an instability and therefore invalid design. From the report it was possible to localize the mistake and correct it in the original netlist.

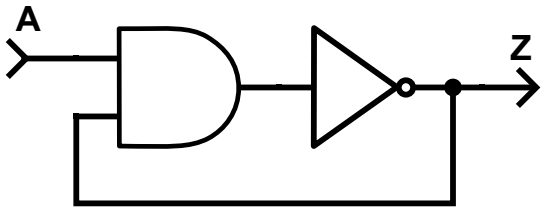


Figure 9.1: Example combinatorial loop, causing instability

9.1.3. Cross-clock boundaries

The cross-clock boundaries are an extremely error-prone part of the design. Therefore it is crucial to check whether the timing is actually correct. Both DC and Encounter are able to generate extended timing reports with all worst-case paths. Furthermore, point to point timing information can be requested. With the initial configuration, correct timing on the cross-clock boundaries was impossible.

As the clock network in DC is ideal, the cross-clock boundary is ideal as well. This causes the timing to always be correct. However, in Encounter this timing has to actually be met. Initially, the synthesis constraints were too optimistic. Only after CTS it was reported that the boundary could not fix setup timing and hold timing at the same time. However, separately they could be met. Therefore not a single problem was reported. Only after reporting the specific timing paths this problem was identified. The solution was a more pessimistic timing model in synthesis, causing it to focus more on generating an optimal boundary. Currently, an extremely detailed set of reports, split over 43 different files, is output to check for any timing error in the design.

9.1.4. Geometry

As described in Section 8.8, Encounter will happily route an invalid design, if it results in better timing. After routing it will report zero errors, even though they are actually there. The geometry check aided in finding these mistakes, allowing the routing to be deleted and improved again.

However, even after doing so 10,000 wiring errors persisted. Every reroute completed successfully, but the geometry report gave conflicting results. The mistake was finally found as part of the FDK, in a file provided by UMC themselves. In the technology file, Encounter was instructed to space a multi-cut via between ME7 and ME8 by $0.75\,\mu m$. Furthermore, a rule was stated that a multi-cut via must be generated for wires wider than $1.0\,\mu m$ and lie within $0.6\,\mu m$. Therefore, all ME8 wires wider than $1.0\,\mu m$ (which is literally every wire) got a generated multi-cut via with spacing of $0.75\,\mu m$ and Encounter was

happy with it. However, when checking it noticed the vias were spaced more than $0.6\,\mu m$, which was wrong according to the same rules.

The corrected mistake was actually a misplacement of the multi-cut via rule. This wasn't supposed to affect ME7 (causing a multi-cut via between ME7 and ME8), but instead affect ME6 (causing a multi-cut via between ME6 and ME7). The correction was made to the FDK. Furthermore, other rules were compared with those stated in the TLR to prevent similar mistakes from happening. The final FDK file can be ensured to be correct.

9.2. Netlist simulation

The original netlist was provided with with a testbench to simulate and verify the functionality of the ρ-VEX. First, the testbench is explained, as well as the changes made to the testbench. Next, the original netlist simulation and its flaws are discussed.

Throughout the project, three additional simulations are run, as also indicated in Fig. 4.1. Before synthesis, some components of the original netlist are replaced. These replacements are verified in the pre-synthesis simulation. After synthesis, another simulation is run, this simulation can be performed with timing information, causing some bugs to be found. Subsequently, the post-route simulation is performed on the final design. The functionality does not change, but timing information is different. Due to the problems with simulation, formal verification is discussed as a possible alternative.

9.2.1. Testbench and changes

The originally provided testbench simulates a combination of the ρ-VEX as well as the connected FPGA controlling it. This has the advantage that the communication protocol is fully tested. On top of that, any software program to be run on the ρ-VEX can be compiled and loaded on the FPGA, allowing for a wide variety of functionality to be tested. However, the testbench had a fixed clock frequency for both the core clock and HSI clock, only one program could be simulated for a fixed period of time and changing the ρ-VEX core was difficult.

Therefore, the testbench was expanded upon. First of all, the core clock frequency, as well as the operational mode (fast, slow, UART or scan) can now be configured from the command line. The clock division ratio depends on the operational mode and is set accordingly. Furthermore, the software program to be executed by the ρ-VEX can be changed. This allows a wide variety of tests to be simulated without recompiling the design.

Another change is the inclusion of a configuration for the design stage. All simulations as described below can be compiled to separate libraries. This allows the testbench to swap out the RTL netlist for the synthesis or P&R gate level netlist, again without any required recompilation. Finally, an automated check is performed to see if the executed program is finished. This is not a straightforward fixed timing, as the runtime depends on operational mode, clock frequency, executed program as well as cache statistics. An easy switch output by the ρ-VEX is used to determine the programs state.

The combined set of changes is used to verify the ρ-VEX with a wide variety of programs, configurations and tests, automated without requiring user input. This is used to increase the functional test coverage as discussed in Section 9.6.1.

9.2.2. Original netlist simulation

The original netlist simulation is supplied as described in Section 9.2.1. The standard compiled program used in the simulation is self-checking, meaning it outputs a message whether the simulation was successful or not. Using the provided settings, the simulation indeed successfully completes. The output waveform is used as golden file, meaning any discrepancy between this waveform and any of the following three simulations can be used to debug the design. This golden waveform is shown in Fig. 9.2.

9.2.3. Pre-synthesis simulation

Before synthesis to an ASIC design, some changes were made to the original netlist. Most notably, the SSI instance. A simulation for the instance was not provided, therefore, it had to be developed as part of this work. The interface description, as well as the timing information required for SDF annotation are implemented as already described in Section 5.4.3. Using this simulation the discrepancy between the provided netlist and SSI implementation have been corrected. The behavioural simulation of the

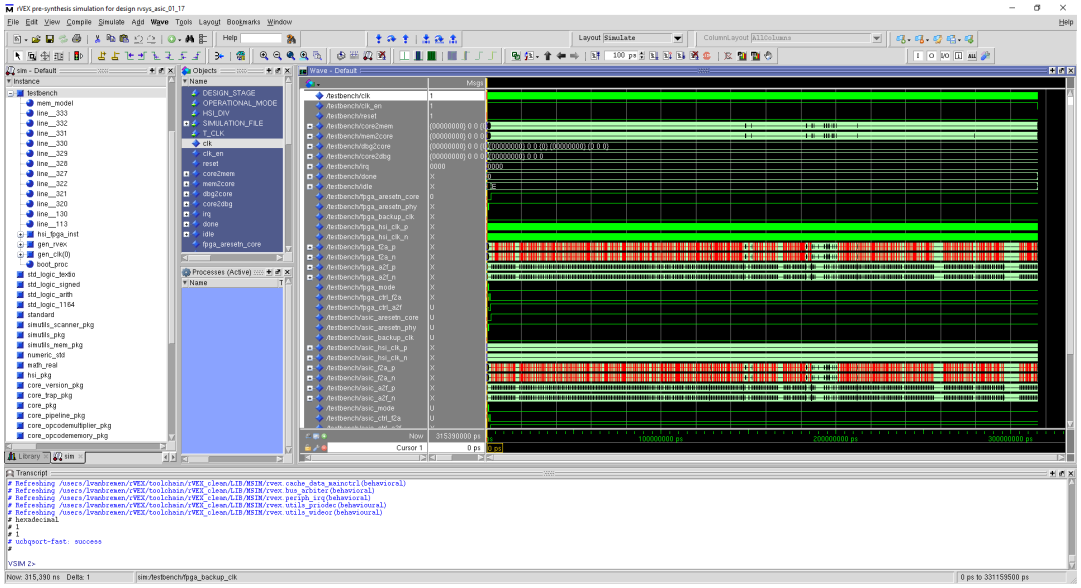


Figure 9.2: Golden waveform from original netlist simulation

caches has also been replaced by simulation files as generated by the Faraday memaker. This allows verification of the correct cache size, functionality and timing information.

An addition to the ASIC are the bond pads, as placed in the IO ring. These bond pads are also simulated, allowing to check for correct configuration of the digital pins. Furthermore, timing information can be verified. For this project pin timing is insignificant and, therefore, not required. The analog pads, as connected to the SSI instances, are slightly different. These do not contain a simulation version, but are only a signal sink. This complicates the simulation as it is impossible to use them for testing the HSI protocol. Instead, specialized test-only virtual pins are added to the design, bypassing the analog pins for simulation. These pins cause warnings throughout synthesis and P&R, as they don't actually exist. However, they do allow the full interface to be verified, quite important for the design.

The output of the pre-synthesis simulation is the same as the original netlist. Therefore, the program again outputs a success message. Furthermore, the waveforms are compared to ensure they are the same. There actually is a small timing difference due to the implemented simulation instances, which better reflects reality. Luckily, this added timing information does not cause problems.

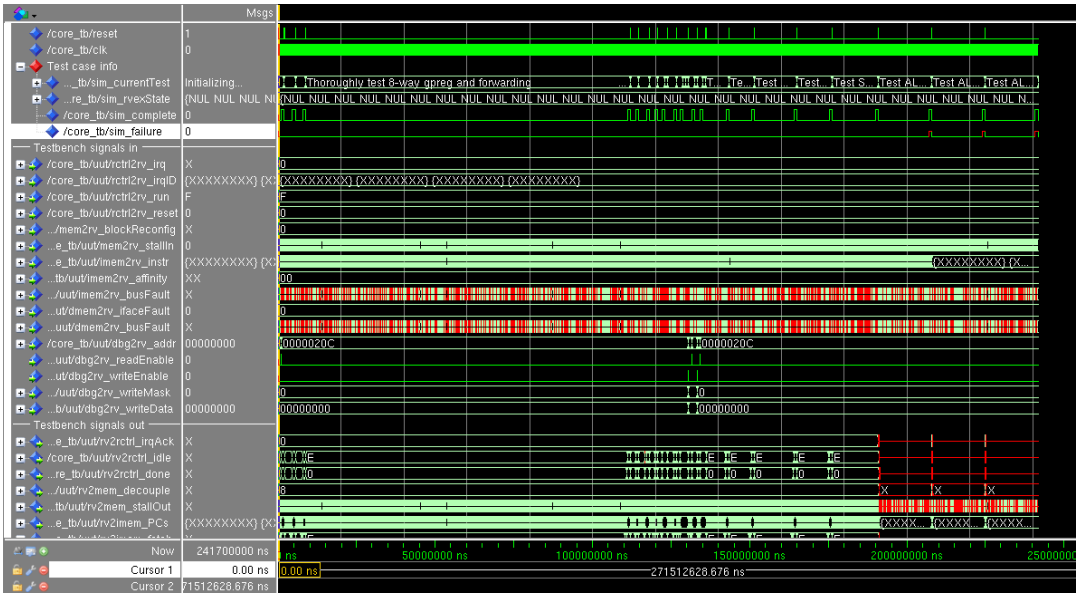
9.2.4. Post-synthesis simulation

After synthesis, the whole RTL netlist is transformed to a gate level netlist. All gate and added nets have generated names, being very ambiguous. Debugging internal errors is extremely difficult, as the waveforms can't really be compared. However, this was still required as the initial design contained some errors. A handful of small timing related errors is corrected, but only two simulation bugs and two crucial mistakes are presented here.

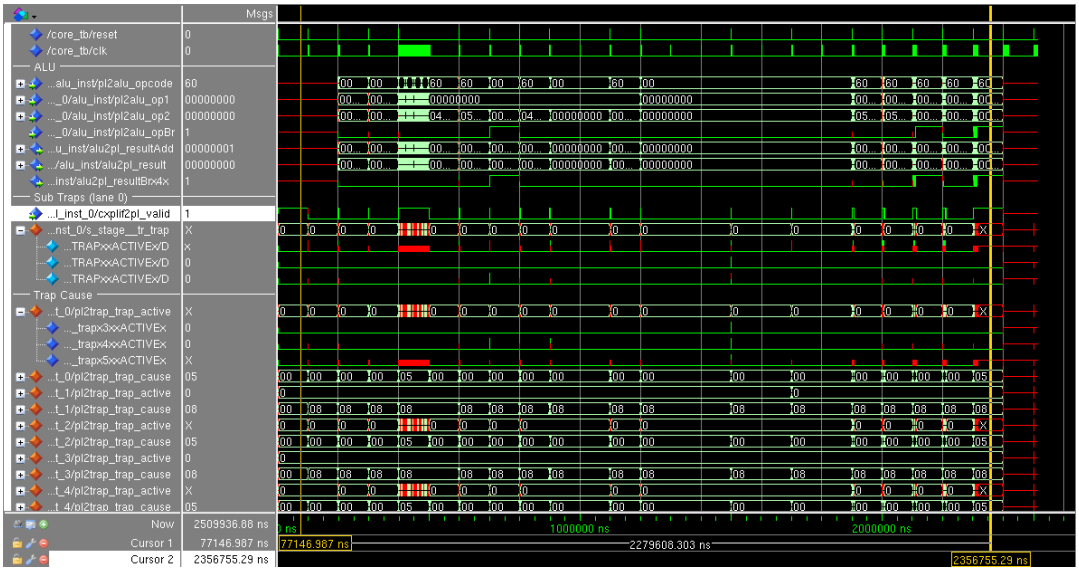
Flip-flop simulation mismatch

Initially, the post-synthesis simulation of the ρ-VEX core failed to complete a set of tests. See Fig. 9.3a for a global overview, after a few tests the core starts to output X-es, indicating a wrong state. After debugging the internal signals, tracing back where the X-es come from, the state as in Fig. 9.3b was identified. In even more detail, the problem always seemed to be triggered by an enable signal.

When singling out the exact flip-flop type in a separate simulation, the waveform as shown in Fig. 9.4a was constructed. Somehow, the output Q was forced to X at every input change. When enable was held low, this output even stays X. As can be seen in Fig. 9.4b, this doesn't occur for the verilog simulation, as opposed to the VHDL simulation. The VHDL simulation library contains incorrect definitions. Therefore, the verilog libraries have to be used.

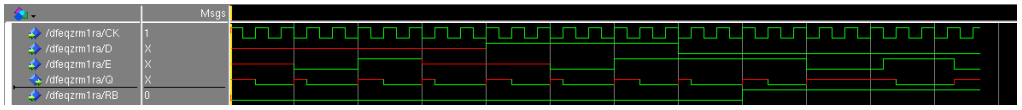


(a) Global signal overview

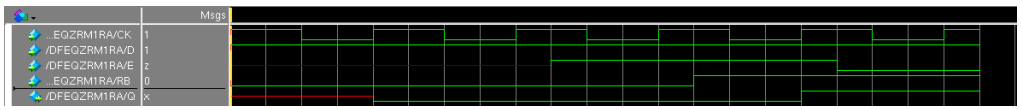


(b) Local signals, zoomed in on error

Figure 9.3: Post-synthesis simulation result fails in specific tests



(a) VHDL simulation of a flip-flop



(b) Verilog simulation of a flip-flop

Figure 9.4: The same flip-flop simulation causes incorrect behaviour in VHDL

X-pessimism

Throughout the simulation, some signals are don't care. No matter what their value is, the ρ-VEX should operate correctly. One example is the initial value of the caches. When the ASIC is powered up, there is 'no' data in the SRAM instances. All values are unknown. This is handled correctly in the ρ-VEX cache using valid bits, indicating the content is invalid. Before the data is used, it is masked with these valid bits. However, as can be seen in Fig. 9.5, the X-es are sometimes still propagated to the core, causing the simulation to fail.

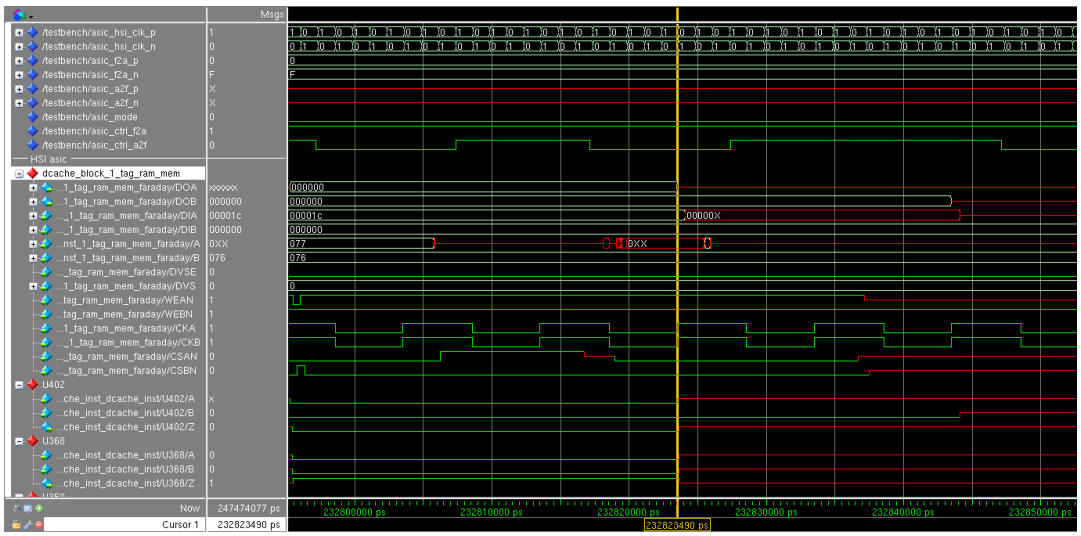


Figure 9.5: X-propagation from cache towards the core

The reason for this failing is actually a limitation of the simulation. First of all, after synthesis all transition timings are annotated using the SDF file. For an inverter, a zero to one transition might take $1.2\,ps$ and a one to zero transition might take $1.4\,ps$. This is the timing the synthesis tool will use. When two inverters are placed in series, a zero to one to zero transition might occur, or a one to zero to one. In both cases, the worst timing is $1.2 + 1.4 = 2.6\,ps$. However, synthesis doesn't know about X. During simulation, an X might occur. Because timing data is not available, the worst-case scenario is taken. For the double inverter, the worst case propagation is thereby $1.4 + 1.4 = 2.8\,ps$. Therefore, the maximum timing of $2.6\,ps$ is actually overstepped, causing incorrect behaviour.

Another example is given in Fig. 9.6. The input is X. This means in real life, it can either be zero or one. In both cases, the output of the XOR gate is zero. However, this is not described in the simulation. In simulation, when either one of the inputs is X, the output will also be X, causing an X to be incorrectly propagated. This case is called X-pessimism and is extensively described as simulation limitation [69].

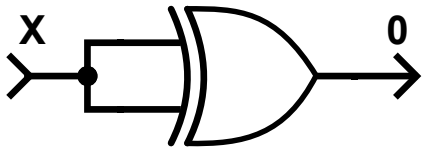


Figure 9.6: X-pessimism causes simulation bugs

The solution here is to fill the simulated SRAM cells with values other than X. A separate simulation is run for all zeroes and all ones as data, in both cases the post-synthesis simulation runs correctly. Therefore, reinforcing the idea that X-pessimism is simply a simulation artifact. In the final simulation the SRAM data is completely randomized, as will happen in real-life.

Reset bugs

An actual mistake in the ρ-VEX code is created due to an optimizationn' of the reset signals. Initial values are set extremely sparingly, even though the improvement is very minor. See for example Listing 9.1. Part of the code is left out.

In the code it is visible that signals ‘arb2rv_mux’ and ‘arb2irq_mux’ are flip-flops. However, they do not have a reset value, meaning their initial value is ‘X’. This can also be seen in the waveform in Fig. 9.7a, where the signal ‘arb2rv_mux_reg/Q’ is X, just before the cursor. Only after the first clock cycle, it gets a value, as set in the code. However, the value ‘writeMask_r_reg_0/EN’ depends on the arb2rv_mux signal. In the original simulation, this code-snippet works. In the waveform, it is visible that the writeMask_r signal becomes X. The discrepancy is related to the added timing information, gained after synthesis.

In the original simulation, after reset is de-asserted, arb2rv_mux instantly gets a value assigned. This value is instantly propagated to the caches and writeMask_r can instantly use it. Using real-life timing information, it takes a non-zero amount of time for arb2rv_mux to get a value assigned. Therefore, during the clock edge, writeMask_r uses the previous value of arb2rv_mux, still being X.

Another effect playing a role are clock gates. To reduce power consumption, unnecessary branches of the clock are turned off. Usually, a second clock cycle would correct the X on the writeMask flip-flop. However, the clock gate only passes through one clock edge, the behaviour that was described in the netlist.

The solution is extremely simple. By adding a reset value for both arb2rv_mux and arb2irq_mux, the waveform of Fig. 9.7b is formed, showing the correct behaviour. Throughout the code, multiple reset signals had to be added to correct similar design errors.

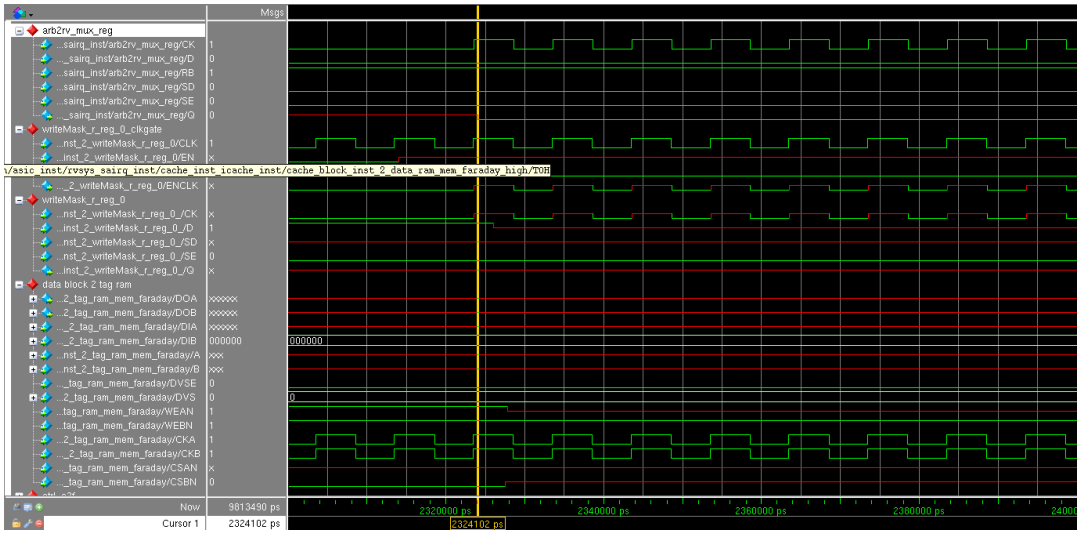
Listing 9.1: Reset signals in rvsys_sairq.vhd

```
1  if rising_edge(clk) then
2    if reset = '1' then
3      -- Reset the dema2arb and arb2dema signals.
4      dema2arb_readEnable_r  <= '0';
5      dema2arb_writeEnable_r <= '0';
6      dema2arb_phase         <= "00";
7      arb2dema_busy          <= '0';
8      arb2dema_ack           <= '0';
9
10     -- Reset the dbg2arb and arb2dbg signals.
11     dbg2arb_readEnable_r   <= '0';
12     dbg2arb_writeEnable_r  <= '0';
13     dbg2arb_phase          <= "00";
14     arb2dbg_busy           <= '0';
15     arb2dbg_ack            <= '0';
16   elsif clkEn = '1' then
17 ...
18     arb2rv_mux <= '0';
19     arb2irq_mux <= '0';
20     if dbg2arb_readEnable = '1' or dbg2arb_writeEnable = '1' then
21       arb2rv_mux <= not dbg2arb_address(13);
22       arb2irq_mux <= dbg2arb_address(13);
23     end if;
24   end if;
25 end if;
```

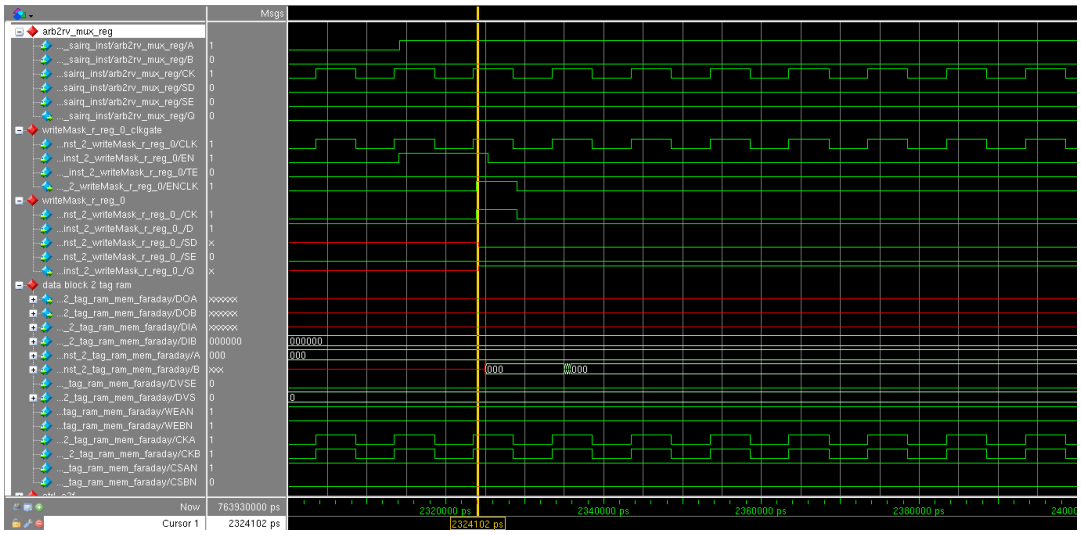
LVDS interface DDR mismatch

The LVDS implementation also had problems in the communication protocol. Before synthesis, the waveform of Fig. 9.8a was acquired. After synthesis and SDF timing annotation, Fig. 9.8b was visible. Both waveforms are of the exact same time instance with the exact same data. However, due to the added timing information, the vale on D is read on the clock falling edge (and output on Q2) instead of read on the clock rising edge (and output on Q1). The next data pulse shows the same, inverted problem.

Looking at the protocol in Appendix B, two data packets are described. By default, the first bit of a package should be sent at the rising edge of the HSI clock, this bit is supposed to be on the Y line (see Figs. 5.3a and 5.4a). The bit can subsequently be received by the ρ-VEX IDDR on the Q2 input. If,



(a) Core signals without reset



(b) Core signals with reset

Figure 9.7: Post-synthesis simulation of arb2rv_mux flip-flop, without and with reset

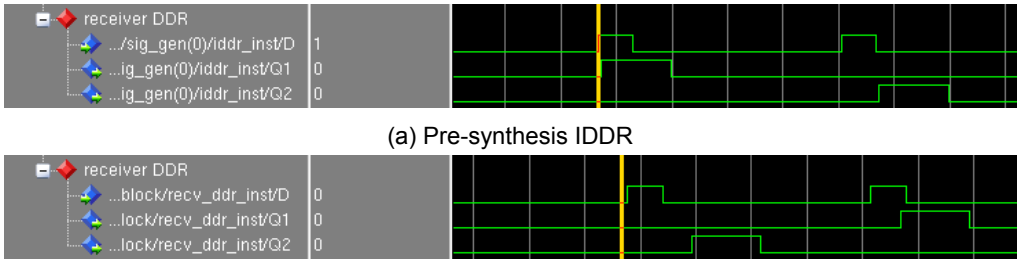
due to imperfect timing the bit is delayed by half a clock cycle, it is received on the X line, or Q1 input. The second case can be detected and the packet can still be reconstructed successfully.

However, in the original case as shown in Fig. 9.8a, the first bit is not received on Q2 but on Q1. Therefore, the fallback mode is already simulated. The post-synthesis simulation indeed delays the data by some non-zero time. As the fallback mode is already used by default, the packet is received out of order. The data is incorrect and fails to be received.

Looking at the provided netlist, the packet description in Appendix B and the ODDR description in [30, p. 94,122], the mistake is made in the provided netlist. Because timing information is not factored in the original simulation, this bug was undetected. However, it would cause a fatal mistake in the ρ-VEX ASIC. The solution is simple, for both the IDDR and ODDR the X and Y connections have to be swapped.

9.2.5. Post-route simulation

Because DC does not synthesis the clock tree, the post-synthesis simulation also assumes it is ideal. Therefore, hold violations still persist and should be ignored in the simulation. After the P&R flow, the clock tree should be correct. Furthermore, additional buffers are inserted to repair hold violations and overall timing is different. Therefore, the post-route simulation with SDF timing annotation is required.



(a) Pre-synthesis IDDR

(b) Post-synthesis IDDR

Figure 9.8: LVDS communication protocol DDR receiver timing mismatch

The most notable difference with post-synthesis simulation is that the cross-clock boundaries can actually properly be simulated. This was also crucial, as they contained a design breaking bug.

Looking at the code in 'hsi/common/hsi_dll_rx.vhd' and the description in Appendix B, there is a very small discrepancy. A hold register is responsible for making sure the data in the HSI clock domain is stable until it is read in the core clock domain. However, the 'ready' signal responsible for holding the data was absent. Instead, it was mistyped a few lines below, as part of the HSI synchronization register. Funnily enough, the functionality of the synchronization register doesn't change due to the addition of the ready signal.

As the original netlist was simulated without timing data and the post-synthesis simulation assumed an ideal clock network, all data was always transferred instantaneously. Therefore, it was not necessary to keep the data stable. However, in the post-route simulation, as well as in real-life, the core clock and HSI clock will never arrive at the exact same moment. Therefore, the post-route simulation was broken.

The hard part was, even the post-route simulation was mostly clean. For most data packets, the HSI was not the limiting factor. Therefore, the data was already held stable without the ready signal. Only in a very specific transfer where two data packets are sent at once did the simulation fail, somewhere halfway through the test program. This very specific case made the bug extremely difficult to discover, especially because all signals were indeed connected. However moving over the ready signal to the correct flip-flop resulted in a correct simulation once again.

9.2.6. Formal Verification

Both post-synthesis as well as post-route simulation are using a gate level netlist. This simulation is a very slow and difficult to debug. The original signal names are largely not available anymore and many levels of gates are added. Furthermore, due to the absence of a clock net, the post-synthesis simulation was not even entirely correct. Instead a method called formal verification can be used [70]. Formal Equivalence Checking (FEC), an instantiation of formal verification, is a method to prove a gate level netlist is functionally equivalent to the original RTL netlist [71]. Whenever this can't be done, it is proven there is a design mistake and the exact location in the RTL netlist can be pinpointed. This significantly speeds up verification, as simulation is mostly obsolete. The required software, as well as the knowledge, were not available during this project. Therefore, FEC was not used. However, for future design changes it is highly recommended to add this to the design flow.

9.3. Design For Test

Design For Test (DFT) deals with specialized hardware to test a circuit after it is fabricated. An example is the BIST for dual ported memory, as coined in Section 7.4. The dual port SRAM can be generated in the Faraday memaker. By enabling the BIST option, the SRAM will test each internal memory cell and disables parts that don't work. Due to the increased complexity, the BIST is not implemented. It can be used for a future implementation. Instead, another DFT method is used, called a scan chain.

The scan chains were introduced in Section 7.2.2. By serially connecting all flip-flops in the design, every sequential element can be written through a single input pin and read through a single output pin. By smartly constructing multiple scan chains, the whole register file can be accessed and used for testing the design. DFT fulfills the secondary role of fault correction (**G**.4b).

To implement scan chains, DC is used. Per context, the register file is $1 \cdot 63 \cdot 32 = 2016\,bit$ in size. As all scan chains are clocked at the same clock, they are always shifted in parallel. To make sure no data corruption occurs, every scan chain has to be of equal length. Meaning all flip-flops in the design have to be divided over chains of length 2016. In total, the design contains 28,393 flip-flops. Meaning 13 chains of length 2016 can be constructed, and 169 flip-flops remain. In total, 1,847 dummy flip-flops are added to the design. These have no function, but just extend the last scan chain to be of equal length. Procedurally, all these extend flip-flops are connected together, connected to the clock network and enabled using the scan_enable signal. This is all done automatically in the scripted environment.

The scan chains were tested in the HDL simulation as one of the operational modes. As they are added in DC, they can't be tested until after synthesis.

9.4. Design For Reliability

One of the future applications of the ρ-VEX ASIC was to be implemented in a satellite. For space applications, reliability is extremely important (**G**.5a). As introduced in Section 2.3, the primary reliability concerns in an ASIC design are IR-drop and EM effects. Therefore, DFR is primarily focused on those two aspects. Before the IR-drop and EM effects can be calculated, the Power Grid Library is explained.

9.4.1. Power Grid Library

By default, Encounter is able to calculate a static approximation of the IR-drop. It does so by bulking every logic gate and uses a fixed capacitance value across the whole chip. Only the wire resistance is used in an actual calculation. However, when a lot of devices are switching, this static approximation is way too optimistic. Especially for the HSI running at $400\,MHz$ this is a problem, but also for the regular core area this can cause a discrepancy. In fact, newer versions of Encounter discourage this method altogether.

Instead, a Power Grid Library (PGL) should be used. The power grid library is a collection of all parasitics of devices on the net. It can be generated from SPICE netlists. Some foundrys don't include the transistor circuits of the standard logic cells in their FDK, rendering this method impossible. Luckily, UMC does provide the full netlist. For the SRAM cells, generated by the Faraday memaker, no SPICE netlist is included. Therefore, PGL generation for the cache instances is skipped. The SSI SPICE netlist should be available, but it was not provided. In a future iteration this should also be included in the PGL.

Generating the PGL is somewhat of a trial and error. There are many different formats provided and Encounter doesn't understand them all. Furthermore, the PVT corners are named differently in different files. Therefore, the correct devices (multiple transistor types, diodes and resistors) have to be separately included. Finally, the standard cell library contains antenna cells. As it is nothing more than a connection to a diffusion area, Encounter doesn't seem to be able to simulate it. Therefore, it is procedurally excluded from the PGL, by copying the library files and removing all lines of code related to the antenna cell.

The generation of a PGL for all standard cells (LVT, RVT and HVT) and a separate library for the different IO pads is included in the TCL scripted environment. With a simple command, all required information is collected, repaired and simulated. The trial and error difficulty is hidden, allowing the PGL to be easily used. This generation is also added to the documentation. The script can be found in Appendix E.2.

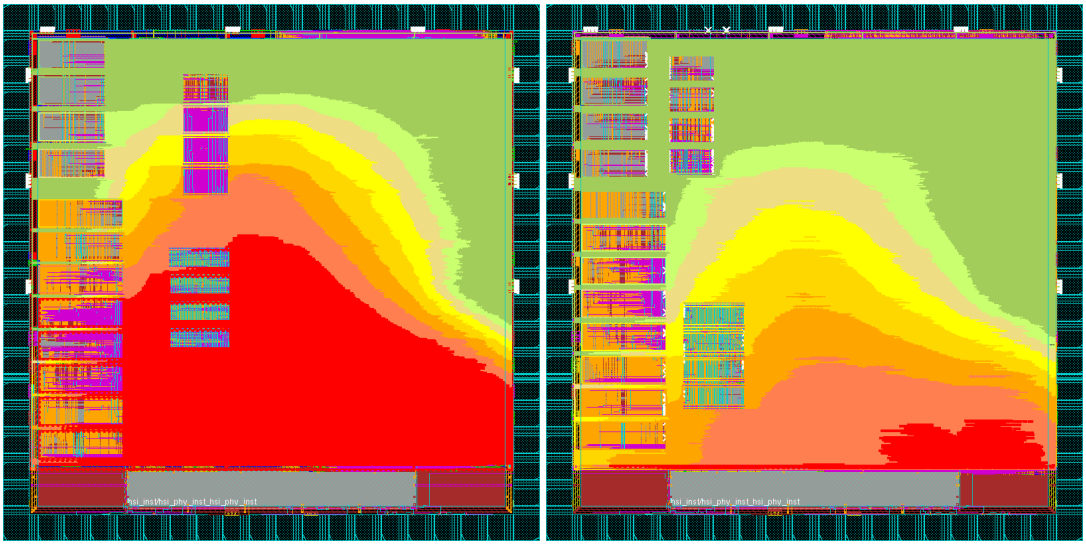
9.4.2. IR-drop

Throughout the project, the IR-drop is constantly monitored. The IR-drop for different iterations of the design is shown in Fig. 9.9. As is visible in Fig. 9.9a, the south side of the chip is a big problem. This is because of the large required area of the SSI. Usually, the power distribution network is connected on the north, south, east and west sides of the chip. This brings the power distribution to the PCB level, where it is a lot easier. Internally, only short connections have to be made. However, the south (and in the final iteration also the north) side of the chip is filled with the SSI, not allowing a single digital pin within the analog power domain.

The absence of a north and south connection are exactly the reason why such an elaborate power routing was required as described in Section 8.5. From iteration 1 to iteration 2, all wires were widened, reducing the IR-drop slightly. The big improvement was made when the horizontal power ring connec-

tions were moved from ME7 to ME8. The effect is visible in Fig. 9.9c. Power distribution in most of the south side is now at least acceptable. Only the centre of the chip still has some problems.

Due to the power routing on ME8, even the north side of the chip would permit two SSI instances, without degrading the IR-drop too much. Furthermore, by implementing large power stripes across the register file, the overall chip IR-drop is contained within 0.99%. The downside of these power stripes was already discussed in Section 8.5.3, causing difficulties in routing the register file. However, for proper reliability, it was absolutely necessary to do so. The final IR-drop is shown in Fig. 9.9d. As the design is timed with a worst-case IR-drop of 10% (including external chip effects), a total of 0.99% should reliably keep the ρ-VEX ASIC running.



(a) IR-drop iteration 1 (b) IR-drop iteration 2

(c) IR-drop iteration 3 (d) IR-drop iteration 4

Figure 9.9: IR-drop through different design iterations

9.4.3. Electromigration

As with IR-drop, the EM effect was closely monitored throughout the project. The biggest improvement in EM reduction was the addition of extra power pairs in the IO ring. This was easily possible, as the ρ-VEX was designed such that it uses the least amount of digital pins as possible. The final EM effects on the whole chip are visualised in Fig. 9.10.

The image looks very similar to the IR-drop, except for a small risk in the top-left corner. However, even this is only a risk on $3.57 \cdot 10^{-7}\%$ of the wiring. In fact, using the reported current density in a calculation as described in Section 2.3.3, leads to a Mean Time To Failure of more than 90 years. This is even more than a satellite would need.

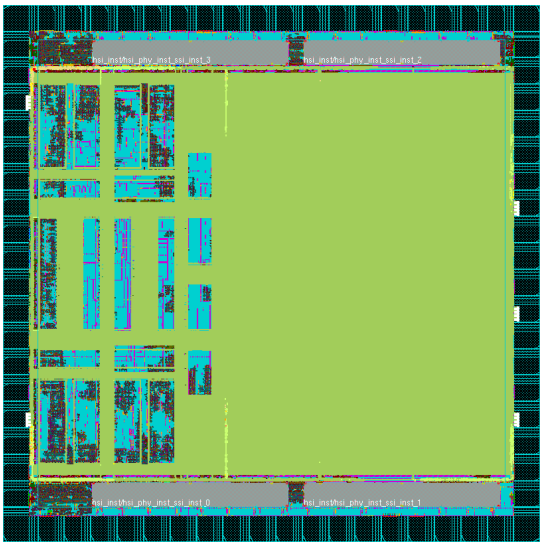


Figure 9.10: Electromigration effects of the final design

9.5. Design For Manufacturability

Before UMC accepts the GDSII for production, it has to strictly meet a set of predetermined requirements. These requirements are listed in the TLR and can be verified using the Design Rule Check, as explained in Section 9.5.1. A special version of this check is also performed in Section 9.5.2. Two other checks can be performed, called the Layout versus Schematic and the Electrical Rule Check. These are not strictly required for manufacturing and, therefore, will not be verified by IMEC. However, they do have a significant impact on the final result, as discussed in Sections 9.5.3 and 9.5.4.

To ensure validity, all checks of this section are performed in the standalone program Calibre. This tool only checks the final GDSII file and, unlike Encounter, doesn't use any data gathered during the design process.

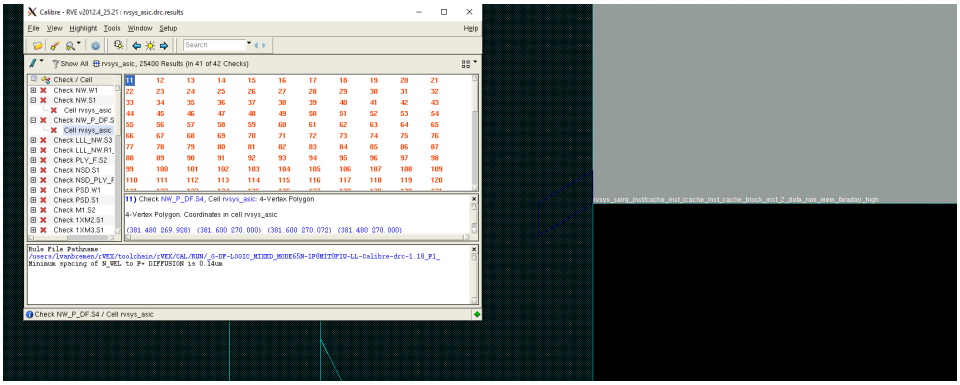
9.5.1. Design Rule Check

The Design Rule Check (DRC) is able to check rules like minimum wire spacing, minimum and maximum wire widths, as well as the metal density. Generated via sized and multi-cut vias are also verified. Furthermore, every single transistor is checked and N-well and P-well spacing have their own rules. As Encounter doesn't deal with these last two layers, it is important to check whether they are actually valid.

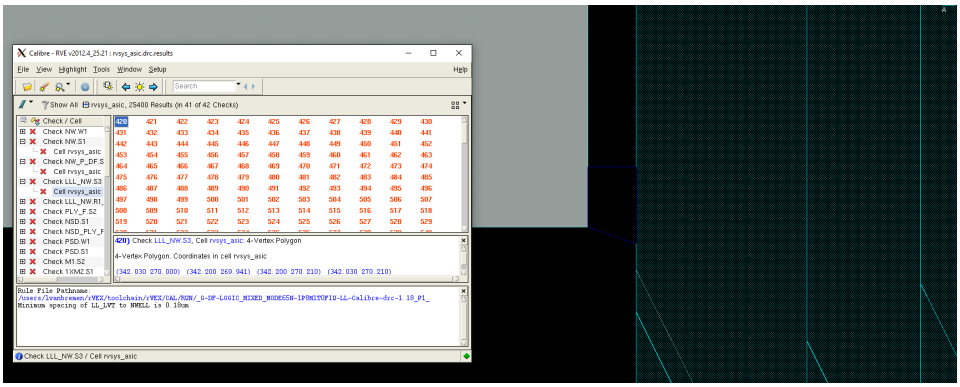
The Calibre rule file for the DRC is handily available as part of the UMC FDK. However, the rulefiles are universal for different metal stacks, different IO cells and requires configuration for the correct input files. All settings can be found in the rulefile itself as well as in several documentation files [14][20][72]. For this project, all information is gathered, the rulefile is adapted and the configuration is part of the TCL scripted environment. Three different types of DRC, the standard cells, antenna violation check and dummy metal (metal fill) are grouped and automatically checked. A menu is added to Encounter to visually inspect any errors, aiding in debugging the root cause. The most significant design errors are listed here.

N-well spacing

As already stated, Encounter is agnostic of the different transistor implementation layers. Using Calibre, these layers can actually be checked. The placement of standard cells caused some Design Rule Violations (DRVs). The violations are shown in Fig. 9.11. The standard library cells are constructed such that they never cause violations when abutted. However, the design also contains some SRAM cells. In all the (thousands of) violations shown in the image, standard cells were placed too close to the transistors internal to the SRAM cells. By default, a halo was automatically generated on the SRAM sides with pins. To make sure the sides without pins don't get standard cells placed too closely, a placement halo of $1\ \mu m$ was forced on every side. This resolved the N-well spacing violations in new design iterations.



(a) N-well minimum spacing 1



(b) N-well minimum spacing 2

Figure 9.11: Minimum spacing violations between N-wells, unknown to Encounter

Stacked vias

Another design rule as described in the TLR is concerned with stacked vias. If too many vias are stacked on top of each other, production can cause the middle via to be destroyed. To ensure a proper connection, redundant vias have to be created for such stacks. Encounter is actually aware of this rule and will do so. However, somehow the rulefile isn't entirely correct and Calibre falsely reports some violations of the rule. See Fig. 9.12. The biggest shown rectangles are stacks of more than four vias. As clearly visible, they are indeed surrounded by redundant vias. Every violation of this rule is checked and should be valid.

The problem is also discussed with IMEC. They will actually waiver the check for this design, however the via stack might cause reliability issues for an actual space application. Therefore, this violations should be resolved in a future iteration. There are actually more DFM rules that can be enabled for improved reliability.

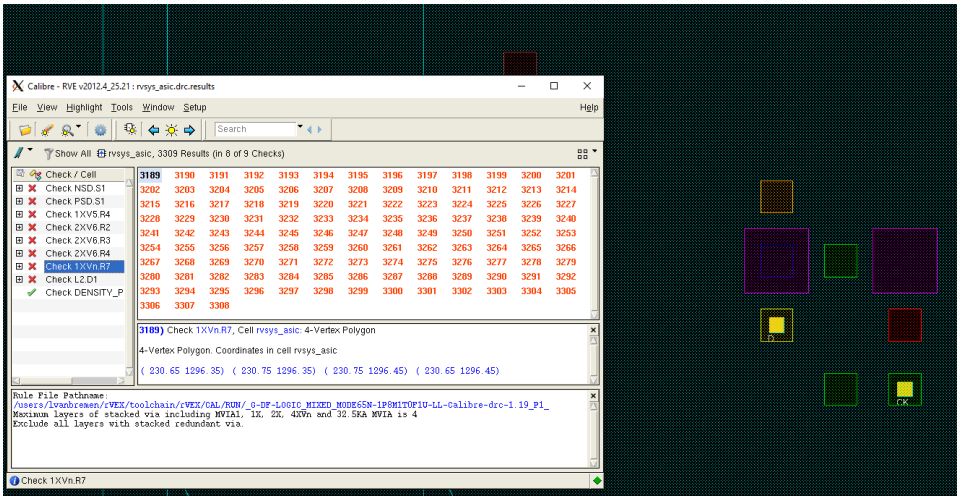


Figure 9.12: Stacked via violation, incorrectly reported by Calibre

9.5.2. Power routing DRC

As discussed in Section 8.5, the implementation of the power plan causes Encounter to generate a large amount of warnings and errors. Due to the forced ME8 power ring, Encounter thinks it will cause a DRV. Therefore, the powerplan is separately checked in Calibre. This initial DRC is used to ensure no problems persist in the later design stages. In fact, a DRC is performed in every design stage, to minimize the chance of unpredicted errors.

The DRC did turn out to be clean in every stage of the design. The results for the separate checks are presented in Table 10.3.

9.5.3. Layout versus Schematic

The generated results were a gate level netlist and a GDSII file for the layout. The functionality of the gate level netlist was verified using simulation. However, there is no guarantee that the GDSII actually functions the same way as the netlist. Therefore, to also verify the functionality of the GDSII, the Layout versus Schematic check (LVS) is required.

LVS consists of three steps. First, the gate level netlist is converted to a SPICE netlist. All gates are replaced by the individual wires and transistors. Next, the GDSII file is analyzed. Every correct combination of N-well, P-well, poly-silicon etc. is identified as device, like a transistor or diode. All remaining poly-silicon and metal is traced and converted to wires. The result is another SPICE netlist. The third step is to compare the gate level SPICE netlist (the source netlist) with the GDSII extracted SPICE netlist (the layout netlist). Any discrepancy between the two will probably cause the layout to implement a different (incorrect) functionality.

The LVS is, like the DRC, executed using a Calibre rulefile. This rulefile is adapted to implement the correct checks for the used technology and metal stack. To create the source netlist, the program ‘v2lvs’ is used. It converts a verilog netlist to an LVS ready SPICE netlist. All required library files to replace the standard cells for transistors are supplied to the program. As the netlists for the SRAM cells and SSI instances are not provided, they are empty in the source netlist.

Calibre is able to extract the layout netlist from the GDSII file. However, the layout contains the full implementation of both the SRAM cells as well as the SSI. Therefore, a special GDSII file is first generated without these instances and fed to Calibre. Note that this does mean that an LVS can’t be performed for the SSI. Therefore, it is expected to be correct. The LVS check needed many more changes, before it was correct. All these changes are added to the TCL scripting environment, fully automating the check and saving a lot of time in the future. Some significant results obtained from the LVS are listed below.

Connection through instance pin

In the SRAM instances, some pins can be connected on multiple metal layers. Internally, a via exists connecting the layers. Encounter can use this to reduce the wiring required when multiple pins are connected together. See Fig. 9.13. In the image, the wire on the left and wire on the right are actually part of the same net. The two pieces of metal are connected together by the instance pin in the middle.

The problem lies in the fact that the SRAM instances could not be included in the LVS. As they are left out, the metal connection through the pin is also absent. Therefore, Calibre detects the single net consist of two separate metal pieces, generating an error. As the connection is there in the actual GDSII file, this error can safely be waivered.

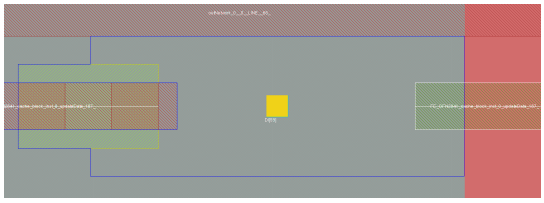


Figure 9.13: Connection through an instance pin, not recognized by Calibre

Separated power nets

The ASIC contains four SSI instances. Two on the north side and two on the south side. Both have an analog power ring, consisting of power nets VDDANA25, VSSANA and VDDANA12. In the source netlist, these nets were the same, as they will be connected to the same power supply on the PCB. However, in the GDSII file they are not physically connected. This discrepancy causes Calibre to again generate an error.

The separated parts of the power net are actually a serious problem. The error causes quite a bunch of other checks to not be correctly performed. Many additional errors are generated, like transistors not being connected to the correct power supply. These 'invalid' errors might mask actual LVS mistakes. Therefore, the physically split power supply rings are given different names. The north ring uses the nets VDDANA25_N, VSSANA_N and VDDANA12_N. The south ring uses equally named nets, ending in _S. The SSI instances are also connected to the corresponding power supply, depending on their vertical position on the chip.

An equal error occurs for the digital IO power nets VDDIO and VSSIO. These nets lie hidden in the digital pads on the east side and west side of the chip. They are connected solely by abutting the IO instances. As breaker cells are added to separate the digital domain from the analog domain, the VDDIO and VSSIO nets are also separated in east and west components. Therefore, these are also given a respective _E and _W extension. This problem does not occur for the VDD and VSS nets, as they are fully connected through the digital power ring.

Absent via

Another serious LVS problem is barely visible. In Fig. 9.15, two horizontal metal parts are visible (the vertical wire can be ignored). From the left side, ME5 is used. From the right side, ME1 is used. These two metal pieces are both part of the same power supply net VDD (the same problem happens for VSS).

As the two metal pieces are aligned, Encounter expects them to be connected. However, it didn't overlap them enough, causing no via to be generated between the different metal layers. Somehow Encounter forgets that it didn't generate a via and even with the built in wiring check this disjunction is not recognized. As both parts are actually connected to the same power supply, the LVS check in Calibre initially also didn't generate an error. However, in one instance, this disjunction happened at two places on the same piece of wire. The result was a floating part of the VDD net, which of course doesn't correctly supply power to the connected standard cells.

The 'phantom' via, still recognized by Encounter but being absent, only happens at the border of all SRAM cells. Sometimes these nets are connected and sometimes they aren't. To resolve the fully disjunct metal piece, one of the SRAM cells was moved very slightly, until Encounter suddenly did create a via. To fully understand the problem, it should be investigated further in a future project. For this project, the problem is remedied as described such that no more LVS errors persist.

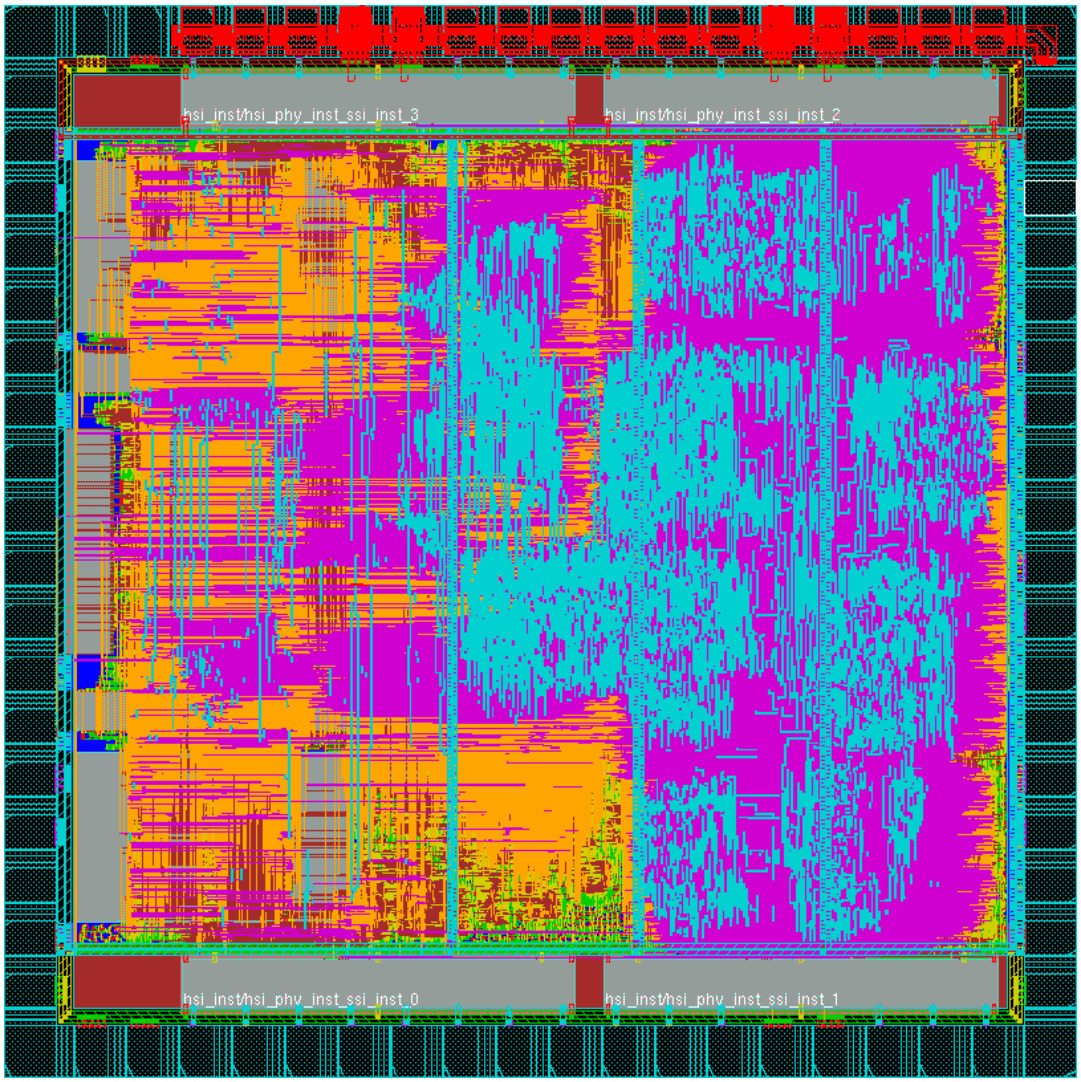


Figure 9.14: Separated parts of net VDDANA, north ring disconnected from south ring

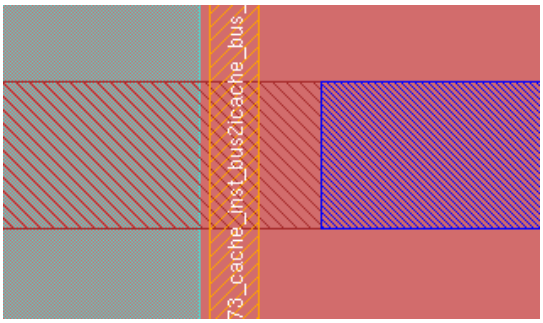


Figure 9.15: Wire overlap too small, causing no via to be generated

9.5.4. Electrical Rule Check

The Electrical Rule Check is concerned only with the power supplies. It functions almost the same and in this FDK it is actually part of the LVS. From the layout, all nets are extracted. From the source netlist, all transistor pins that should be connected to a power supply are marked. Subsequently, the power supply is fully traced from root to all transistors, to check whether a physical connection exists.

The ERC is mostly useful in combination with the well tap cells, as used in this FDK. The well tap cells cause multiple standard cells to share a single power supply connection. As there is no direct connection, it is ignored in all other checks. Only the ERC will validate correct placement of the well tap cells. Therefore, for this design it is highly recommended and by default performed with the LVS. The ERC must report zero errors.

9.6. Signoff

Signoff is the ensemble of checks required before an IC is ready for fabrication. It ensures a valid, functional design and manufacturability. Some of the signoff checks are already discussed in this chapter, but a final overview will be given here.

All required checks for signoff analysis are listed below. A method to ensure the signoff quality is constructed in Section 9.6.6.

1. Functional test coverage, Section 9.6.1
2. Static Timing Analysis, Section 9.6.2
3. IR-drop and EM analysis, Section 9.4
4. Signal Integrity, Section 9.6.3
5. DRC, LVS, ERC, Section 9.5
6. Visual inspection, Section 9.6.4
7. Power Analysis, Section 9.6.5

9.6.1. Functional test coverage

The functional tests are performed as explained in Section 9.2. For signoff, only the post-route simulation is required. As long as the final design functionally works, it will be correct. The provided testbench will run the program 'ucbqsort-fast' by default. This is a short version of the similar named program of the PowerStone benchmark, it is used for testing due to its shorter run time. Using this program, all four different operational modes are tested.

To improve the functional test coverage, some additional programs are run. These programs include 'ucbqsort', 'compress' and 'convolution'. 'ucbqsort' is the full length PowerStone benchmark program, it will sort a list of integers. 'compress' is also part of the PowerStone benchmark, it is used to compress and decompress a fixed string of characters. Both 'ucbqsort' and 'compress' are self-checking, meaning they test whether their output is valid. They can be used to verify the functionality. 'convolution' applies a blur filter on top of any image. For the simulation, the image is just random data. It can't be used to verify functionality, but it is chosen due to its good ILP. The ρ-VEX should be able to complete it much faster in 8-way mode than in 2-way mode. All programs are tested in fast functional mode, with reconfiguration to either 8-way or 2-way. Therefore, the effect of reconfiguration is tested.

9.6.2. Static Timing Analysis

Static Timing Analysis (STA) is actually already performed in Encounter. It is the way setup and hold timings are calculated. However, as explained in Section 4.1.2, Encounter was configured to only calculate the 'best case' and 'worst case' PVT corners. This is done to reduce computation time, but there is no guarantee about timing in any of the other corners. Furthermore, Encounter uses some approximation models, which might be inaccurate. Therefore, Tempus is used to perform the same calculations again. This time with the highest possible accuracy.

9.6.3. Signal Integrity
If two wires on the chip are close together, there will be some capacitive coupling. When one of the wires is switching fast, it will influence the other wire. If this happens too much, an unwanted transition can occur, called a glitch. The Signal Integrity (SI) needs to be checked, to ensure no glitches happen. Again, Encounter already uses a model to calculate SI, but Tempus is used to improve the accuracy.

9.6.4. Visual inspection
At this stage Calibre is already used to verify the design is correct. However, it is still useful to visually inspect the GDSII file. For example, the SSI, which couldn't be checked using LVS, might somehow be empty. Or the bond pads, which aren't visible in Encounter, might be absent. The GDSII file can be opened in Virtuoso. Every layer is now visible, allowing a quick check of the final IC.

9.6.5. Power analysis
Finally, a big goal of this work was to create a low-power design. Up until now, only the reports of DC are used as an indication. However, at that design stage there was no clock net present. An estimation of the power consumption is available, but not too accurate. Therefore, Tempus in combination with the generated PGL library will be used to accurately calculate the final power consumption. Of course, to get the real power consumption, the chip has to be fabricated.

Furthermore, in different applications the power consumption might change. The ρ-VEX will be used to run different software and might be used for more or less parallel designs. As explained in Chapter 3, using less computing resources over a longer period of time can reduce power consumption. Therefore, the switching activity of all simulations in Section 9.6.1 is stored. Subsequently, they can be used in Tempus to get accurate application specific power consumption over the course of the program. By multiplying this with the run time, the total energy consumption for a program is calculated.

9.6.6. Ensuring signoff quality
A main goal of this project was to be reproducible, (**G**.6). This means a followup project needs to be able to achieve the same quality without the effort. Therefore, a full signoff checklist is created as explained in this section. It can be found in Appendix C. All values can be easily reproduced with the helper scripts as explained throughout the chapter. The same checklist is used for this project. The signoff results are presented in Section 10.1.

9.7. Conclusion
Even when the synthesis and P&R software create a design, there is no guarantee it actually works. Different generated reports are used as early check to see if any mistakes happened. This has helped in repairing combinatorial loops as well as ensure correct timing on the cross-clock boundary.

Using simulation in ModelSim, the function of the ρ-VEX was validated. All operational modes, being functional fast, functional slow, UART and scan, are tested using a generalized testbench. Some downsides to the simulation include X-pessimism, as well as incorrect VHDL libraries in the UMC FDK. After these problems were overcome, bugs in the reset state as well as the HSI have been repaired. In a future project it is recommended to use Formal Equivalence Checking instead of post-synthesis simulation. This will help in quicker verification and debugging of the functionality.

The simulation is also used for several PowerStone benchmarks as well as a custom convolution program. Different software in different ρ-VEX configurations will be used to fully verify the ASIC functionality. All the simulations together can be used to characterize the power consumption in different scenarios.

To make sure the final GDSII file is correct, it is verified in external programs. Cadence Tempus in combination with a PGL is used for high accuracy STA, IR-drop and EM analysis and SI analysis. The design is verified using every possible PVT corner. Calibre is used to perform DRC, LVS and ERC checks. Virtuoso is used to visually check the GDSII file, to see if no big mistakes are made.

After all errors are corrected, some DRVs still persist. These are regarding multiple stacked vias and can safely be waivered for this project. For a future project, actually going to space, it is crucial to make sure no vias are stacked anymore. It is even possible to enable extra DRC for improved DFM. If the area permits so, all DFM rules should be followed.

10 Results

The final results of the chip are determined by signoff verification, as explained in Section 9.6. As long as the end result is error-free, it can be fabricated. Therefore, the signoff results are given in Section 10.1. This section also briefly summarizes why some violations persist. Furthermore, the final simulation timing and power results are given.

The second part of this chapter compares the achieved results with other work, in Section 10.2. As this work is a direct port of the ρ-VEX running on an FPGA, the improvements with respect to this FPGA version are listed. Furthermore, the ASIC is compared with similar processors.

10.1. Signoff

For signoff results, the checklist as provided in Appendix C is used. All numbers are acquired as explained in Chapter 9 and can be automatically obtained and reproduced using the auxiliary scripts in Appendix E.1. See Tables 10.3 to 10.5 for the final results.

10.1.1. Remaining violations

The DRC as performed in Calibre is not yet entirely clean. The 163 listed violations are all stacked via violations. As explained in Section 9.5.1, these violations can be waivered. This has also been explicitly discussed with IMEC. A future iteration that will actually go to space needs to have this repaired, as it might cause reliability issues.

The LVS is also not entirely clean. The remaining 3 violations are connections through instance pins, as explained in Section 9.5.3. As the SPICE netlist for the SRAM cells are not available, Calibre can't correctly identify the connection. In virtuoso all these connections are visually verified and are actually correct.

10.1.2. Visual inspection

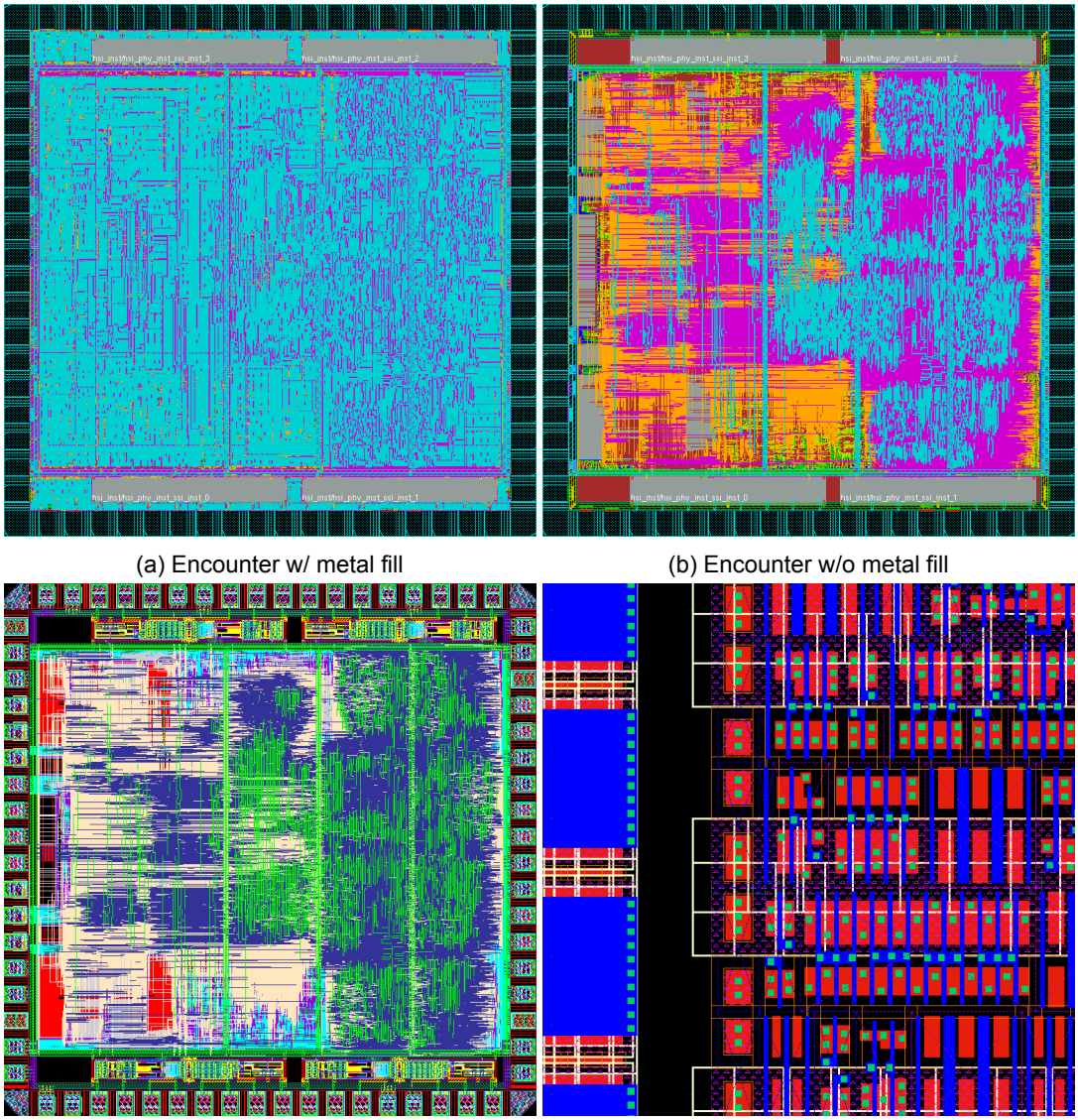
The design is visually inspected in both Encounter as well as in Virtuoso. The results are shown in Fig. 10.1. In Fig. 10.1a, the metal fill is covering the whole chip, obscuring further inspection. Therefore, it is temporarily hidden in Fig. 10.1b. In Fig. 10.1c, the whole chip including SRAM and bond pads is checked. Figure 10.1d is an arbitrary close-up, showing the actual devices.

10.1.3. Area results

The final chip result area breakdown is given in Table 10.1. The utilization is a good indication of the required routing area. It can be used to compare synthesis results to the final results.

The lowest utilization is achieved by the register file. As already stated, this is due to the very high routing congestion. A better register file design can reduce the logic area, as well as reduce the routing. This will significantly reduce the required area.

Furthermore, the SSI as used in the HSI has, with about 66%, also a fairly low utilization. This is mainly due to the cell width. In Fig. 10.1b, the bottom left and top right corners are completely empty. If the SSI would be made thinner such that three or four instances fit one side, this area could better be used.



(a) Encounter w/ metal fill

(b) Encounter w/o metal fill

(c) Virtuoso global view

(d) Virtuoso zoomed in on devices

Figure 10.1: Visual results of the ρ-VEX ASIC

Finally, the instruction and data cache SRAM only constitute to about 18% of the total area. Ideally, the caches should cover as much area as possible, usually more than 50%. This will be possible only when the register file and SSI are improved.

Table 10.1: Breakdown of chip area

Instance	Synthesis area (μm^2)	P&R area (μm^2)	Utilization	% of total
Full chip	1,875,690	2,673,225	70.2%	100%
Register file	435,355	1,116,295	38.9%	41.8%
Instr. cache	37,052	40,716	91%	1.5%
Instr. cache SRAM	305,712	305,712	100%	11.4%
Data cache	60,282	66,244	91%	2.5%
Data cache SRAM	176,408	176,408	100%	6.6%
HSI logic	13,638	15,865	86%	0.6%
SSI	248,000	376,050	66%	10.3%
Pipelanes	163,116	178,855	91.2%	6.7%

10.1.4. Timing results

The final ρ-VEX ASIC runs at $100\,MHz$, about 71% of its full potential $141\,MHz$. Using this frequency, all simulations as explained in Section 9.6.1 are performed. The results are presented in Table 10.4.

Note that the simulation time includes boot and initial and final communication, not only the program. Post CTS was unable to correctly simulate cross-clock boundaries, therefore, it is not run with HSI. Furthermore, due to setup violations the clock period had to be lowered to $15\,ns$. Both result in quite a longer execution time.

On the ρ-VEX ASIC, a large part of the overhead is the HSI. Due to the small caches, many cache misses can occur. For every such miss a communication between the ρ-VEX and FPGA happens. The total effect of this is compared in Section 10.2.1.

A big downside of the current ρ-VEX is its shallow pipeline. The worst-case timing path has 109 combinatorial stages, including the instruction cache SRAM adding $1.647\,ns$. Only due to the used CCOpt it was at all possible to actually achieve the desired clock frequency. Going any higher is going to be a real problem. Other VLIW architectures, as shown in Section 10.2.2, have two to three times the amount of stages. For a future design it is advised to add more stages.

10.1.5. Power results

In Section 9.4.3, the electromigration risk during power-on was determined to be $3.57 \cdot 10^{-7}\%$. This resulted in a useful lifetime of more than 90 years. This is plenty for a space mission, as targeted for (**G**.5). Note that the electromigration risk does not take device failure into account. Therefore, the transistors themselves or components external to the ρ-VEX ASIC will probably reduce the useful lifetime of the final system.

The rest of the dynamic power dissipation on chip is measured using the PGL. The results are presented in Table 10.2. The DC estimate is added to show the relation between an initial synthesis run and the final value. Together with the area increase, this can be used to create an early estimation of the final results. All DC values are consistently too low. This is probably caused due to a poor wire RC prediction.

Note that the macro instances should include the SSI. However, as no transistor level circuit was provided, no internal or static power numbers are available. The SSI is part of the macro switching power, for about 92%. The large increase is due to the constant communication and eight times higher data frequency.

Furthermore, the macro consists of the SRAM cells. The power dissipation of an SRAM cell is mostly determined by its size. Increasing the storage capacity increases the energy consumption. As they also contribute to over 60% of the total power, comparing the total dissipation results with other devices would be mostly a comparison of the SRAM size. Therefore, the SRAM cells are mostly left out of the equation for Section 10.2. The IO pads are generally also not taken as part of the core power.

Therefore, the final core power is a total of $36.68\,mW$ while running the clock at $100\,MHz$. The energy consumption, as used in comparisons, will be $367\,\mu W\,MHz^{-1}$.

The power and energy results for all tested programs is presented in Table 10.5. As the final chip can only be measured from the outside, separating the core and caches is rather difficult. Therefore, these results are including the SRAM and are valuable as comparison between estimated and actual power. The IO power is still left out, as it is physically drawn from separate pads and can be separately measured.

Table 10.2: ρ-VEX simulated power consumption breakdown

Group	Internal (mW)	Switching (mW)	Static (mW)	Total (mW)	Percentage	DC (mW)
Macro	18.14	10.97	0.044	29.43	42.64%	27.65
IO	1.47	1.44	$2.69e^{-5}$	2.91	4.21%	3.29
Seq.	9.24	0.98	0.018	10.24	14.84%	0.0039
Comb.	4.54	15.56	0.15	20.25	29.34%	0.2706
Clock tree	0.58	3.61	0.0022	4.20	6.08%	0.6337
Registers	0.81	1.18	0.0034	1.99	2.88%	3.8339
Total	35.05	33.75	0.22	69.01	100%	35.68

10.1.6. Final chip

The last column in Tables 10.4 and 10.5 is intentionally left empty. The final chip column is supposed to be filled with actual measured data, as acquired from the fabricated IC. As of the writing of this document, the fabrication has not yet started. The design is registered for fabrication in the next available run.

Table 10.3: All performed software checks before tape-out

		Pre CTS	Post CTS	Post Route	Signoff (Tempus)
Synopsys	Setup WNS	0.0			
	Comb. loops	0			
Encounter	Setup WNS	0.234	0.073	0.006	0.003
	Hold WNS		0.065	0.049	0.050
	Early DRVs	4	0	0	0
	Chip utilisation	62.45%	60.93%	61.01%	100.0%
	IR-drop	1.39%			0.99%
	EM effects	$9.63 \cdot 10^{-7}$%			$3.57 \cdot 10^{-7}$%
	SI*			0	0
	Filler DRVs			0	
	Geometry DRVs†			0	
	Wiring DRVs‡			0	
	Antenna DRVs			0	
Calibre	DRC	0	0	(163)§	
	Antennae			0	
	Metal fill			(1)¶	
	LVS			(3)□	
	ERC			0	
Virtuoso	Bondpads	✓			
	Visual inspection	✓			

*Checks for transition glitches caused by switching activity on nearby wires.
†The geometry check is an early form of DRC.
‡The wiring check is a form of LVS, without the actual transistors.
§Remaining DRCs are stacked vias, all manually checked and may be waivered.
¶The remaining metal fill error is internal to a library cell and may be waivered.
□The stated LVS errors are false, as they are caused due to intentionally absent macro instances.

Table 10.4: Program simulation execution time

		Pre CTS	Post CTS *	Post Route	Final chip
ucbqsort-fast	4:1 HSI	$314.9\,\mu s$		$314.6\,\mu s$	
	2:1 HSI	$337.9\,\mu s$		$337.6\,\mu s$	
	UART	$766.0\,\mu s$	$1.24\,ms$	$765.4\,\mu s$	
	Scan mode		✓	✓	
ucbqsort	8-way, 4:1 HSI			$3.7\,ms$	
	2-way, 4:1 HSI			$9.1\,ms$	
compress	8-way, 4:1 HSI			$7.2\,ms$	
	2-way, 4:1 HSI			$9.4\,ms$	
convolution	8-way, 4:1 HSI			$2.70\,ms$	
	2-way, 4:1 HSI			$6.78\,ms$	

Table 10.5: Program simulated power and energy consumption, excluding IO

		Post Route	Final chip
ucbqsort-fast	4:1 HSI	$20.8\,\mu J$	
	2:1 HSI	$22.3\,\mu J$	
ucbqsort	8-way, 4:1 HSI	$244.6\,\mu J$	
	2-way, 4:1 HSI	$601.5\,\mu J$	
compress	8-way, 4:1 HSI	$475.9\,\mu J$	
	2-way, 4:1 HSI	$621.34\,\mu J$	
convolution	8-way, 4:1 HSI	$178.5\,\mu J$	
	2-way, 4:1 HSI	$448.2\,\mu J$	

*Post CTS simulation can only be run in UART mode, resulting in a larger communication overhead. Furthermore, the clock frequency is only $66.7\,MHz$.

10.2. Related work

This work is a direct ASIC port of the ρ-VEX FPGA as described in [34]. The main goal was to develop a prototype version, that could be used as a reference for future work. To accurately set this reference, it is compared to the original FPGA in Section 10.2.1. To further set the reference, other multimedia processors are added to the comparison in Section 10.2.2.

10.2.1. Improvements on FPGA

For the original FPGA, different design time configurations are described and tested [34, p. 97]. This work fixed the configuration as described in Chapter 6. The exact configuration is eight pipelanes, four lane groups and four contexts. Every lane has a multiplier and, by default, all even lanes have a memory unit. The instruction cache is $8\,KiB$ per lane group, totalling to $32\,KiB$. The data cache is $2\,KiB$ per lane group, totalling to $8\,KiB$. Therefore, the grand total of cache memory is $40\,KiB$. The configuration is exactly the same as the 'sa96' and 'sa24' designs, except for the cache sizes, which is somewhere in between the two.

All differences between this work and the two stated designs is listed in Table 10.6. As is clear, the technology nodes of the two FPGAs is quite a bit smaller than the used node for ASIC. This is possible due to the mass-production of FPGA, but a big part of the advantage is lost in the generic FPGA fabric. The clock frequencies achievable on the ML605 and VC707 are around $35\,MHz$ and $61\,MHz$ respectively. The ASIC achieves a clock speed of $100\,MHz$ with ease.

A performance measurement is somewhat more difficult. The amount of cycles a program takes is largely dependent on the cache accesses. As the ASIC has an external memory connected through the SSI, a cache miss might have a large penalty. Therefore, the amount of cycles is taken as performance reference. [34] does not state separate cycle counts for the programs ucbqsort and compress, but instead only the full PowerStone benchmark. For the gr24 design, running on the ML605, separate cycle counts are given. For this program, ucbqsort takes $290/8848 \approx 3.278\%$ of the total cycles and compress takes $197/8848 \approx 2.223\%$. These percentages are used to approximate the cycles in Table 10.6.

Clearly, the larger caches of the sa96 result in fewer cache misses and, therefore, shorter cycle count. As the ρ-VEX cache size is somewhere between the two shown designs, its cycle count will also lie in between. The overhead of the HSI is, due to improvements in bandwidth and latency, quite small.

Interestingly, the amount of cycles used for ucbqsort is with 366 k half the cycle count of the FPGA ρ-VEX. Presumably, due to the small data overhead, the difference should be small. However, as the cycle counts are estimated, some difference is incurred. In contrast, compress is completed in 719 k cycles, somewhat more than the FPGA. Here the SSI overhead is visible.

10.2.2. Other multimedia processors

The ρ-VEX is a VLIW processor. This type of architecture was mostly popular for multimedia applications. Furthermore, due to its ambitions in space (**G**.5), the ASIC should target low power.

A very popular low power processor with similar functionality are processors of the ARM Cortex-M4 architecture. Although performance is difficult to compare, it also targets multimedia and low power applications. It will be the main competitor for the ρ-VEX. From the ARM website*, their targeted frequency, energy consumption and core area are retrieved.

Other compared processors are the TM3270, TMS320C64x and PIC32MM. The TM3270 is one of the latest NXP Semiconductors TriMedia processor. Like the ρ-VEX, it is a VLIW processor. This specific model is compared as it is described very detailed in [16], although the processor line was discontinued in 2009.

TMS320C64x is the VLIW processor developed by Texas Instruments. More powerful versions are available, but this specific model has very similar capabilities. Finally, the PIC32MM is a MIPS32 processor by MicroChip Technology. It is their most power efficient design. Although it is quite old, it used to be extremely popular for low power multi-media applications. As its parameters kind of disappointed, it is not further compared. The ρ-VEX ASIC performs better on every level, solely due to the better process technology.

*https://developer.arm.com/products/processors/cortex-m/cortex-m4

ARM Cortex-M4
Detailed statistics for a $65\,nm$ design are difficult to find. The energy consumption will be somewhere between $90\,nm$ and $40\,nm$, with $32.82\,\mu W\,MHz^{-1}$ and respectively $12.26\,\mu W\,MHz^{-1}$. With a predicted energy consumption of $367\,\mu W\,MHz^{-1}$, the ρ-VEX ASIC gets outperformed by about 15 times.

However, the Cortex-M4 is a different architecture. The ρ-VEX will be able to execute eight instructions in parallel and has a pipelane of four stages compared to three. A testbench will have to be run on the final IC to exactly compare the performance. However, the extra energy consumption in part comes from the increased performance.

Furthermore, the largest energy consumer in the ρ-VEX is its 24-port register file. The reduced performance of the Cortex-M4 also simplifies its register file, probably leading to the rest of the power different.

To fully quantify the performance per energy, the same program should be run on both the ρ-VEX and a Cortex-M4. The total energy consumed to complete the program is the only accurate measurement.

TM3270
The TM3270 has, with 13 stages compared to 4, a far deeper pipeline. Therefore, its clock frequency is also more than three times as high. However, the main feature of the TM3270 is not its speed, but its voltage scaling. By bringing down the core device voltage from $1.2\,V$ to $0.8\,V$, the energy consumption goes down by more than half. This is especially useful when the processor resources are not fully utilized.

Without DVS, a feature the ρ-VEX does not yet have, the energy consumption of the TM3270 is about 22% higher. With DVS, it outperforms the ρ-VEX by about 46%. Therefore, the real energy savings are made by the added power modes, a feature that can be added to the ρ-VEX as well.

TMS320C64x
This processor also gains its speed advantage due to a deeper pipelane. With 11 stages, it can outperform the ρ-VEX clock frequency by five times. Even doing so, the active energy consumption is somewhere in between the two modes of the TM3270. Note that the stated energy consumption is from the TMS datasheet, being measured from an only 50% occupied 'typical' case [73, p. 94]. Therefore, this processor also improves its power consumption by utilizing low-power modes.

As the ρ-VEX ASIC does not yet have a low-power mode, the comparison is somewhat more difficult. However, even if the TMS value is extrapolated to full load by doubling it, the ρ-VEX gets outperformed by about 35%.

Table 10.6: The ρ-VEX ASIC of this work compared to the ρ-VEX FPGA designs

Design	Platform	Technology	Instr. cache (KiB)	Data cache (KiB)	Frequency (MHz)	ucbqsort (cycles)	compress (cycles)
This work	ASIC	$65\,nm$	32	8	100	366 k	719 k
sa24	ML605	$40\,nm$	16	8	35.3	902 k	613 k
sa24	VC707	$28\,nm$	16	8	61.5	-	-
sa96	ML605	$40\,nm$	64	32	35.2	671 k	456 k
sa96	VC707	$28\,nm$	64	32	60.9	-	-

Table 10.7: This work compared to similar multimedia processors

Design	Architecture	Technology	Frequency (MHz)	Pipeline stages	Energy[§] ($\mu W\,MHz^{-1}$)	Core area (mm^2)
This work	VLIW	UMC 65LL	100	4	367	1.82[¶]
Freescale Kinetis K70	ARM Cortex-M4	TSMC 90LP	150	3	32.82	0.119
ARM Cortex-M4*	ARM Cortex-M4	GF 65LPe	80	3	< 40	-
ARM Cortex-M4*	ARM Cortex-M4	TSMC 40LP	-	3	12.26	0.028
TM3270[‡]	VLIW	$90\,nm$	350 / 8	7-13	447 / 198[†]	4
TMS320C64x	VLIW	$130\,nm$	500	11	136	-
PIC32MM	MIPS32	$250\,nm$	25	5	184	10

*Data from ARM website, no device name stated.
[‡]The TM3270 allows voltage scaling, reducing the frequency and energy consumption. This is the main operational mode of the TM3270 and, therefore, added to the comparison.
[†]Only Decode, Regfile and Execute modules are added. Instruction fetch and Load/store are left out as no clear separation of SRAM is available. The result is an optimistic power consumption.
[§]The energy is under active load, for only the core. Memories are factored out where possible.
[¶]Core area with register file and routing, caches and SSI instances are left out.

10.3. Conclusion

The signoff results of the ρ-VEX ASIC are presented. Some violations exist, but all have been manually investigated and can be safely waivered. Visual inspection shows no unintended mistakes. Furthermore, with an IR-drop less than 1% and electromigration risk of less than $4 \cdot 10^{-7}\%$, the design is expected to reliably operate for tens of years.

When looking at the ρ-VEX area, the register file and SSI instances are placed very inefficiently. Due to the difficult routing, a utilization of only 38.9% is achieved. Therefore, the register file takes more than 41% of the total chip area. A future redesign of the register file can significantly improve upon this area. The SSI instances have a design flaw, whereby they don't nicely fit the width of the chip. This causes 34% of the chip area to be wasted, not a single other cell can be placed. A redesign should take the total chip width into account, probably thinning the SSI such that three or four instances exactly fit next to each other. Improving the register file and SSI area means larger instruction and data caches can be used, which now only account for 18% of the total area.

With $100\,MHz$, the ASIC ρ-VEX outperforms the FPGA ρ-VEX by two to three times. Even though a cache miss is more expensive, as data has to be retrieved through the HSI, the final design is faster in completing a program. Compared to the TM3270 or TMS320C64, it is still not enough. These similar VLIW processors run at three and five times the frequency. The main reason for this difference are their far deeper pipelines. With 109 logic levels, the critical path in the ρ-VEX is extremely long. A future design can improve upon the speed by restructuring this pipeline.

Using simulation switching activity and a PGL, accurate power results are generated. The ρ-VEX core is able to achieve $36.68\,\mu W\,MHz^{-1}$. Compared to the TM3270 and TMS320C64, this performance is right in the middle. The ρ-VEX is able to compete with other VLIW designs under full load. However, the TM3270 can outperform the ρ-VEX if DVS is activated, reducing clock frequency for a lower power dissipation.

Compared to the Cortex-M4, the ρ-VEX has about 10 to 15 times higher power dissipation. However, the different architecture allows the ρ-VEX to executes more than eight operations in the time the Cortex-M4 executes one. The rest of the power difference comes from the ρ-VEX register file. The 24-port standard cell design probably consumes half the total energy. To get on the same energy level as the Cortex-M4, the register file has to be redesigned.

11

Conclusion

To conclude this work, a full summary of all chapters is given in Section 11.1. Subsequently, the design goals as introduced in Section 1.6 are revisited in Section 11.2. This section states for each goal which parts are and which parts aren't fully achieved. Finally, in Section 11.3, the recommendations for future research and improvements are given.

11.1. Summary

11.1.2. General background

All background generally necessary for understanding IC fabrication has been expanded upon. For this particular project, the $65\,nm$ process of UMC is used. Both analog and digital libraries are available. Because IMEC is used as IC service, their MPW projects can be accessed for cheap prototyping. Furthermore, it allows for transistors in three different flavors, low-, regular- and high-threshold voltage. To connect all devices, one poly layer and eight metal layers are available. Metal layer seven is double thickness and layer eight is eight times thickness.

In general, it is important to check for reliability problems on chip. Two methods have been described important for determining the useful lifetime in the 'bathtub' curve. Taking IR-drop into account allows for improving yield as well as reducing infant mortality. To extend the useful lifetime up to wear out, the total effect of EM will be monitored and kept within bounds, as specified in the FDK.

11.1.3. Power dissipation on chip

In this chapter, the reader was familiarized with the different energy consumers in an IC. The short-circuit energy and switching energy in combination with a switching activity make up the dynamic power. The total power dissipation furthermore includes the static power. The main goal was to introduce the different ways to reduce power dissipation, as required by (**G**.5b). Dominant components can be predicted for a circuit. Due to its constant switching, the clock tree should be considered separately.

The different threshold voltage for transistors plays a big role in reducing leakage power and result in a trade-off between performance and energy consumption. Another trade-off can be made with Dynamic Voltage Scaling. A known effective technique of reducing overall consumed energy by lowering the voltage. This comes at the cost of a lower clock speed and increased complexity.

The last discussed option for reducing power dissipation are global power down, stand-by or partial power down modes. In global power down, the total IC is turned off, causing a slower reaction time and the requirement of an external memory. Stand-by overcomes these problems by using internal power switches. However, the energy savings are also less.

The most interesting option for the ρ-VEX is partial power down. Its reconfigurability perfectly synchronizes with disabling parts of the circuitry. If the processor load can be effectively predicted, entire pipelanes and contexts can be power gated. The result is a significant reduction in power consumption under partial loads and less timing-critical applications.

As DVS, stand-by and partial power down require changes to the circuitry as well as increase the complexity of the chip design, implementing them would conflict with the main goal of a prototype, (**G**.1). Therefore, they are not implemented in this project and are left for a future design. This work

will mainly focus on reducing the power consumption under full load, when no performance trade-off should be made.

11.1.4. ASIC design flow

An overview is given of all design steps necessary for converting an RTL netlist into a verified and manufacturable layout. The flow includes synthesis for mapping a design description onto logical gates and Place & Route for mapping logical gates on a physical chip. The flow is a global overview of how the ρ-VEX was optimized, but can be applied to any other design. Two additional improvements are touched upon, one to reduce power dissipation and one to achieve tighter timing closure.

Should it be necessary to reduce the static power dissipation, then it is possible to divide the IC into different power domains. Using power switches provided in the library it is possible to locally lower the voltage or power down the whole chip. In this work, power switches are not yet implemented. The main focus for this project was laid on reducing dynamic power dissipation, whereas power switches mostly reduce leakage power.

To achieve a tighter timing closure, physically aware synthesis was introduced. This improvement can be helpful when future designs focus on clock speed. Again, this is not yet applied as the main focus laid on reduced power dissipation. To implement physically aware synthesis, the design tools as well as the software scripts need some changes.

11.1.5. Source-Synchronous Interface

The SSI is an externally developed analog IP block, allowing it to be included in the digital design flows of DC and Encounter. It is an interface running at two times or four times the clock frequency of the ρ-VEX core and sends DDR signals over LVDS signal pairs. This allows the main memory of the ρ-VEX to reside on an FPGA external to the ASIC, leaving more chip area for the actual core while still allowing for useful memory sizes.

To use the SSI IP block, a clock divider, a DDR input buffer and a DDR output buffer are constructed. Furthermore, all timing information was manually gathered in the liberate format to be used in DC and Encounter. In a future, more timing critical, version of the ρ-VEX ASIC, it is crucial to generate this lib file with the provided software. It will allow for more accurate timings and actual power estimation.

For physical information needed during placement and routing, a LEF file was provided and explained. Missing from the LEF file, however, was gate and diffusion area size, required for antenna violation detection. Therefore, it is required to separately verify antennae on the final layout and manually correct any detected violation. For automation of this process, a future version of the LEF file should include this information.

Finally, a VITAL file was written for digitally simulating the SSI. The VITAL constructs for internal path delay and propagation delay allow the simulation to be annotated using an SDF file. Therefore, the simulation can accurately detect setup and hold violations.

11.1.6. ρ-VEX configuration

The ρ-VEX design is extensively investigated before implementing it as an ASIC, as expected for (**G**.3). One of ρ-VEX's features is its design time configuration flexibility. A test run is set up and used to compare the different options. From these initial reports, a design that will finally be implemented is chosen.

Initially, the design was expected to need some pruning. The best option is reducing the number of hardware contexts from four to two, limiting the amount of threads the ρ-VEX can run in parallel. This would allow the ASIC to be implemented on the smallest and cheapest chip area, as is preferred for (**G**.1). Later on in Section 7.5 the register file will be improved and reduced in area. This allows all four contexts to be implemented.

To significantly improve the ASIC clock speed, the number of software breakpoints has been reduced from four to one. As more breakpoints are rarely used, this is not a problem. Furthermore, the trace unit has been entirely removed, significantly reducing the logic connectivity and improving the clock speed further.

Aiming for (**G**.5b), low power design, the clock speed is constrained at $100\,MHz$. This should be easily achievable during synthesis and allows some slack during P&R. The constraint is very loose, allowing for more focus on reducing power consumption.

11.1.7. Design overview

All important signals required for implementation of the SSI are discussed. The HSI pin sharing scheme is highlighted due to the increased complexity for implementation (**G**.2). Furthermore, four different debugging schemes are discussed, UART, scan, standalone mode and a program ROM. The UART interface allows full functionality without using the SSI (**G**.4). Scan chains will be used to allow full observability and controllability of the ρ-VEX, allowing the design to be debugged to correct any mistake (**G**.4b). Standalone mode and a program ROM would allow the ASIC to operate with reduced or no external input at all, however will not yet be implemented.

The different modes of operation available to the ASIC implementation of the ρ-VEX have been discussed. The operational modes are functional fast, functional slow, UART and scan mode. The different operational modes allow for the design to be useful, even if there is an undetected mistake (**G**.4). Furthermore, the netlist changes required to implement the data and instruction cache and register file have been explained.

All possible register file implementations are thoroughly discussed. A comparison is made between a banked, replicated, multi-pumped, standard cell or full custom memory. In addition, the benefits of clustering the ρ-VEX are discussed (**G**.3). Due to its low complexity and adequate performance, the register file of this prototype will be implemented using standard cells.

For future designs it is suggested to weigh the advantages of a banked or clustered design against the added complexity. A banked register file is extremely popular in superscalar processors. Furthermore, an overall different architecture by clustering the processor, might lead to more than 40% savings in both area and power consumption, or improve processor speed by almost four times.

11.1.8. ASIC implementation

To implement the RTL to GDSII flow for the ρ-VEX, a set of TCL scripts is created. Every step and optimization is automated, allowing for the results to be easily reproduced, as required for (**G**.6). This also includes verbose comments throughout the code and an extensive user guide to aid new designers in continuing the project. The scripted environment is set up such that it can be easily configured from one file and even allows other projects to be implemented.

All synthesis and P&R constraints are implemented as required for the general flow. Timing is underconstrained by up to 30%, allowing for an energy reduction of 85%. Simulation data is collected in an SAIF file and used to reduce power consumption by another 14%. The final design is low-power (**G**.5b) while still being faster than the FPGA, keeping the design useful (**G**.4).

All steps required to achieve error-free routing and meet timing requirements in P&R are discussed. These include manual optimization of cache instance placement, module constraints for the register file and HSI, module padding for congestion reduction and cell padding for hold violation repair. To improve timing, Clock Concurrent Optimization was required, due to the imbalanced pipeline. Furthermore, an elaborate routing flow consisting of four repair steps was necessary to completely route the congested design. In total, placement requires one ECO iteration, CTS two and routing four iterations.

To improve reliability, as required by (**G**.5a), vias were replaced by multi-cut vias. Furthermore, conform to the UMC design rules, standard cell filling and metal filling are applied.

Finally, some chip considerations are stated. These are important for other designers working with the project (**G**.3c). For the PCB, power-up sequencing is required to prevent unwanted signal switching. A future design might need internal sequencing if the SRAM row redundancy option is implemented. When a next iteration of the ASIC is not fabricated using the MPW service of IMEC, it is important to manually add a die seal ring. Finally, an improvement (**G**.3c) to the placement and routing of the register file might include bit slicing, a technique that improved a similar design [51].

11.1.9. Verification & Signoff

Even when the synthesis and P&R software create a design, there is no guarantee it actually works. Different generated reports are used as early check to see if any mistakes happened. This has helped in repairing combinatorial loops as well as ensure correct timing on the cross-clock boundary.

Using simulation in ModelSim, the function of the ρ-VEX was validated. All operational modes, being functional fast, functional slow, UART and scan, are tested using a generalized testbench. Some downsides to the simulation include X-pessimism, as well as incorrect VHDL libraries in the UMC FDK. After these problems were overcome, bugs in the reset state as well as the HSI have been repaired.

In a future project it is recommended to use Formal Equivalence Checking instead of post-synthesis simulation. This will help in quicker verification and debugging of the functionality.

The simulation is also used for several PowerStone benchmarks as well as a custom convolution program. Different software in different ρ-VEX configurations will be used to fully verify the ASIC functionality. All the simulations together can be used to characterize the power dissipation in different scenarios.

To make sure the final GDSII file is correct, it is verified in external programs. Cadence Tempus in combination with a PGL is used for high accuracy STA, IR-drop and EM analysis and SI analysis. The design is verified using every possible PVT corner. Calibre is used to perform DRC, LVS and ERC checks. Virtuoso is used to visually check the GDSII file, to see if no big mistakes are made.

After all errors are corrected, some DRVs still persist. These are regarding multiple stacked vias and can safely be waivered for this project. For a future project, actually going to space, it is crucial to make sure no vias are stacked anymore. It is even possible to enable extra DRC for improved DFM. If the area permits so, all DFM rules should be followed.

11.1.10. Results

The signoff results of the ρ-VEX ASIC are presented. Some violations exist, but all have been manually investigated and can be safely waivered. Visual inspection shows no unintended mistakes. Furthermore, with an IR-drop less than 1% and electromigration risk of less than $4 \cdot 10^{-7}\%$, the design is expected to reliably operate for tens of years.

When looking at the ρ-VEX area, the register file and SSI instances are placed very inefficiently. Due to the difficult routing, a utilization of only 38.9% is achieved. Therefore, the register file takes more than 41% of the total chip area. A future redesign of the register file can significantly improve upon this area. The SSI instances have a design flaw, whereby they don't nicely fit the width of the chip. This causes 34% of the chip area to be wasted, not a single other cell can be placed. A redesign should take the total chip width into account, probably thinning the SSI such that three or four instances exactly fit next to each other. Improving the register file and SSI area means larger instruction and data caches can be used, which now only account for 18% of the total area.

With $100\,MHz$, the ASIC ρ-VEX outperforms the FPGA ρ-VEX by two to three times. Even though a cache miss is more expensive, as data has to be retrieved through the HSI, the final design is faster in completing a program. Compared to the TM3270 or TMS320C64, it is still not enough. These similar VLIW processors run at three and five times the frequency. The main reason for this difference are their far deeper pipelines. With 109 logic levels, the critical path in the ρ-VEX is extremely long. A future design can improve upon the speed by restructuring this pipeline.

Using simulation switching activity and a PGL, accurate power results are generated. The ρ-VEX core is able to achieve $36.68\,\mu W\,MHz^{-1}$. Compared to the TM3270 and TMS320C64, this performance is right in the middle. The ρ-VEX is able to compete with other VLIW designs under full load. However, the TM3270 can outperform the ρ-VEX if DVS is activated, reducing clock frequency for a lower power dissipation.

Compared to the Cortex-M4, the ρ-VEX has about 10 to 15 times higher power dissipation. However, the different architecture allows the ρ-VEX to executes more than eight operations in the time the Cortex-M4 executes one. The rest of the power difference comes from the ρ-VEX register file. The 24-port standard cell design probably consumes half the total energy. To get on the same energy level as the Cortex-M4, the register file has to be redesigned.

11.2. Main contributions

This project started off with the following research question.

"How to design and implement a reliable, power-optimized Application Specific Integrated Circuit prototype of the ρ-VEX core with external memory?"

In this document, the ASIC prototype is described. The full flow of implementing a generic design is explained and applied to the ρ-VEX. In collaboration with another student, a High Speed Interface is developed and implemented, allowing communication to an FPGA. The FPGA will handle storage of larger data. During the design, reliability is taken into account from the start. IR-drop and EM are kept to a minimum, while using as many multi-cut vias. To optimize power, different threshold voltage transistors are used. The clock frequency is loosely constrained, resulting in significant power savings.

Furthermore, a set of design goals were extracted from this research question. Each of the goals is revisited below.

(G.1), construct a first iteration ASIC prototype of the ρ-VEX.
Using a general RTL-to-GDSII flow, the ρ-VEX is be implemented as an ASIC. The cache memories are replaced using the Faraday memaker. The register file, due to its 24 ports, required special attention. Initially, the FPGA design was naively ported. The duplicated memory caused a large constrain on the chip area. The final implementation is constructed using standard logic cells. A bit is stored using a flip-flop, while allowing read and write access using multiplexers. For each port, a separate multiplexer can be added, allowing the design to be easily scaled conform the design time reconfigurability of the ρ-VEX. As it is only a first iteration prototype, many improvements on the design have not yet been implemented.
(G.2), design and implement an interface for off-chip memory.
Parallel to this project, a Source Synchronous Interface has been developed. This interface allows for Double Data Rate communication at $400\,MHz$, creating a high-bandwidth $4\,Gbit\,s^{-1}$ connection. To integrate the SSI on the ASIC, physical as well as timing information is needed, however this was not provided. Therefore, the required lib file is written entirely by hand. Furthermore, the lack of antenna information in the LEF file has been bypassed and manually checked for design errors. To verify the SSI, a VITAL simulation was constructed. As the SSI runs at four times the clock frequency of the ρ-VEX core, low-latency cross-clock boundaries are implemented. Furthermore, custom DDR input and output buffers are necessary for data conversion. The final implementation is able to communicate to external memory through LVDS signalling, either at $400\,MHz$ or $200\,MHz$. Additionally, slower communication over UART is possible if necessary.
(G.3), explore the design space as much as possible.
Part of the design space consists of the design time configuration flexibility of the ρ-VEX. As only a single instance can be implemented on an ASIC, a configuration had to be decided upon. This was done in Chapter 6. The final implementation includes the full eight pipelanes, four lanegroups and four contexts. Due to their limited usefulness, only a single breakpoint is implemented and the trace unit is left out. This allows a combined increase in clock speed of 52%. With $40\,KiB$, the total combined cache size is somewhat meagre, but should be just enough. Another big part of design space exploration was the implementation of the register file. The initial duplicated memory implementation did not fit the ASIC area. A standard logic cell implementation was good enough for this prototype. However, the register file should definitely be improved in a future respin. Discussed options include a banked, multi-pumped, full-custom or clustered design. The options are verbosely discussed and compared in Section 7.5. Overall improvements on the register file design are also discussed. By clustering the ρ-VEX, more than 40% savings in area could be achieved. On top of that, power dissipation could be reduced by 40% as well. If required, a different architecture could improve the speed by up to four times, necessary for achieving higher clock frequencies.
(G.4), ensure a useful final design.
To ensure a valid design, extensive verification tests were performed in Chapter 9. Every possible design mistake has been debugged using ModelSim functional simulations. With DRC, LVS and ERC, the P&R steps are verified and should contain no more problems. If, in the unfortunate case, a mistake did happen to slip through, different fallback methods are added, to ensure the ASIC is still useful. Using the UART, any mistake in the HSI can be bypassed. Performance is lower, but at least the FPGA is still useful. If the ρ-VEX still doesn't operate, scan chains can be used to check what went wrong. Even if no further communication or even parts of the ρ-VEX core itself don't work, the scan mode can be used to fully observe and control every flip-flop in the design. The registers can be manually read and written, while small chunks of the design or even individual paths can be debugged. Not every desired extra DFT has been implemented. Ideally, the ASIC would be able to operate standalone, allowing radiation tests to be performed without any other interference. A larger SRAM could be used as main memory, or a ROM would allow a single program to be executed by default. As the current ASIC area is very limited, these options were not possible.

(G.5), design with future space applications in mind.
For a space application, the FPGA should be able to operate for years. By proper design for power, both IR-drop and EM effects are kept to a minimum. The ASIC is functionally designed for a maximum IR-drop of 10%, but the power routing limits it to less than 1%. This means infant mortality is reduced and wear-out lengthened. The EM risk is, using accurate SPICE simulations and a Power Grid Library, estimated to be less than $3.57 \cdot 10^{-7}\%$, theoretically allowing an on-time of 90 years. Of course, external design components will probably limit this. As large batteries are undesirable in space, power dissipation should also be kept to a minimum. By properly constraining the design, power reduction of 85% was achieved. Furthermore, by using switching activity information, another 14% could be saved. Compared to other VLIW designs, the ρ-VEX performs fairly well under full load. The big problem lies in partial load or even no load at all. As the current design is only able to be fully operational, it gets outpaced by the TM3270 and TMS320C64x in most scenarios. Furthermore, compared to a low power Cortex-M4, the ρ-VEX still consumes 10 to 15 times more energy.
(G.6), every design step must be well documented and reproducible for future improvements.
To prevent having to redo all this work or regaining all knowledge for a future student, the design is set up to be fully automatic. With many TCL scripts, as well as a vast set of compound aliases, the full RTL-to-GDSII flow can be performed first-try valid. The full flow is extensively tested and will yield the same error-free results as presented in this project, using just a few simple commands. Furthermore, all DRC, LVS and ERC rulefiles have been configured. Errors have been removed from the rulefiles, as well as from several FDK components. Simulation has been configured to mostly deal with the shortcomings of ModelSim and the FDK. With the updated files, a new designer does not have to worry about any misconfiguration and can quickly prototype and fully verify any design changes. In the final results, a relation has been described between the initial synthesis run and final post P&R results. This relation allows frequency, area and power numbers to be estimated with just a quick synthesis run, saving two or even more days of additional steps. The final flow allows for quick prototyping and will significantly help in improving the future design. Finally, Chapters 2 to 4 in combination with Appendix D form an easy to follow verbose user guide for any future student continuing with the project. Even without the user guide, extensive comments throughout the scripts and README files included in the project should make the whole flow self-explanatory. Every design choice can be checked and common pitfalls prevented as long as all provided resources are utilized.

11.3. Future work

As this is only a first-iteration prototype, quite a few implementation improvements can be made. The four most prominent discussed improvements are the register file, HSI, power dissipation and debug capabilities. Furthermore, the ρ-VEX can be improved by using a deeper pipeline.

11.3.1. Register file

The register file was implemented using standard logic cells. Due to its simplicity, no architectural changes are required for the ρ-VEX, while the design is still a big improvement over the initial duplicated register file. However, research showed even better designs are possible.

In a future design it is at least recommended to implement clustering. With a standard logic cell implementation, area and power can be reduced by 40%, or performance increased by almost 4 times [51]. Further research should be performed to get the actual benefits for the ρ-VEX. To achieve these improvements, a redesign of the ρ-VEX is required, as it does not yet support clustering. The VEX, after which the ρ-VEX was designed, and the accompanying compiler suite do already support clustering.

In superscalar processors, a banked register file is extremely popular. Such a design is expected to significantly reduce the area, as optimized SRAM cells can be used for storage, while not needing the duplication. Furthermore, this improvement will directly lead to a power reduction. As a banked register file can easily be pipelined, significant performance improvements can additionally be made. However, to implement an efficient banked register file quite an elaborate design is required. On top of that, changes to the compiler are required for reducing the amount of bank conflicts. This does fit the VLIW architecture extremely well, as it is fully designed to hide the processor complexity and instead improve the compiler.

11.3.2. HSI and SSI

In its current design, the SSI wastes about 34% of its area in nothing. The width of the SSI was improperly designed, such that two abutted instances leaves a lot of space, while no third instances can fit. Furthermore, only after the project was complete, the bond pad pitch requirement of $90\ \mu m$ was lifted. This means all pins can be a lot closer together, allowing three or even four SSI instances on one side, significantly improving area utilization.

As the parallel project did not provide the required lib files and only partial information in the LEF file, required timing and antenna information were substituted for hand-collected data. In a future design, the HSI can have a lot stricter timing if the correct information is actually generated using the Liberate AMS characterization software.

11.3.3. Power dissipation

Due to its reconfigurability, the ρ-VEX can change a program from eight pipelanes to two pipelanes on the fly. If the processor load or ILP is predicted to be low, this can be done without a large penalty. By allowing the remaining six (or four, depending on configuration) remaining pipelanes to be fully powered down, dynamic power consumption could be instantly cut by almost 75%. On top of that, unused context registers or empty caches could have the same functionality, for a significant reduction in static power consumption.

Besides this partial power down, full power down or stand-by have also been discussed. These modes will be crucial for a space mission. This document focused mostly on reducing full load power dissipation, as that is the main use case for a prototype. However, when implemented in a satellite, the ρ-VEX will be powered on for days while only travelling. In this time no significant calculations are performed and, therefore, precious energy can be saved by utilizing low power modes.

Other processors already effectively show the benefits of these low power modes for multimedia and IoT applications as well. As these applications are also a targeted market, implementing low power modes is even more important for the ρ-VEX. On top of that, the TM3270 is able to reduce the total energy consumption of a program by half by utilizing Dynamic Voltage Scaling. All are interesting research opportunities for use-cases outside the scope of this document.

11.3.4. Debug capabilities

In its current state, debugging is fully enabled through the scan chains. However, other useful discussed debugging capabilities are a standalone mode, program ROM or extending the scan chain functionality by using it for configuration registers. For example, memory redundancy can be configured, such that defect SRAM bit cells can be swapped out. Additionally, BIST can help in detecting these defect cells.

The already constrained area of this prototype limited these implementations and it was decided to not implement them. However, if a future design includes a better and smaller register file, these extra functions become viable.

11.3.5. Deeper pipeline

The ρ-VEX is lacking performance compared to other VLIW processors. With a clock frequency of $100\ MHz$, it gets outperformed by four to five times. The clock is mostly limited due to extremely many logic levels, counting up to 109. Deepening the pipeline, by going from four to over ten stages, can bring the ρ-VEX to the same performance as commercial VLIW processors. This improvement can be made in both the ASIC of this work as well as the original FPGA design.

Glossary

ASIC An Application-Specific Integrated Circuit is an IC specifically designed for a single application. i, vii, 3–5, 7, 19, 35–38, 40, 41, 43–45, 48–51, 54, 55, 64–66, 68, 81, 82, 85, 86, 88, 90, 92, 93, 97, 100, 101, 103, 107–110, 112–117, 137

BEOL The Back End Of Line consists of all process steps in IC fabrication concerning the connections between transistors. See also FEOL. 8

BIST The Built-In Self Test is (a part of) an IC that is able to test itself, without external intervention necessary. 54, 91, 117

BOAC Bonding Over Active Area allows a design to bonds wires connecting package pins with IC pads on top of transistors. This is not always possible, as the bonding machine might damage the underlying circuit. 23

BRAM A Block Random Access Memory is a piece of memory with standardised size and address length, used short-term storage in FPGAs. 64

CCOpt Clock Concurrent Optimisation is a licensed CTS product developed by Azuro, useful for simultaneously optimising the clock tree and data paths. 27, 28, 76, 77, 103

CISC The Complex Instruction Set Computing is an ISA designed to use long compound instructions, allowing multiple operations to be performed in one call. 1, 2

clean room An environment within the foundry that is highly controlled, only a limited amount of particles per cubic metre are allowed, required for fabrication of an IC. 7

CTS Clock Tree Synthesis is an IC design step after standard cell placement and before signal routing, where the clock net is created. 27, 32, 38, 39, 67, 73, 75–78, 82, 84, 113

DC Design Compiler is Synopsys' software solution for RTL to gate level netlist synthesis. 19, 38–41, 44, 47, 48, 65, 83, 84, 90, 92, 100, 103, 112

DDR Double Data Rate is a communication protocol where data is sent on both the rising and falling edge of the clock signal, achieving twice the data rate. 36, 37, 41, 112, 115

DFM Design For Manufacturing is a design methodology to ensure the correct device is manufactured. Prevents components that a foundry cannot handle. 83, 95, 100, 114

DFR Design For Reliability is a design methodology where special focus is laid on improving the reliability. 83, 92

DFT Design for Test is a design methodology where specialised hardware is added to aid in testing the manufactured IC. 22, 83, 91, 115

DFX Design For x is the common name for different design methodologies, many values for x exist. In this project, DFR, DFT and DFM are performed. 83

DRC The Design Rule Check is an automated check for the TLR. i, 8, 31, 72, 94, 96, 99–101, 114–116

DRV A Design Rule Violation is a violation of a DRC. 31, 79, 95, 96, 100, 114

DSP Digital Signal Processor, a processor with hardware dedicated to a very specific task. 1

DVS Dynamic Voltage Scaling is a power dissipation reduction technique. When the load permits, the voltage is dynamically lowered at the cost of an increased transition time. 16, 17, 108, 110, 111, 114

ECO Engineering Change Order allows the default order of different IC design steps to be reversed, to make small changes without redoing the whole flow. 32, 75, 77, 78, 82, 113

EDR The Electrical Design Rules list the electrical characteristics of all wiring, transistors, resistors etc. for a particular IC process. 9

EM Electromigration is the effect of metal ions being moved due to an electron flow. It is a main contributor to reliability issues. 10–12, 24, 31, 72, 73, 92, 94, 99, 100, 111, 114, 116

ERC The Electrical Rule Check is an automated software check for determining correctness of all VDD and VSS connections. i, 31, 99, 100, 114–116

ESD An Electrostatic Discharge is a sudden burst of current caused by connection with a charged object. ESD can cause damage to unprotected circuitry. 23, 72

fanout The amount of logic gate inputs connected to a logic gate output. 21

FDK The Foundry Design Kit consists of the whole set of libraries, verification rules and manuals provided by a foundry for a specific process. 7–9, 12, 15, 26, 50, 84, 85, 92, 94, 99, 100, 111, 113, 116

feature size The smallest transistor gate length that can be fabricated for a particular process. 7, 8

FEC Formal Equivalence Checking allows the user to prove the IC design before and after synthesis are equivalent. 91

FEOL The Front End Of Line consists of all process steps in IC fabrication concerning the manufacturing of transistors. See also BEOL. 8

flip-chip Technique where an IC is connected directly to external circuitry through solder bumps. The top is flipped over to create a connection. 23

foundry The site responsible for manufacturing an IC. 7–9, 11, 12, 19, 22, 92

FPGA A Field Programmable Gate Array is a piece of generic hardware fabric, allowing programming of different hardware designs. Commonly used for prototyping before an ASIC. vii, 3, 19, 37, 41, 43, 49, 51, 55, 57, 64, 66, 67, 71, 82, 83, 85, 101, 103, 107, 109, 110, 112–117, 137

fringe capacitance The usually unwanted capacitance in a metal wire due to the non-zero thickness. 9

FSM A Finite State Machine is an abstract model for describing the different states and flows in a digital circuit. 49, 50

GDSII A common binary file format for describing an IC layout. v, 19, 20, 30, 31, 65, 82, 83, 94, 96, 97, 100, 113–116, 149

golden file A reference file containing the correct solution to a certain simulation. 30, 85

ground bounce The effect where the ground net locally has a higher voltage than the global VSS net. 11

halo The area around a macro, can be used to block placement or routing of other instances. 24, 95

HDL A Hardware Description Language is a language for describing hardware components, the most widely used languages are VHDL and verilog. 5, 21, 30, 83, 92

HSI The High Speed Interface is the communication between the this design and external components, running at a higher frequency. 3, 49–52, 64, 67, 71, 82, 85, 86, 89, 91, 92, 100, 101, 103, 107, 110, 113–117, 122

HVT High Voltage Threshold refers to transistors with a higher than normal threshold voltage, useful for reducing leakage. 9, 26, 92

IC An Integrated Circuit is an electronic circuit consisting of transistors, fabricated on a semiconductor material. i, 1, 4, 5, 7–17, 19, 21–23, 25, 27, 29–31, 33, 35, 39, 50, 54, 81, 99, 100, 104, 108, 111, 112, 120, 121, 145

ILP Instruction Level Parallelism is when a program contains multiple instructions without inter-dependencies, allowing them to be executed in parallel. i, 2, 99, 117

IMEC The MPW IC service provider for IMEC. i, 4, 9, 12, 21, 47, 66, 71, 81–83, 94, 95, 101, 111, 113

IP Intellectual Property is a design owned and sold by another company. In IC design, usually exported as a black box macro to hide the internal design. 24, 30, 35, 36, 38, 41, 112

IR-drop A drop in voltage caused by a current flowing through a wire with non-zero resistance. i, 11, 12, 24–26, 31, 32, 73, 92–94, 99, 100, 110, 111, 114, 116

ISA An Instruction Set Architecture is an abstracted model of a specific type of computer. 55, 63

LED A Light Emitting Diode is a simple electronic component for emitting light. 51

LEF The Library Exchange Format is a file format for storing library information. 38, 39, 41, 112, 117

LVDS Low-Voltage Differential Signalling is a current driven communication protocol where small voltages are enough for noise correction. Frequently used for high speed data. i, 36, 37, 40, 41, 67, 77, 89, 112, 115

LVS Layout versus Schematic is a comparison of a GDSII layout file with the intended design described in a SPICE netlist. i, 31, 96, 97, 99–101, 114–116

LVT Low Voltage Threshold refers to transistors with a lower than normal threshold voltage, useful for reducing transition time. 9, 26, 27, 92

macro A description of a small IC that can be readily used within larger designs. Used primarily for IP. 23

mask A photomask for exposing very specific parts of a wafer to light. 9, 10

metal layer Metal wires separated by a dielectric isolation that can be used to connect transistors on an IC. 8, 9

metal stack The amount, material and thickness of metal layers that a foundry can manufacture for a specific process. 8, 9

MPW For a Multi-Project Wafer, a single silicon wafer is produced with multiple IC designs, to reduce the cost of the individual designs. 4, 9, 10, 12, 21, 47, 66, 81, 82, 111, 113

NDR A Non-Default Rule is a set of rules to augment the TLR for specific components in an IC design. Used in Encounter for constraining individual wires. 77, 78

OPC Optical Proximity Correction is a correction technique used in lithography, it corrects for the finite resolution of light waves used for patterning. 28

PCB A Printed Circuit Board consist of circuitry used to mount and connect multiple ICs and other components. 25, 65, 81, 82, 92, 97, 113

PGL The Power Grid Library describes all parasitics inherent to the devices in an IC design, as extracted from SPICE netlists. The PGL is used for accurate power calculations. 92, 100, 103, 110, 114

PowerPC A RISC superscalar processor architecture developed by IBM, Apple and Motorola. 1

PVT Process, Voltage and Temperature are usually uncertain while designing an IC, a PVT corner is a combination of three possible values. 22, 31, 76, 92, 99, 100, 114

re-spin Improving a design with the gained knowledge. 3, 5, 7, 115

RISC The Reduced Instruction Set Computing is an ISA designed to use simple instructions, to reduce the complexity of a processor. 1, 2

ROM Read Only Memory is a memory with pre-determined values. The values can only be read. 51, 52, 64, 113, 115

RTL Register Transfer Level is an abstraction used in HDLs, where a design is described as data flows and registers. v, 19–23, 30, 31, 33, 36, 65, 82, 83, 85, 86, 91, 112, 113, 115, 116, 149

RVT Regular Voltage Threshold refers to the standard transistors in a library. 9, 26, 38, 92

SAIF Switching Activity Interchange Format is a file format for exporting and importing logic gate switching activity in software tools. 23, 80, 82, 113

SDC The Synopsys Design Constraint is a file format to exchange IC design constraints, most common supported format for all major software. 67

SDF Standard Delay Format is a standardised file format for storing IC design timing information. 22, 30, 40, 41, 83, 85, 88–90, 112

SEM The Scanning Electron Microscope is a microscope using a scanning electron beam that can be used for imaging of very small details. 11

sheet resistance The resistance of a square piece of wire, useful for calculations in an IC where the thickness of a wire is fixed. 9

SI Signal Integrity is a quantification of signal distortion in a wire, caused by the switching activity of nearby wires. 28, 31, 100, 114

slack The margin on a timing constraint, a negative slack results in violation of the constraint. 21, 22, 27, 28

SNR The Signal to Noise Ratio is a quantification of the amount of noise. Small SNR values result in loss of data. 61

SPARC The Scalable Processor Architecture is a RISC processor architecture developed by Sun Microsystems, targeting workstations and servers. 1

SPICE SPICE is the most common way of simulation transistor-level circuits. 92, 96, 101, 116, 155

SRAM Static Random Access Memory is fast and dense memory used for in circuit storage. 24, 30, 35, 45, 51, 52, 54–59, 61, 64, 81, 82, 88, 91, 92, 95–97, 101–103, 109, 113, 115–117

SSI The Source Synchronous Interface is a macro designed specifically for use in the HSI for this work. 35–41, 47, 49, 50, 52, 64, 66, 68, 71, 72, 77, 85, 86, 92, 93, 96, 97, 100–103, 107, 109, 110, 112–115, 117, 137

STA Static Timing Analysis is a quick setup and hold timing computation bypassing the need for full circuit simulation. 99, 100, 114

superscalar processor A processor with duplicated hardware, allowing it to execute multiple instructions simultaneously on the fly. 2, 116

TCL An easy-to use programming language, commonly embedded in large software tools. Pronounced tickle. 65, 82, 92, 94, 96, 113, 116, 149

TLP Exploiting Thread-Level Parallelism means dividing a program or task in separate threads, allowing them to be executed in parallel. i, 1, 2

TLR The Topological Layout Rules is a set of rules created by a foundry that are required when drawing transistors and metal wires. The rules are required for correct fabrication of an IC. 7, 8, 11, 12, 29, 31, 80, 85, 94, 95

TU The Total Utilisation is the percentage of design area that is covered by transistors. 24, 71, 73

UART Universal Asynchronous Receiver and Transmitter is a hardware communication protocol for asynchronous serial data transmission and reception. 50–52, 64, 67, 85, 100, 113, 115

UMC United Microelectronics Corporation is the IC manufacturer which process is used in this project. 8, 9, 12, 13, 21, 50, 82, 84, 92, 94, 100, 111, 113

VDD The common power pin name used in IC design. See also VSS. 11, 13–16, 24, 31, 97

verilog Verilog is an HDL, mostly popular within the USA. See also VHDL. 19

VEX The VLIW Example is an open-source combination of compiler, simulator and basic processor, targeting the VLIW architecture. i, vi, vii, ix, x, xiii, 2–5, 16, 17, 19, 21, 28, 33, 35–38, 41, 43–52, 54–57, 61–66, 68, 81–86, 88–90, 92–94, 99–103, 107–117, 129, 137, 155

VHDL In full, VHDL stands for Very High Speed Integrated Circuit Hardware Description Language. It is an HDL, mostly popular outside the USA. See also verilog. 3, 19, 40, 43, 65, 86, 100, 113

via An electrical connection between two metal layers, or a metal layer and transistor. 9

VITAL The VHDL Initiative Towards ASIC Libraries is an IEEE standard for adding timing information required in ASIC design to VHDL simulations. 40, 41, 112, 115

VLIW Very Long Instruction Word is an ISA with multiple instructions per word, allowing the processor to exploit ILP. i, 2, 16, 56, 61, 103, 107, 110, 114, 116, 117

VSS The common ground pin name used in IC design. See also VDD. 11, 13, 14, 24, 31, 72, 97

wafer The basis for IC manufacturing. It is a thin piece of semiconductor material, on which transistors can be patterned. 9, 10, 21

WNS The Worst Negative Slack is the smallest slack in a design. 21, 26, 28, 44, 77–79

yield The amount of dies that function directly after fabrication. 7, 10, 12, 29, 31, 78, 111

Bibliography

[1] V. Petrovic *et al.*, “Design methodology for fault tolerant ASICs”, in *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, Apr. 2012, pp. 8–11. DOI: 10.1109/DDECS.2012.6219014.

[2] *Synchronous High-sensity Single-port SRAM Compiler with Row Redundancy Option Specification*, 1.2, Faraday, Jul. 2010.

[3] J. R. Marshall and J. Robertson, “An Embedded Microcontroller for Spacecraft Applications”, in *2006 IEEE Aerospace Conference*, 2006. DOI: 10.1109/AERO.2006.1655953.

[4] S. Zhang and H. Liu, “Synthetical Analysis on Space Radiation Tolerance Techniques in ASICs and FPGAs”, in *2011 International Conference on System science, Engineering design and Manufacturing informatization*, vol. 2, Oct. 2011, pp. 305–310. DOI: 10.1109/ICSSEM.2011.6081305.

[5] (2006). Intel 4004 Anniversary Project, [Online]. Available: http://www.4004.com (visited on 07/12/2017).

[6] *65 nm Logic and Mixed-Mode Low Leakage Low-K Process Electrical Design Rule*, 1.5_P.1, UMC, Jul. 2014.

[7] S. Hamdioui, *ET4076-11: VLSI Test Technology & Reliability*.

[8] A. Bossche, *ET4277: Microelectronics Reliability - Reliability Distributions*.

[9] A. Bossche, *ET4277: Microelectronics Reliability - Failure Mechanisms*.

[10] K. Neubeck, *Practical Reliability Analysis*. New Jersey, USA: Pearson Prentice Hall, 2004.

[11] J. R. Black, “Electromigration - A brief survey and some recent results”, *IEEE Transactions on Electron Devices*, vol. 16, no. 4, pp. 338–347, Apr. 1969, ISSN: 0018-9383. DOI: 10.1109/T-ED.1969.16754.

[12] A. Bossche, *ET4277: Microelectronics Reliability - interconnect related failures*.

[13] J. Grose *et al.*, “Optical Studies of Single Molecule Transistors”, Jul. 2007.

[14] *65 nm Logic and Mixed-Mode Low Leakage Low-K Process Topological Layout Rule*, 1.19, UMC, Jun. 2015.

[15] N. Banerjee *et al.*, “Novel low-overhead operand isolation techniques for low-power datapath synthesis”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 9, pp. 1034–1039, Sep. 2006, ISSN: 1063-8210. DOI: 10.1109/TVLSI.2006.884054.

[16] J. W. van de Waerdt *et al.*, “The tm3270 media-processor”, in *38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO’05)*, Nov. 2005, p. 342. DOI: 10.1109/MICRO.2005.35.

[17] J. Marshall, *RTL Coding and Optimization Guide for use with Design Compiler*, Tera Systems Inc.

[18] SynTunSys, [Online]. Available: http://researcher.ibm.com/researcher/view%5C_group.php?id=7020 (visited on 07/19/2017).

[19] M. Anwar *et al.*, “Early Scenario Pruning for Efficient Design Space Exploration in Physical Synthesis”, in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, Jan. 2016, pp. 116–121. DOI: 10.1109/VLSID.2016.94.

[20] *UM065GIOLL25MVIR_B UMC 65nm Low-K 1.8V/2.5V/3.3V Low Leakage RVT BOAC In-line IO Library Application Note*, B04, UMC, Sep. 2013.

[21] *EDI System User Guide*, 14.12, Cadence, Jun. 2014.

[22] M. Coenen and D. de Greef, “Optimal Techniques for Minimizing IR-Drop and Supply Bounce”, Dec. 2008.

[23] Z. Di *et al.*, “Lc-ko: A congestion-aware and area timing-oriented placement method”, in *2015 IEEE 11th International Conference on ASIC (ASICON)*, Nov. 2015, pp. 1–4. DOI: 10.1109/ASICON.2015.7516959.

[24] H.-Y. Chen and Y.-W. Chang, “Chapter 12: Global and detailed routing”, in *Electronic Design Automation: Synthesis, Verification, and Test (Systems on Silicon)*, Mar. 2009, pp. 687–750.

[25] B. Y. Su *et al.*, “An exact jumper-insertion algorithm for antenna violation avoidance/fixing considering routing obstacles”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 4, pp. 719–733, Apr. 2007, ISSN: 0278-0070. DOI: 10.1109/TCAD.2007.892338.

[26] T. W. Tseng *et al.*, “A power delivery network (pdn) engineering change order (eco) approach for repairing ir-drop failures after the routing stage”, in *Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test*, Apr. 2014, pp. 1–4. DOI: 10.1109/VLSI-DAT.2014.6834874.

[27] H.-T. Chen *et al.*, “New spare cell design for ir drop minimization in engineering change order”, in *2009 46th ACM/IEEE Design Automation Conference*, Jul. 2009, pp. 402–407.

[28] G. Kudva. (Mar. 2015). Using Physically Aware Synthesis Techniques to Speed Design Closure of Advanced-Node SoCs, Cadence, [Online]. Available: http://chipdesignmag.com/sld/blog/tag/physical-synthesis/ (visited on 07/19/2017).

[29] Jayshree *et al.*, “A methodology for designing lvds interface system”, in *2016 Sixth International Symposium on Embedded Computing and System Design (ISED)*, Dec. 2016, pp. 284–288. DOI: 10.1109/ISED.2016.7977098.

[30] *Virtex-6 FPGA SelectIO Resources User Guide*, 1.6, Xilinx, Nov. 2014.

[31] *Liberty User Guides and Reference Manual Suite*, 2016.12, Synopsys, Dec. 2016.

[32] *LEF/DEF Language Reference*, 5.7, Cadence, Nov. 2009.

[33] S. J. Krolikoski, “Standardizing asic libraries in vhdl using vital: A tutorial”, in *Proceedings of the IEEE 1995 Custom Integrated Circuits Conference*, Mar. 1995, pp. 603–610. DOI: 10.1109/CICC.1995.518256.

[34] J. van Straten, “A Dynamically Reconfigurable VLIW Processor and Cache Design with Precise Trap and Debug Support”, 2016.

[35] *65 nm Synchronous High-Density Single-Port SRAM Compiler with Row Redundancy Option Specification*, Faraday, Jul. 2008.

[36] *65 nm Synchronous High-Density Dual-Port SRAM Compiler with One-Pair Row Redundancy and BIST Test Interface Option Specification*, Faraday, Jul. 2008.

[37] *65 nm Synchronous Single-Port Register File Compiler Specification*, Faraday, Apr. 2008.

[38] J. Wawrzynek *et al.*, *CS250 VLSI Systems Design Lecture 8: Memory*, 2010.

[39] G. Yijun and W. Zuo, “Mapping n-port memory with dual-port array”, in *2009 WRI World Congress on Computer Science and Information Engineering*, vol. 3, Mar. 2009, pp. 444–447. DOI: 10.1109/CSIE.2009.889.

[40] J. H. Tseng and K. Asanovic, “Banked multiported register files for high-frequency superscalar microprocessors”, in *30th Annual International Symposium on Computer Architecture, 2003. Proceedings.*, Jun. 2003, pp. 62–71. DOI: 10.1109/ISCA.2003.1206989.

[41] S. Wallace and N. Bagherzadeh, “A scalable register file architecture for dynamically scheduled processors”, in *Proceedings of the 1996 Conference on Parallel Architectures and Compilation Technique*, Oct. 1996, pp. 179–184. DOI: 10.1109/PACT.1996.552666.

[42] N. Manjikian, “Design issues for prototype implementation of a pipelined superscalar processor in programmable logic”, in *2003 IEEE Pacific Rim Conference on Communications Computers and Signal Processing (PACRIM 2003) (Cat. No.03CH37490)*, vol. 1, Aug. 2003, 155–158 vol.1. DOI: 10.1109/PACRIM.2003.1235741.

[43] B. C. C. Lai and J. L. Lin, “Efficient designs of multiported memory on fpga”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 1, pp. 139–150, Jan. 2017, ISSN: 1063-8210. DOI: 10.1109/TVLSI.2016.2568579.

[44] A. K. Jones *et al.*, “An FPGA-based VLIW Processor with Custom Hardware Execution”, Feb. 2005.

[45] K. Johguchi *et al.*, “A 0.6-tbps, 16-port sram design with 2-stage- pipeline and multi-stage-sensing scheme”, in *ESSCIRC 2007 - 33rd European Solid-State Circuits Conference*, Sep. 2007, pp. 320–323. DOI: 10.1109/ESSCIRC.2007.4430308.

[46] C. E. LaForest *et al.*, “Multi-Ported Memories for FPGAs via XOR”, Feb. 2012.

[47] A. A. Muddebihal and C. Purdy, “Design and implementation of area efficient multi-ported memories with write conflict resolution”, in *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug. 2015, pp. 1–4. DOI: 10.1109/MWSCAS.2015.7282041.

[48] A. M. S. Abdelhadi and G. G. F. Lemieux, “A multi-ported memory compiler utilizing true dual-port brams”, in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2016, pp. 140–147. DOI: 10.1109/FCCM.2016.45.

[49] A. Canis *et al.*, “Multi-pumping for resource reduction in fpga high-level synthesis”, in *2013 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2013, pp. 194–197. DOI: 10.7873/DATE.2013.053.

[50] H. E. Yantir *et al.*, “Efficient implementations of multi-pumped multi-port register files in fpgas”, in *2013 Euromicro Conference on Digital System Design*, Sep. 2013, pp. 185–192. DOI: 10.1109/DSD.2013.28.

[51] A. S. Terechko, “Clustered VLIW architectures: a quantitative approach”, Technische Universiteit Eindhoven, Eindhoven, 2007. DOI: 10.6100/IR617141.

[52] J. J. Wu *et al.*, “A 45-nm dual-port sram utilizing write-assist cells against simultaneous access disturbances”, *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 59, no. 11, pp. 790–794, Nov. 2012, ISSN: 1549-7747. DOI: 10.1109/TCSII.2012.2228398.

[53] S. Nakata *et al.*, “Increasing static noise margin of single-bit-line sram by lowering bit-line voltage during reading”, in *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug. 2011, pp. 1–4. DOI: 10.1109/MWSCAS.2011.6026600.

[54] E. Seevinck *et al.*, “Static-noise margin analysis of mos sram cells”, *IEEE Journal of Solid-State Circuits*, vol. 22, no. 5, pp. 748–754, Oct. 1987, ISSN: 0018-9200. DOI: 10.1109/JSSC.1987.1052809.

[55] B. Daya *et al.*, “Synchronous 16x8 SRAM Design”, University of Florida.

[56] R. Kaur *et al.*, “A 6t sram cell based pipelined 2r/1w memory design using 28nm utbb-fdsoi”, in *2015 28th IEEE International System-on-Chip Conference (SOCC)*, Sep. 2015, pp. 310–315. DOI: 10.1109/SOCC.2015.7406973.

[57] D.-P. Wang *et al.*, “A Two-Write and Two-Read Multi-Port SRAM with Shared Write Bit-Line Scheme and Selective Read Path for Low Power Operation”, National Chiao-Tung University, 2012.

[58] S. Li *et al.*, “Design of a high-speed low-power multiport register file”, in *2009 Asia Pacific Conference on Postgraduate Research in Microelectronics Electronics (PrimeAsia)*, Jan. 2009, pp. 408–411. DOI: 10.1109/PRIMEASIA.2009.5397357.

[59] E. S. Fetzer and J. T. Orton, “A fully-bypassed 6-issue integer datapath and register file on an itanium microprocessor”, in *2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No.02CH37315)*, vol. 1, Feb. 2002, 420–478 vol.1. DOI: 10.1109/ISSCC.2002.993111.

[60] K. Noda *et al.*, “A loadless cmos four-transistor sram cell in a 0.18- mu;m logic technology”, *IEEE Transactions on Electron Devices*, vol. 48, no. 12, pp. 2851–2855, Dec. 2001, ISSN: 0018-9383. DOI: 10.1109/16.974716.

[61] A. A. Mazreah *et al.*, “A novel zero-aware four-transistor sram cell for high density and low power cache application”, in *2008 International Conference on Advanced Computer Theory and Engineering*, Dec. 2008, pp. 571–575. DOI: 10.1109/ICACTE.2008.12.

[62] A. Kumar, “Sram cell design with minimum number of transistor”, in *2014 Recent Advances in Engineering and Computational Sciences (RAECS)*, Mar. 2014, pp. 1–3. DOI: 10.1109/RAECS.2014.6799510.

[63] J. A. Fisher *et al.*, *The VEX System*, 2005.

[64] L. N. Soh *et al.*, “An innovative fpga internal core clock jitter prediction methodology”, in *2010 Asia-Pacific International Symposium on Electromagnetic Compatibility*, Apr. 2010, pp. 346–349. DOI: 10.1109/APEMC.2010.5475881.

[65] *EG-2121/2102CA series datasheet*, Epson Toyocom.

[66] Europractice - Prototyping - Bondpads, [Online]. Available: http://europractice-ic.com/prototyping%5C_bondpads.php.

[67] *UMK65LSCLLMVBBR_B UMC 65nm Low-K Multi-Voltage Low Leakage RVT Tapless Standard Cell Library Databook*, B03, UMC, Jan. 2014.

[68] *65 nm Process Die Seal Ring Rule*, 1.19, UMC, Jun. 2015.

[69] K. H. Chang and C. Browy, “Improving gate-level simulation accuracy when unknowns exist”, in *DAC Design Automation Conference 2012*, Jun. 2012, pp. 936–940.

[70] T. Zhang *et al.*, “Automatic assertion generation for simulation, formal verification and emulation”, in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2017, pp. 471–476. DOI: 10.1109/ISVLSI.2017.88.

[71] C. I. C. Marquez *et al.*, “Formal equivalence checking between high-level and rtl hardware designs”, in *2013 14th Latin American Test Workshop - LATW*, Apr. 2013, pp. 1–6. DOI: 10.1109/LATW.2013.6562666.

[72] *Calibre DRC of UMC 65nm Logic and Mixed Mode Low Leakage Purpose Application Note*, 1.1_p1, UMC, Apr. 2013.

[73] *Tms320c6413, tms320c6410 fixed-point digital signal processors data manual*, SPRS247F, Texas Instruments, Jan. 2006.

A

Source-Synchronous Interface Application Note

This is an excerpt taken from the thesis of Shizao Zhang, responsible for the design of the Source-Synchronous Interface. Both projects were developed in parallel. The application note includes all required information for the integration of the Source-Synchronous Interface and the ρ-VEX. The integration is described in detail in Chapter 5.

A.1. Overview

This source-synchronous interface (SSI) provides a low-cost solution to the inter-chip communication. It supports data rate from 421.4Mbps to 1.25Gbps. The interface provides a data transmission channel, a data reception channel, and a half-rate clock reception channel. It uses LVDS as external signaling standards. The embedded data recovery can calibrate the received data phase to be aligned to the $90°$ phase of the half-rate reference clock. The interface is designed for UMC $65\,nm$ Low-leakage technology.

A.2. Physical Information

Fig. A.1 shows the physical dimension information of the SSI. The location of each port and available accessing metal layers are also shown. The distance between the center of adjacent external signal ports is $92.5\,\mu m$. A full metal width of $12\,\mu m$ is recommended for the connections between SSI external signal ports to I/O cells, and the connections should also be as short as possible to reduce parasitic resistance. As many power connections as possible is also preferred to improve power supply.

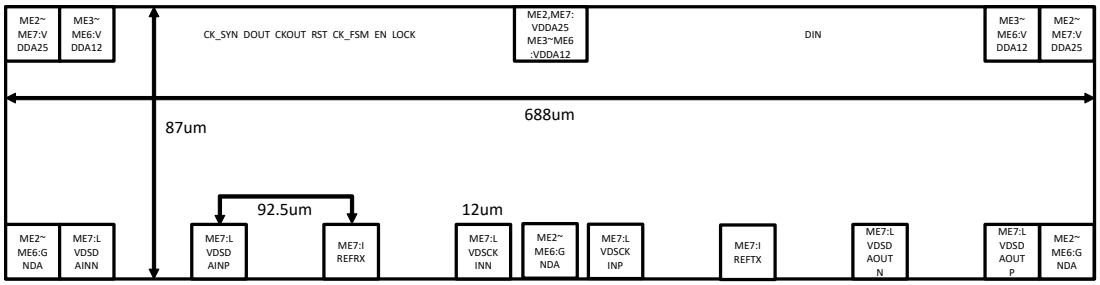


Figure A.1: Physical dimension information of SSI

A.3. Port Description

All digital signals are active high.

Table A.1: Port description

Port name	Direction	Description
CK_FSM	Input	Clock input for the internal finite state machine
CK_SYN	Input	Half-rate reference clock input for the data recovery
DIN	Input	Input data to be transmitted
EN	Input	Enable signal for the data recovery
LVDSCKINN/P	Input	LVDS half-rate reference clock input
LVDSDAINN/P	Input	LVDS data input
RST	Input	Synchronous reset for the data recovery
CKOUT	Output	Direct clock output from the LVDS clock receiver
DOUT	Output	Recovered data output
LOCK	Output	Lock indication for the data recovery
LVDSDAOUTN/P	Output	LVDS data output
IREFTX	N/A	Reference current for the LVDS transmitter
IREFRX	N/A	Reference current for the LVDS,receiver
VDDA25	N/A	2.5V power supply
VDDA12	N/A	1.2V power supply
GNDA	N/A	0V power supply

A.4. Configuration

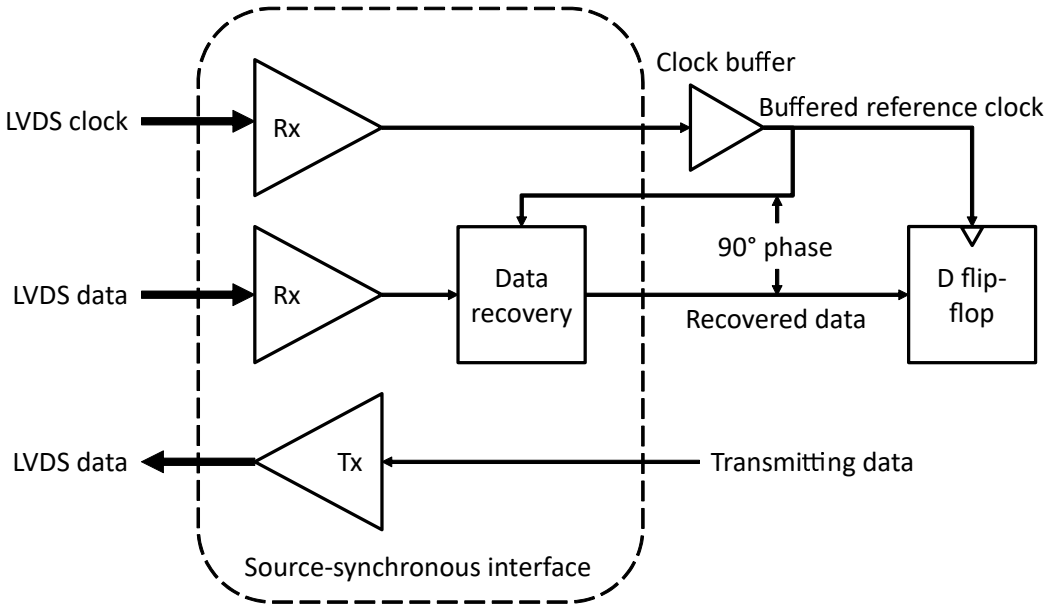


Figure A.2: Block diagram of the SSI configuration

Fig. A.2 shows the block diagram of the top-level interface, which contains an LVDS driver, two LVDS receivers and a data recovery block. One of the LVDS receiver is used for half-rate reference clock reception, and the other used for data reception. The LVDS receivers convert LVDS signals to standard digital signals which have $0\,V$ to $1.2\,V$ signal swing. The received clock signal first goes through the external clock buffers, and then return to the data recovery as a half-rate reference clock. The received data signal is phase-calibrated in the data recovery, aligning the $90°$ phase of the reference clock. In other words, the edges of the reference clock are located at the center of the recovered data eye, which

can ensure a correct data sampling by the following D flip-flop using the same reference clock. The LVDS driver converts the standard digital signal to LVDS signal without any timing manipulation, as the data will be recovered by the receiver at the other side of the communication link.

Like most other source-synchronous interfaces, the LVDS half-rate reference clock must be generated from the same clock source as for the LVDS data, as any frequency shift in the data is also reflected in the reference clock. This ensures the jitter and skew of the received data is finite relative to the reference clock, such that the data recovery is able find a correct phase for the recovered data.

The configuration shown in Fig. A.2 can be duplicated to realize a multiple instance configuration, where each instance has its own reference clock. It is also possible for multiple instances to share a common buffered reference clock, provided that all instances operate at the same data rate. For unused LVDS clock receivers, their input must be connected to power supply (VDDA25).

A.5. Data Recovery

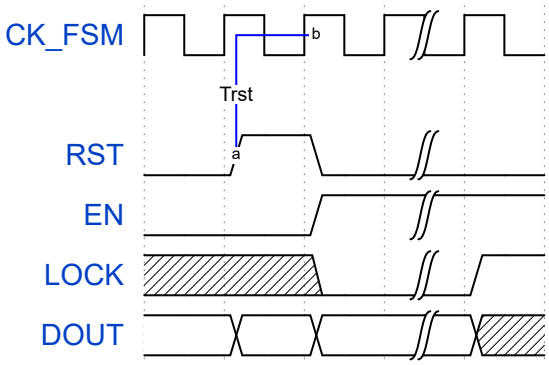


Figure A.3: Timing diagram of data recovery

The built-in data recovery can calibrate the received data phase to be aligned to the $90°$ phase of the half-rate reference clock. Fig. A.3 shows the timing diagram of the data recovery. The reset signal “RST” must become valid-high for at least $1\,ns$ before the clock signal “CK_FSM ” to ensure proper reset for the data recovery. If the enable signal “EN” is low, the data recovery is not working, and there will be a minimum delay from LVDS data input to the data output at “DOUT”. After ”EN” becomes high, the data recovery starts to calibrate the data phase according to the reference clock at “CK_SYN ”. It usually takes 24 cycles before the lock signal “LOCK” becomes valid-high, depending on the operating data speed and clock frequency at “CK_FSM ”. As soon as “LOCK” becomes valid-high, the phase of the recovered data at “DOUT” is calibrated to the $90°$ of the reference clock with a finite phase error.

In order for the data recovery to work properly, a square-wave data sequence, i.e.“101010...”, is required to start appearing before the data recovery starts to work, until the lock signal “LOCK” becomes valid-high. This is to ensure that the data recovery skips the possible initial unstable operation of the LVDS transceiver, when much higher jitter could be present.

A.6. Port Input Capacitance

Table A.2: Port input capacitance

Symbol	Description	Min.	Typ.	Max.
C_{CK_FSM}	Input capacitance of port CK_FSM		$24.8\,fF$	
C_{CK_CYN}	Input capacitance of port CK_SYN		$17.7\,fF$	
C_{DIN}	Input capacitance of port DIN		$7.9\,fF$	
C_{EN}	Input capacitance of port EN		$5.8\,fF$	
C_{RST}	Input capacitance of port RST		$25.8\,fF$	

A.7. Constraints

Table A.3: Constraints

Symbol	Description	Min.	Typ.	Max.
C_{CKOUT}	Load capacitance of port CKOUT			$5\,fF$
C_{DOUT}	Load capacitance of port DOUT			$20\,fF$
T_{r,CK_SYN}	Input transition time of port CK_SYN			$50\,ps$ (1)
DC_{CKREF}	Duty cycle of the reference clock for data recovery		50% (2)	

(1) A transition time of $50\,ps$ or less is required for the data recovery to work properly. The transition time is defined as the time interval between 10% to 90% of the full signal swing.

(2) A duty cycle as close to 50% as possible is preferred for the data recovery to work properly.

A.8. AC Characteristics

Table A.4: AC Characteristics

Symbol	Description	Min.	Typ.	Max.
F_{CK_FSM}	Clock frequency for the internal FSM			$500\,MHz$
T_{bit}	Data bit time, half clock period	$800\,ps$		$2373\,ps$
F_{bit}	Data bit rate	421.4Mbps		1.25Gbps
F_{CK_SYN}	Reference clock frequency	$210.7\,MHz$		$562.5\,MHz$

A.9. DC Characteristics

All digital signals are active high and (should) have $0\,V$ to $1.2\,V$ signal swing. Digital signals include CK_FSM, CK_SYN, DIN, EN, RST, CKOUT, DOUT and LOCK.

All LVDS signals (should) have nominal $1.25\,V$ common-mode voltage, and nominal $350\,mV$ differential voltage.

In the following graphs, V_{out} means the differential output voltage measured at the LVDS driver output ports; R_{out} means the total load resistance measured at the LVDS driver output ports. Therefore, designers should take care of the actual resistance of the top-level chip routing wire, package parasitic resistance, and the resistance of the transmission line, as well as the resulting voltage drop.

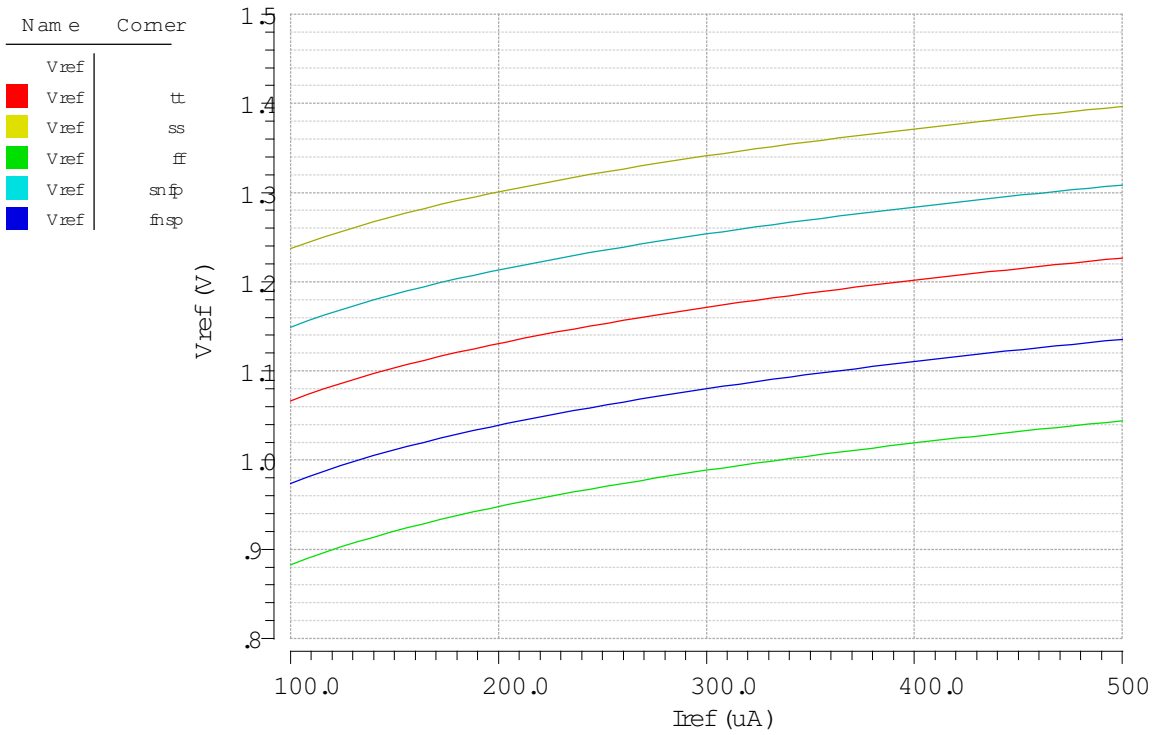


Figure A.4: Reference voltage vs. reference current of LVDS driver

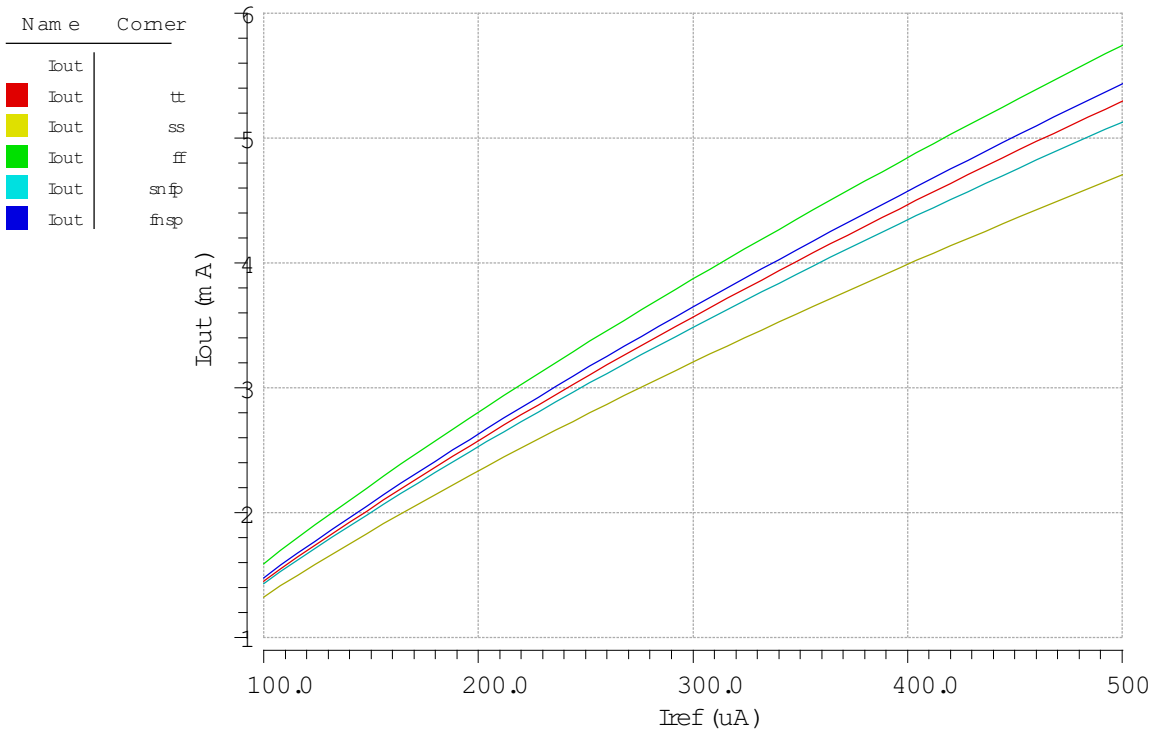


Figure A.5: Output current vs. reference current of the LVDS driver @R_{out}=110 Ω

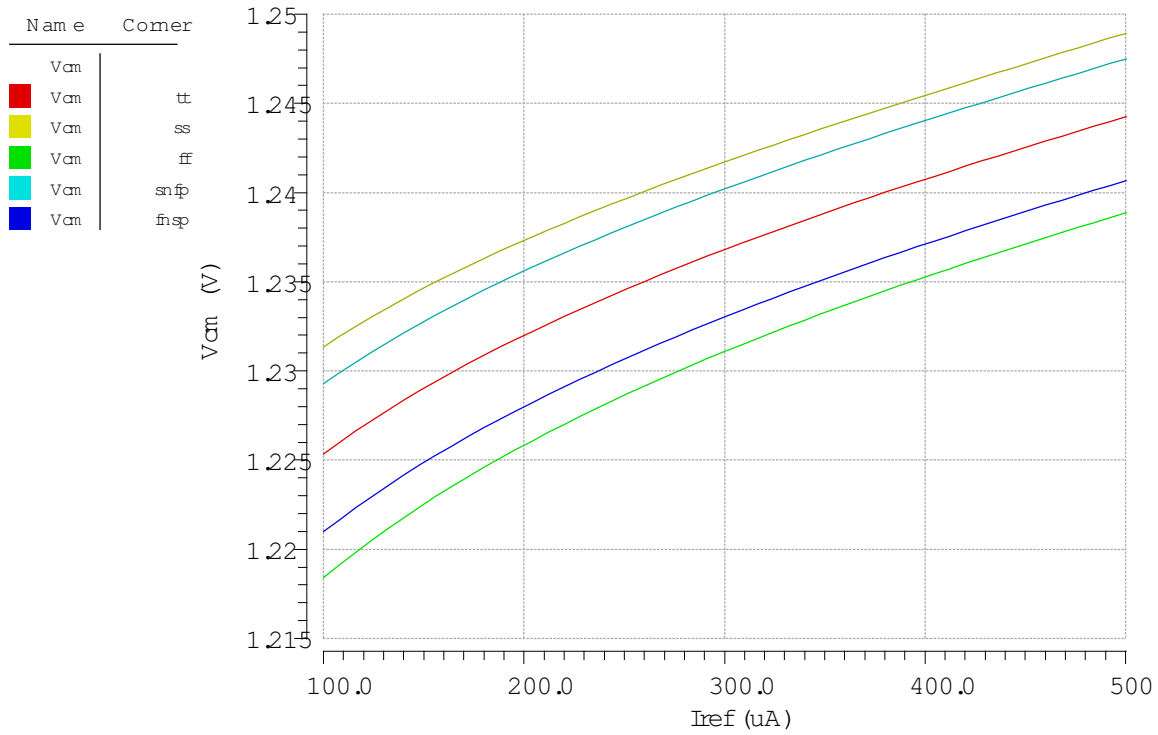


Figure A.6: Output common-mode voltage vs. reference current of the LVDS driver @R_{out}=110 Ω

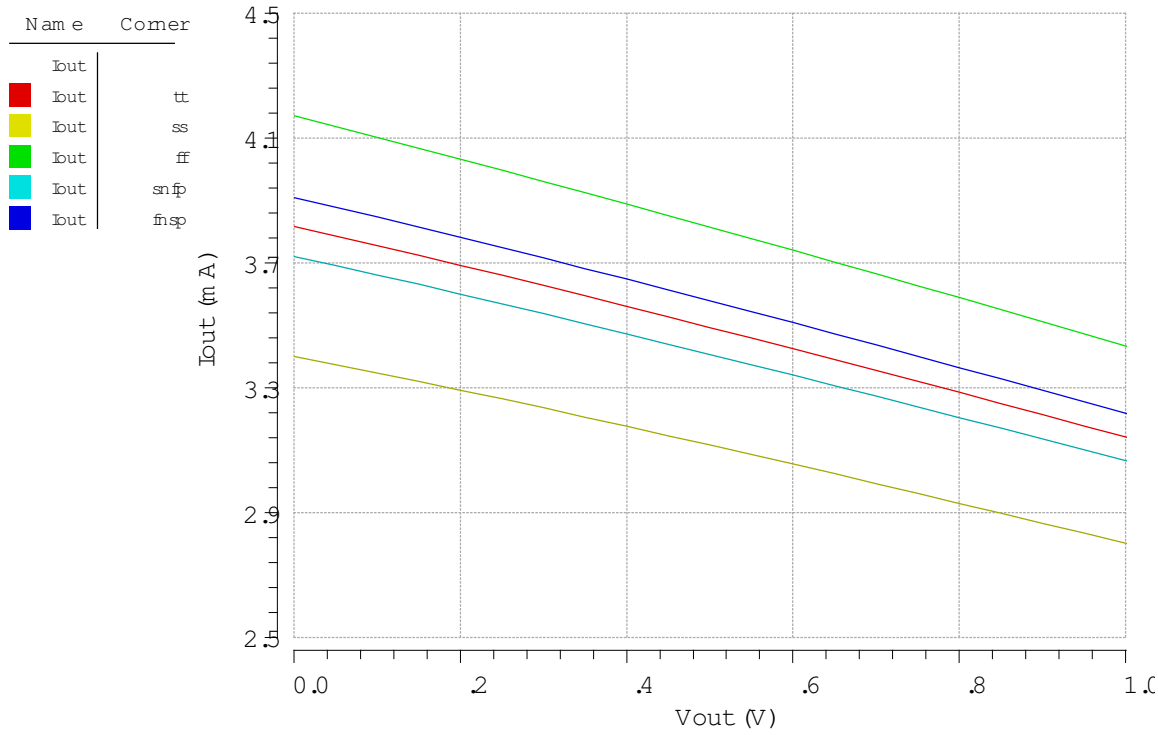


Figure A.7: Output current vs. output differential voltage of the LVDS driver @Iref=300 μA

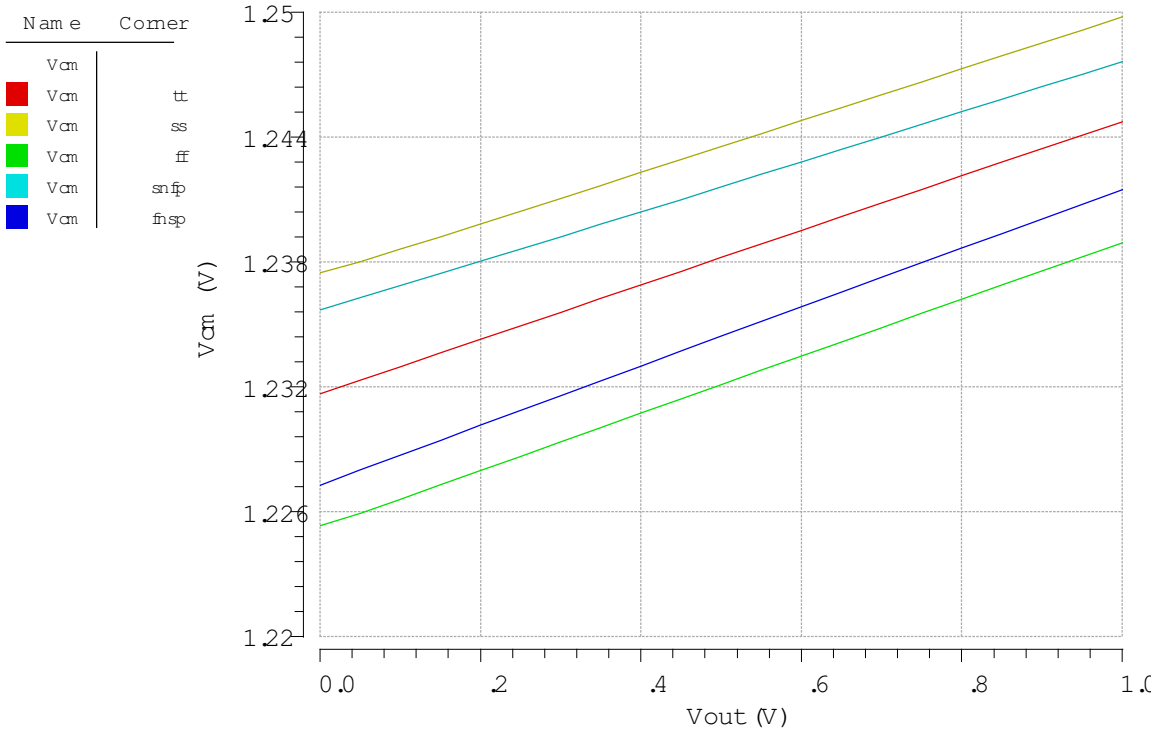


Figure A.8: Output common-mode mode voltage vs. output differential voltage of the LVDS driver @Iref=300 μA

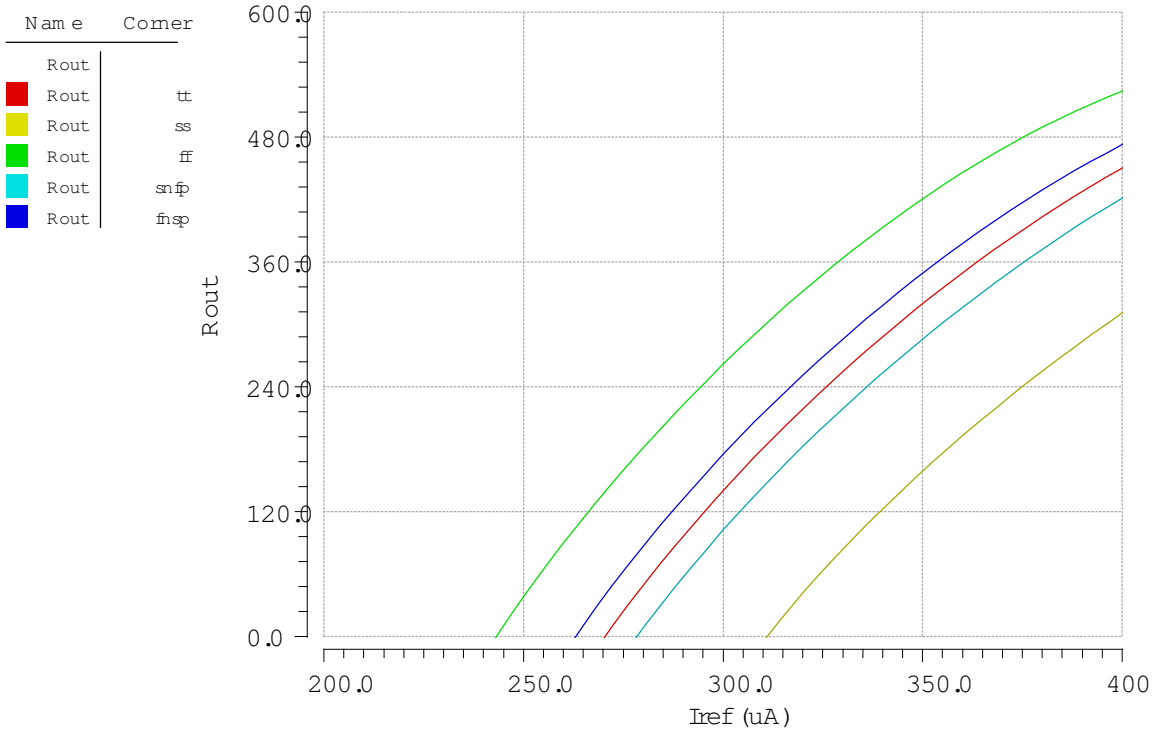


Figure A.9: Total output resistance vs. reference current of the LVDS driver @Iout=3.5 mA

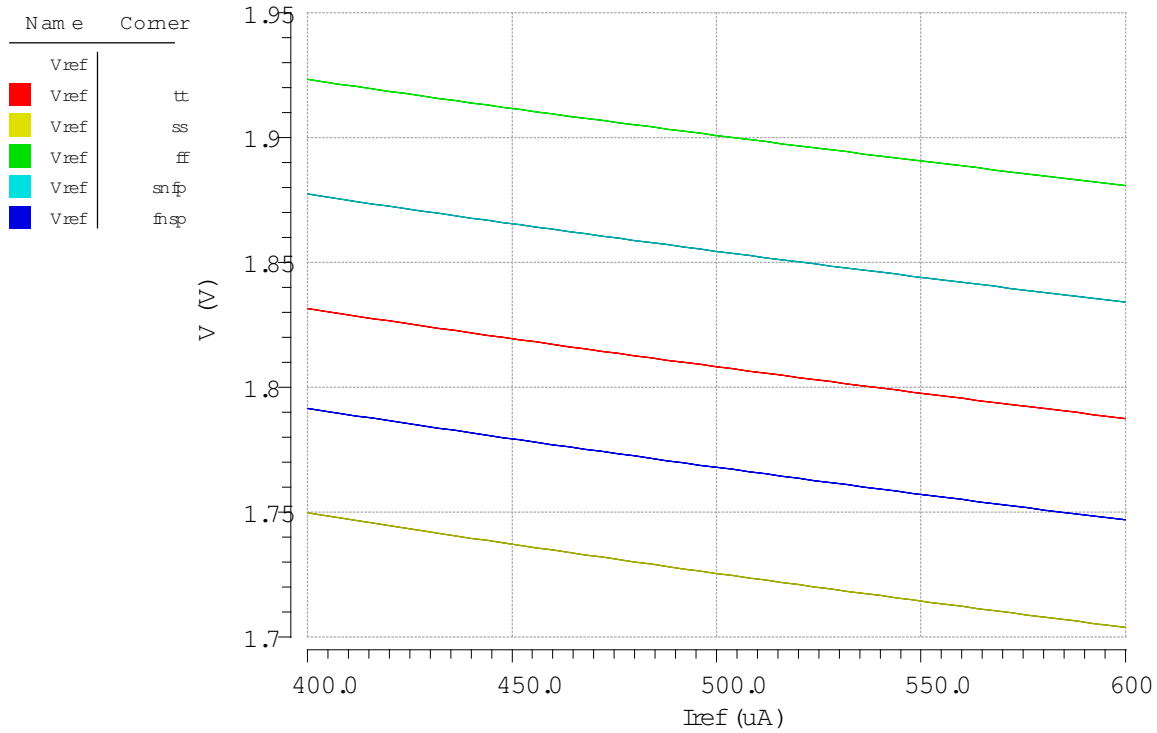


Figure A.10: Reference voltage vs. reference current of LVDS receiver

B

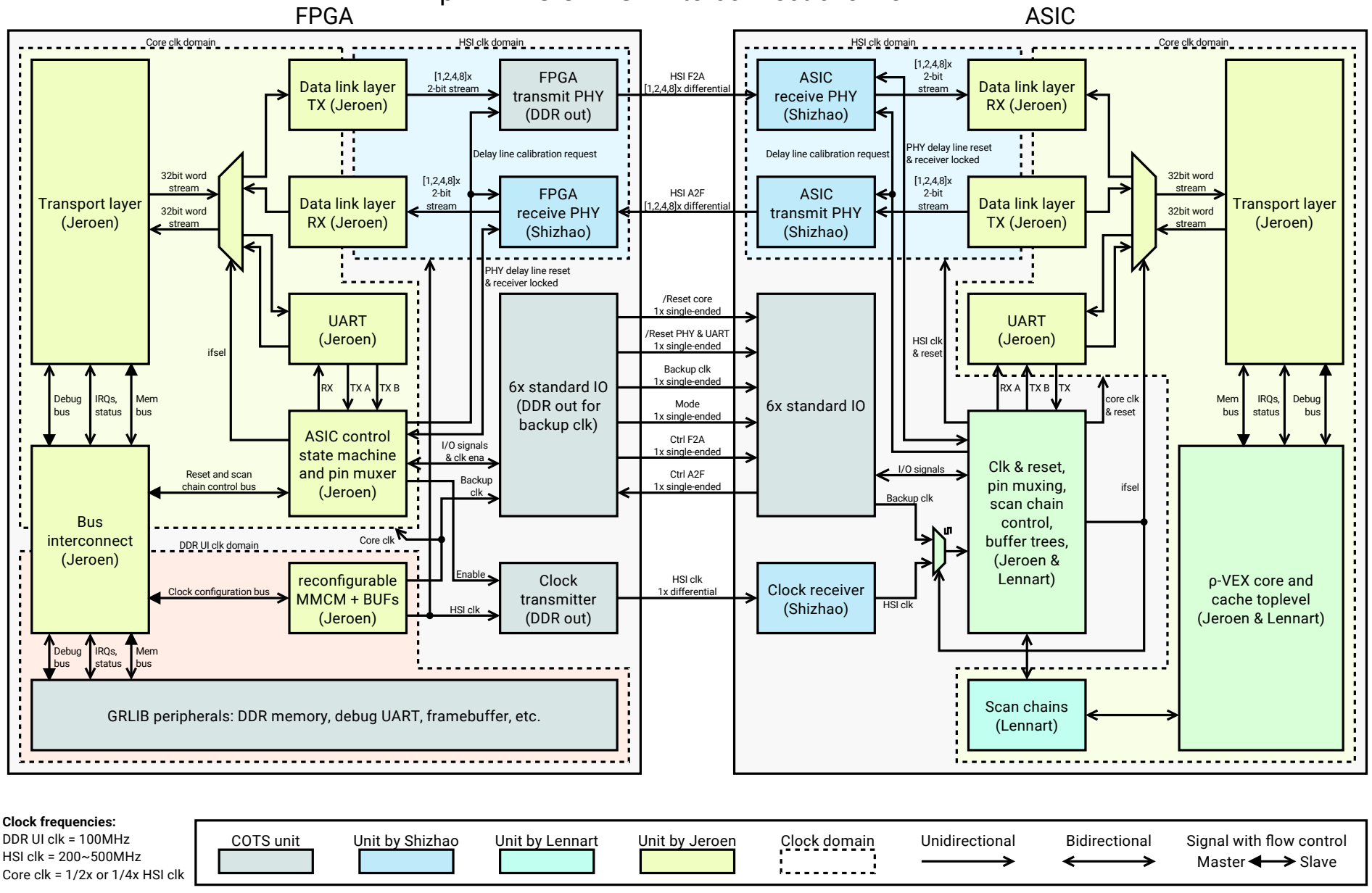
HSI overview

The High Speed Interface is a protocol responsible for communication with an FPGA. This FPGA will hold the full data and instruction memory of the ρ-VEX. By only using small on-chip caches, the area and thereby the price of this prototype is reduced.

As the High Speed Interface is developed by Jeroen van Straten, it is not directly part of this project. The important details have been discussed and are listed in the following overview. The overview includes all components that are necessary for communication by both the ASIC and FPGA. Furthermore, a pin-sharing scheme and cross-clock boundaries are described. The pin-sharing scheme allows the ASIC to use a minimal number of digital pads, allowing more pads to be used for the SSI. The cross-clock boundaries are required as the SSI will run at either four times or two times the ρ-VEX core frequency.

The rest of the protocol deals with configuration and data packets. As these are part of a higher-level protocol, they are less important to the actual ASIC design. However, they are simulated and debugged as presented in Section 9.2.4.

ρ-VEX ASIC-FPGA interconnect overview



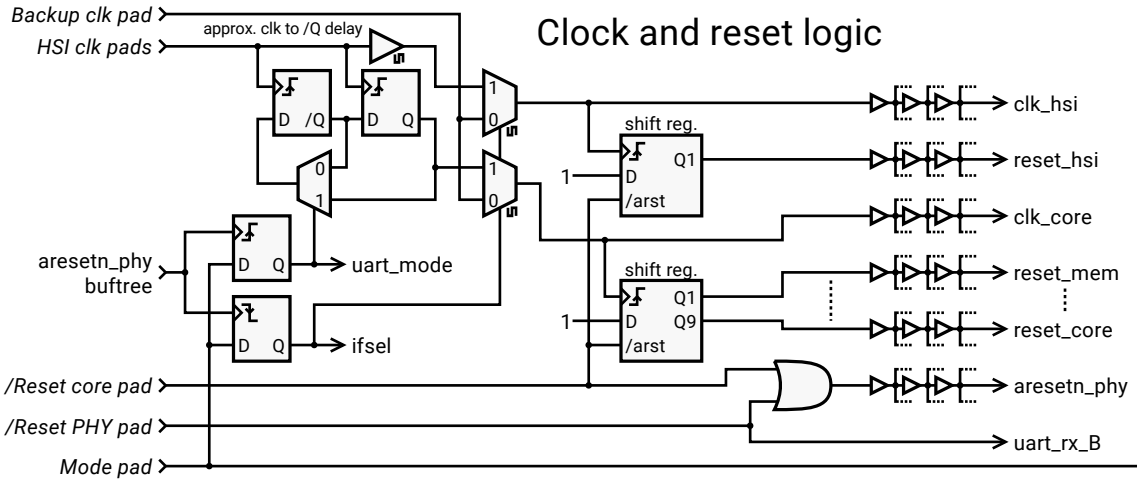
Clock frequencies:
DDR UI clk = 100MHz
HSI clk = 200~500MHz
Core clk = 1/2x or 1/4x HSI clk

COTS unit | Unit by Shizhao | Unit by Lennart | Unit by Jeroen | Clock domain | Unidirectional | Bidirectional | Signal with flow control (Master ↔ Slave)

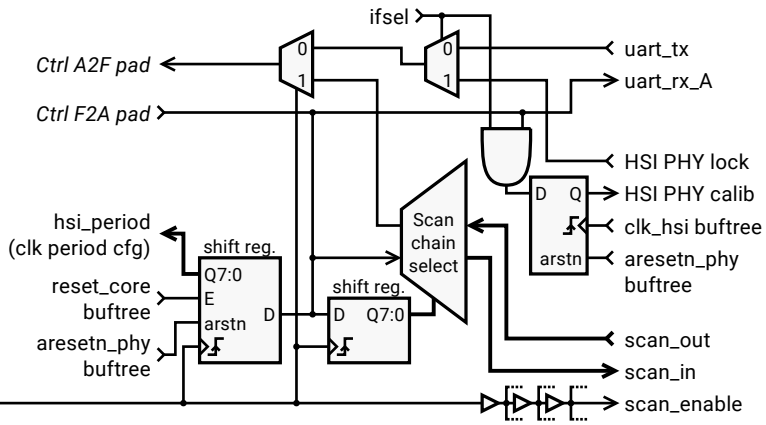
Initialization and pin sharing

In order to be able to devote as many pins to the HSI data pairs as possible, the other functions of the ASIC are shared over only six pins. These functions include the backup UART interface for if the HSI does not work properly, HSI initialization control signals, resets, interface mode selection, and the scan chain interface. The pins are connected to the internal signals using a very simple, though perhaps somewhat controversial circuit as many signals are used to clock in various control registers, shown below. The timing diagrams show the initialization/reset sequencing for the ASIC for the two interface modes; the scan chain protocol is shown on another page. The sequencing is handled by a state machine in the FPGA.

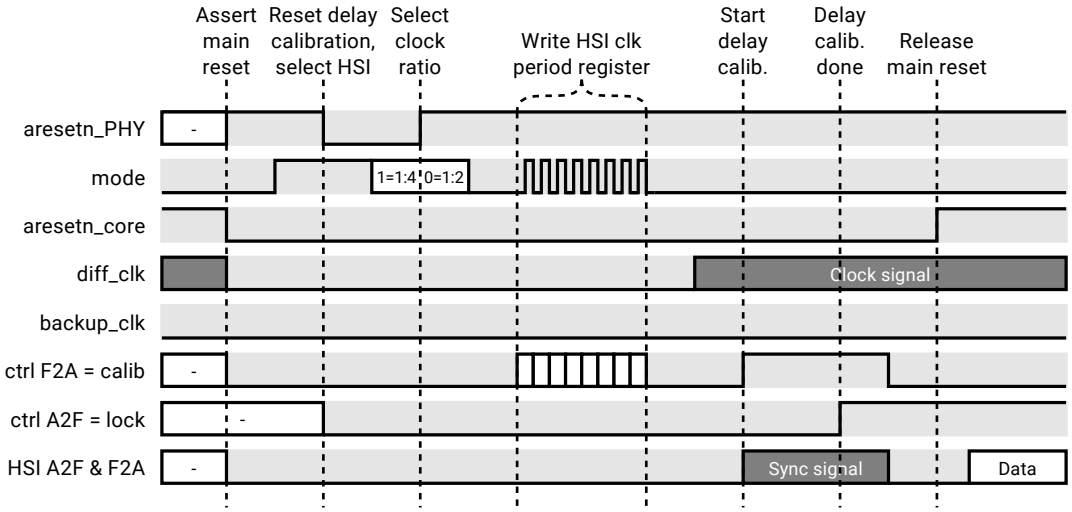
Clock and reset logic



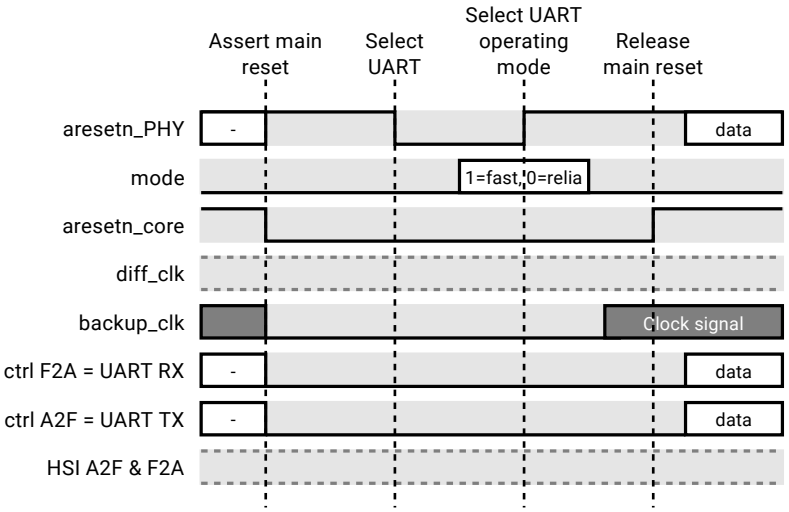
Pin sharing logic



HSI mode initialization sequence

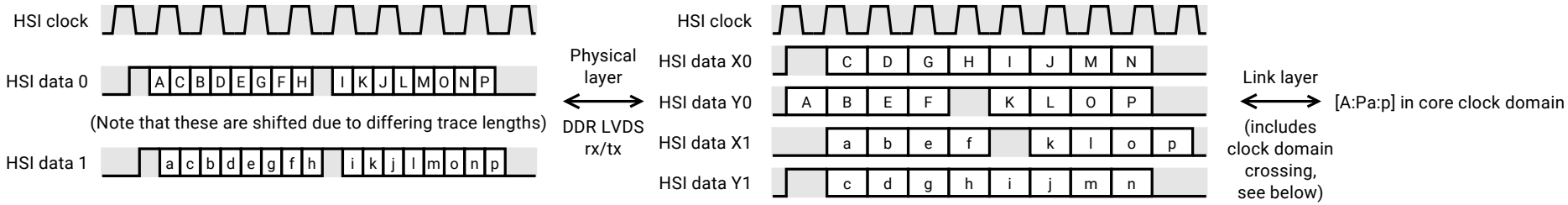


UART mode initialization sequence

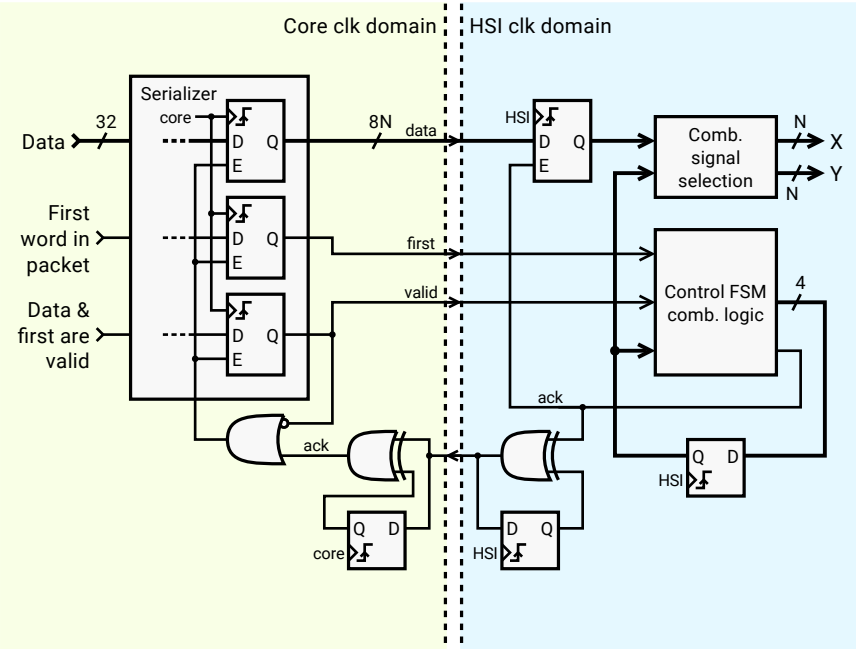


HSI link protocol

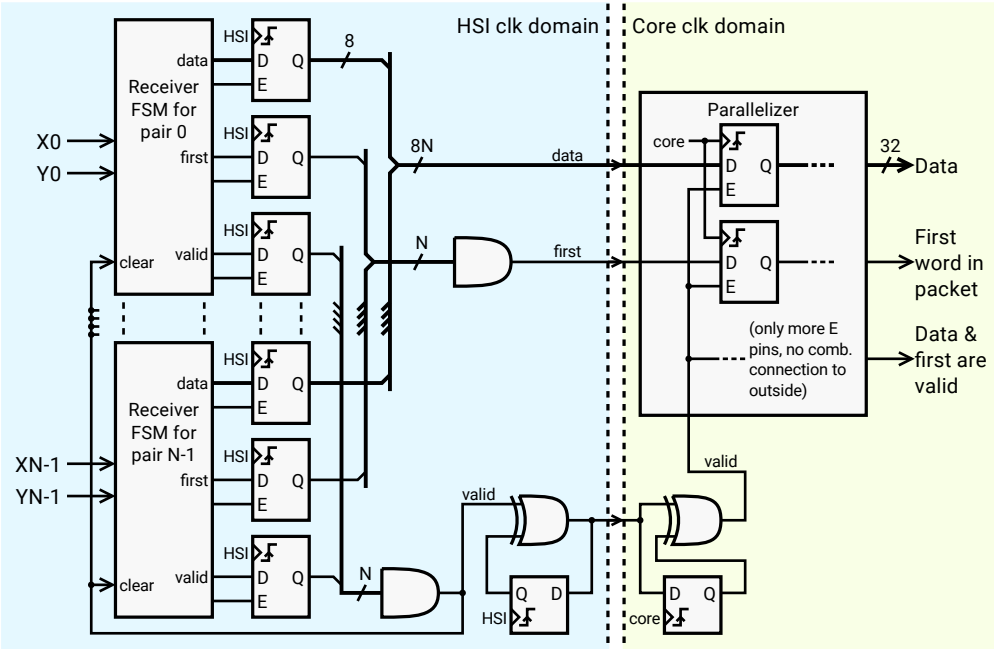
HSI is split into three layers: physical, link, and transport. The physical layer consists of the LVDS driver/receiver, delay compensation/calibration logic, and DDR <--> 2-bit SDR conversion. It runs at the HSI clock, which may be twice as fast or the same speed as the core clock. The HSI can use 1, 2, 4, or 8 differential pairs in either direction; each pair has its own functionally independent PHY unit. The data link layer converts 5-bit "flits" on the pairs into 32-bit words in the core clock domain. It can compensate for up to 1 HSI clock cycle's worth of inter-pair latency. It also detects whether subsequent data words form a single packet. The transport layer interfaces between the packet streams and bus interfaces; the packets are shown on another page. Note that the physical and link layers may be exchanged for the UART (also on another page) in case something is wrong with the HSI. The timing below shows how the HSI works for 2 differential pairs. A single word is exchanged; longer packets are simply transmitted with no idle time in between. A packet boundary requires one or more idle half-cycles.



HSI transmitter clock domain crossing



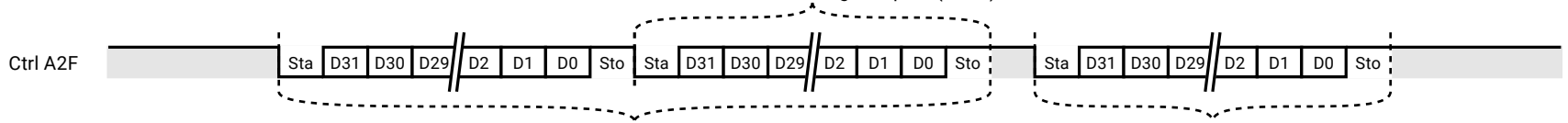
HSI receiver clock domain crossing



UART link protocol

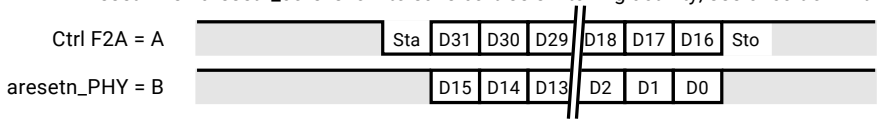
The UART is an alternative to the HSI, which may be used if the LVDS pads or delay logic does not work as intended. It must also be used for the scan chain to work properly, as scanning requires the clock to stop temporarily, which will not work properly in conjunction with the delay lines. Note that the protocol is not a typical UART anymore (longer frames, two data pins from FPGA to ASIC) in order to get the most bandwidth out of it. Two modes are available, fast and reliable. The only difference is the baud rate and degree of signal conditioning.

One data frame consists of 32 data bits, a low start bit, and a high stop bit (32N1). The ASIC to FPGA direction is shown.



FPGA to ASIC direction

The aresetn_PHY signal is used as a secondary data pin to speed up the F2A direction. It is only used for data; start/stop bits are only transmitted on Ctrl F2A. The aresetn_PHY pin only works as PHY reset when aresetn_core is low to save buftree switching activity; see circuit on init. seq. page.

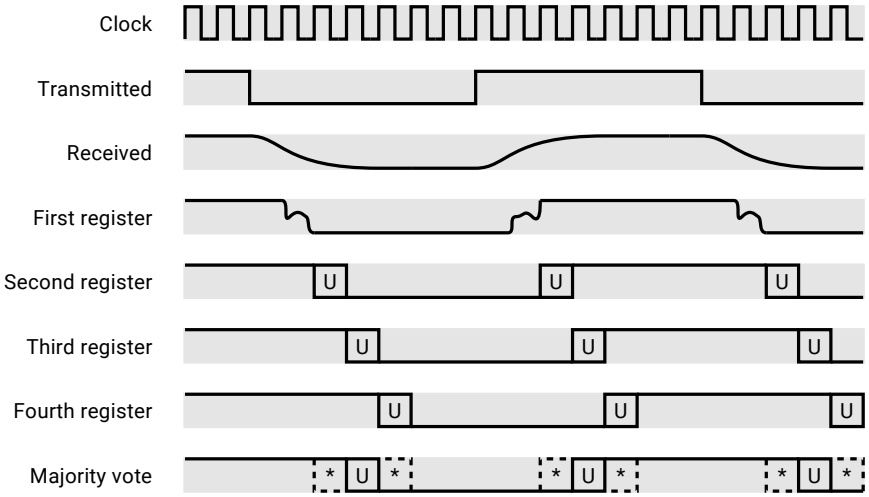


Fast mode

When the UART is configured in high-speed mode, the bit time is one backup_clk cycle. The ASIC always samples and sets up data at the rising edge of the clock, whereas the FPGA can set up and sample at either the rising or falling edge, allowing timing problems to be solved manually.

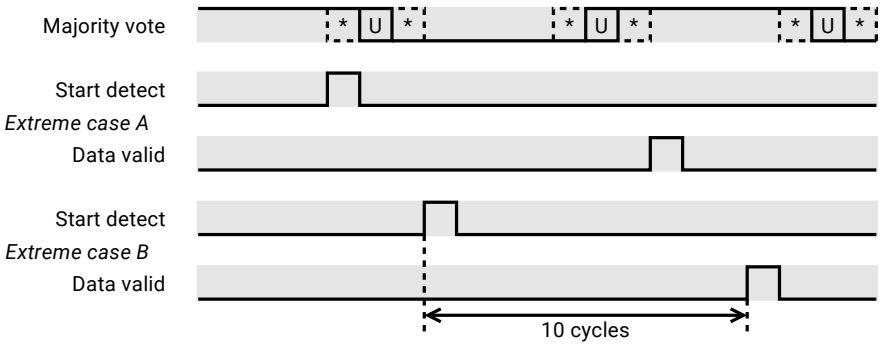
Reliable mode

In reliable mode, the bit time is seven backup_clk cycles. A shift register is used at the input for synchronization, and a three-way majority voter is used to filter the signal.



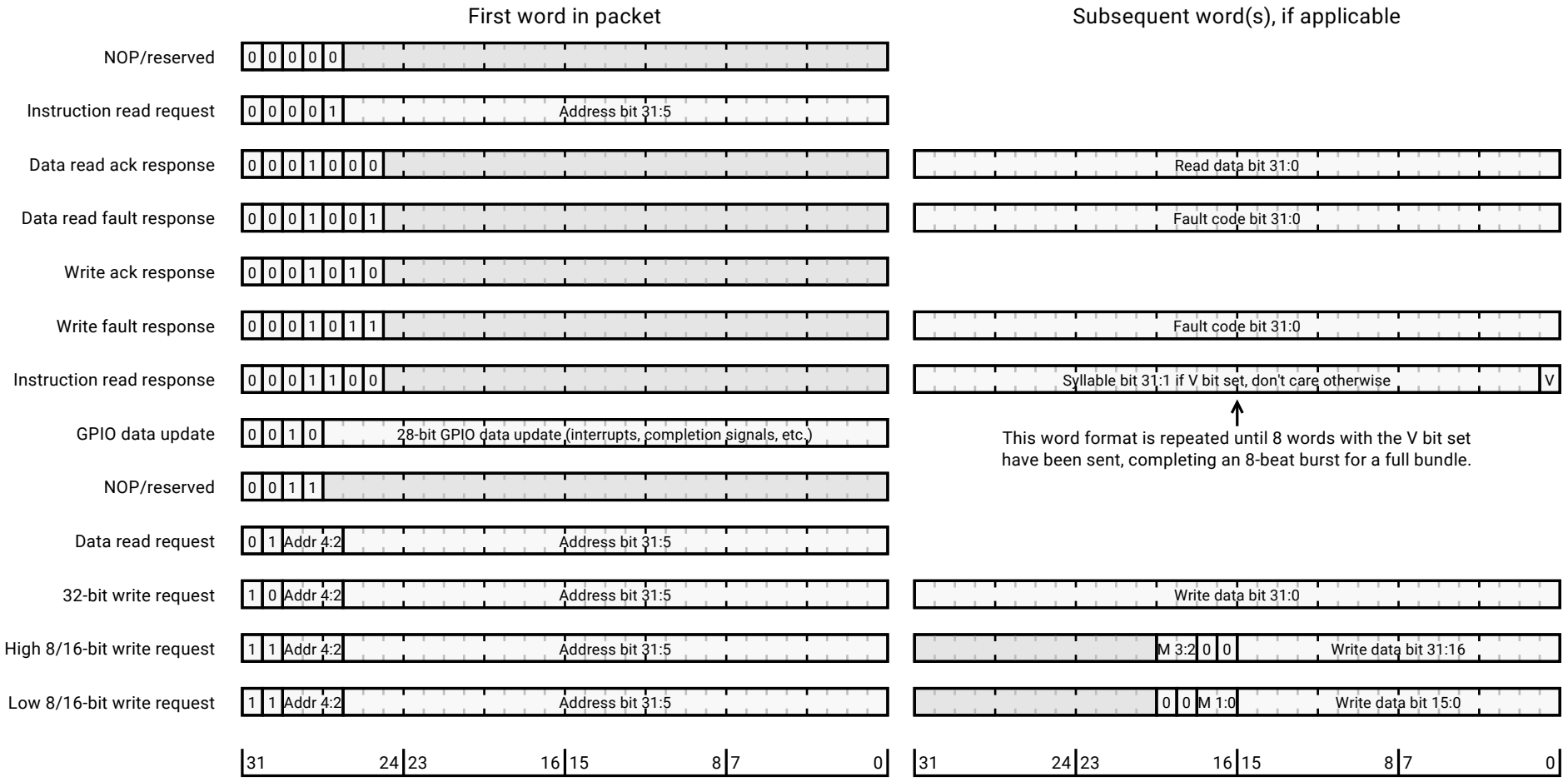
* bit is undefined if 1-bit glitches/errors are tolerated

A start bit is detected when the majority vote output goes low after the bus being idle for some time. The first data bit is taken from the majority vote output 10 cycles later, and every subsequent bit every 7th cycle, until 32 bits have been received. 7 cycles after the last data bit the start bit detection logic is re-enabled. If no start bit is detected within 14 cycles after the last data bit, the current packet is terminated and the next data frame will have the "first" flag set.



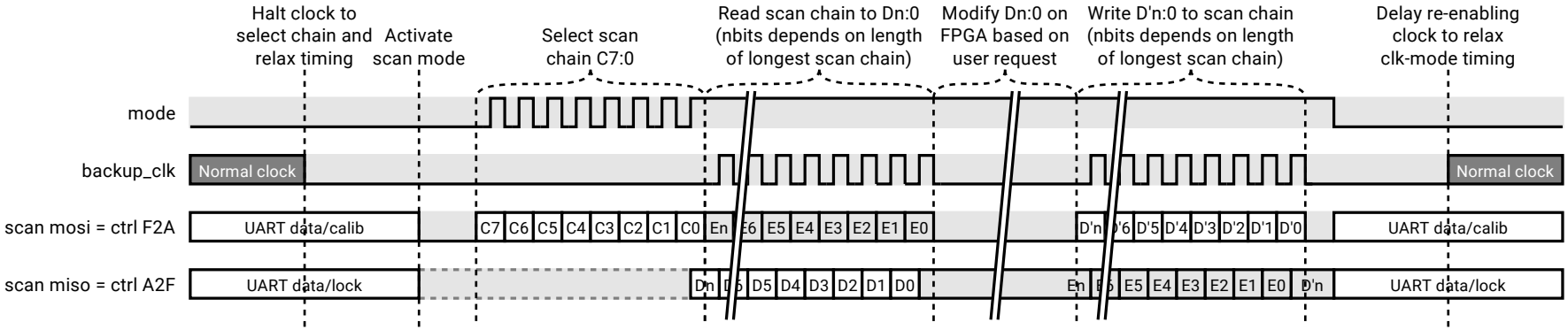
Transport layer

The transport layer logic is used in HSI and UART mode. It encodes memory bus master and slave signals in either direction, as well as 28 bits of slow-changing I/Os used for interrupts and core status flags. The transport layer is symmetrical, allowing identical logic to be implemented on the FPGA and ASIC sides. The only thing is that the FPGA side will never generate instruction read requests, and thus the ASIC side does not need the logic to handle such requests.



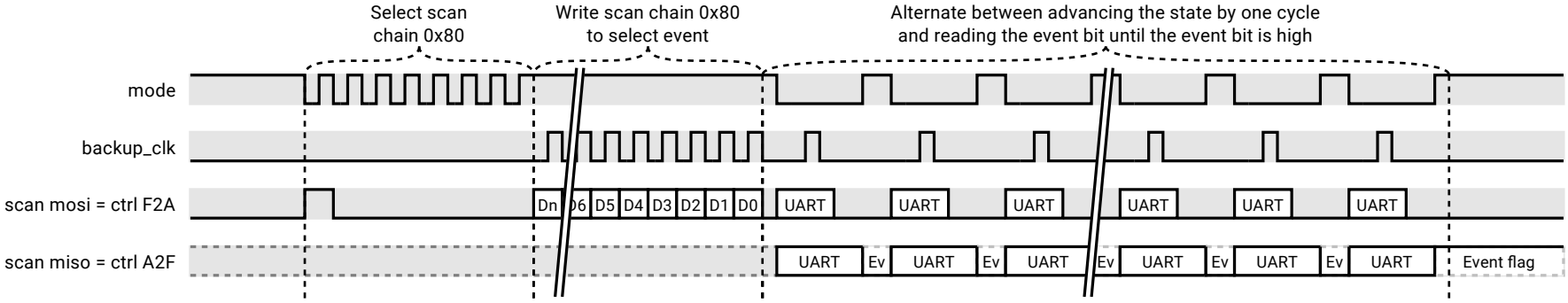
Scan chain interface protocol

The diagram below shows a read-modify-write scan chain operation. The scan chain index is clocked in on the rising edge of the mode signal, as the clock signal must be disabled while this is done (otherwise the scan chains would start clocking in garbage). The scan data is clocked in and out on the rising edge by the ASIC, and on the falling edge by the FPGA. The number of clock cycles while mode is high must ALWAYS be an integer multiple of the length of the longest scan chain, and all scan chains must be a positive integer times smaller than the longest chain. Otherwise, scan chain states will be undefined after mode is released.



The ASIC always samples and sets up scan data at the rising edge of its clock. The FPGA samples and sets up data at the falling edge of the ASIC clock. If the FPGA passes the MISO line through to the MOSI line (with or without a single falling-edge register in between) the scan chain is unaffected after the usual amount of clocks.

Scanning stops the UART data transmission process. This makes single-stepping the core a difficult process when the cache is busy fecthing data. To mitigate this, the scan chain 0 output must be combinatorially connected to a chain-controlled multiplexer that selects a to-be-monitored event. This will be called the event flag. The idea is that this bit can be read without transferring a whole scan chain. Furthermore, because scan chain 0 has all zeros in its index, rising edges on the mode pin do not affect the selected chain, as long as UART F2A is held low during the transition. This allows the following to wait for an event. The multiplexer that selects the event should be controlled by a short scan chain at index 0x80; because the scan index shifts left, the scan index will be 0 after the next clock/mode transition.



Note that the scan interface can only be used in UART mode, as the HSI delay calibration logic will most likely not work with clock strobes.

C

Signoff quality checklist

A big part of this project deals with verification (**G**.4a) and documentation (**G**.6). As discussed in Section 9.6, signoff is the ensemble of checks required before an IC is ready for fabrication. It ensures a valid, functional design and manufacturability. All discussed checks are performed as part of this project.

To make sure a followup project achieves the same signoff quality, a checklist is created with all required steps and verification. It is recommended to only continue IC fabrication if the whole checklist is valid. Furthermore, simulation details are listed to verify functionality as well as compare performance between simulated and real life data. This comparison is useful to check the final performance, before a new production is run.

The tables with actual data are presented in Section 10.1. The results and any signoff discrepancies are also listed there, as well as the waivered checks. These empty tables are useful for future reference.

Table C.1: All performed software checks before tape-out

		Pre CTS	Post CTS	Post Route	Signoff (Tempus)
Synopsys	Setup WNS				
	Comb. loops				
Encounter	Setup WNS				
	Hold WNS				
	Early DRVs				
	Chip utilisation				
	IR-drop				
	EM effects				
	SI*				
	Filler DRVs				
	Geometry DRVs†				
	Wiring DRVs‡				
	Antenna DRVs				
Calibre	DRC				
	Antennae				
	Metal fill				
	LVS				
	ERC				
Virtuoso	Bondpads				
	Visual inspection				

*Checks for transition glitches caused by switching activity on nearby wires.
†The geometry check is an early form of DRC.
‡The wiring check is a form of LVS, without the actual transistors.

Table C.2: Program simulation execution time

		Pre CTS	Post CTS	Post Route	Final chip
ucbqsort-fast	4:1 HSI				
	2:1 HSI				
	UART				
	Scan mode				
ucbqsort	8-way, 4:1 HSI				
	2-way, 4:1 HSI				
compress	8-way, 4:1 HSI				
	2-way, 4:1 HSI				
convolution	8-way, 4:1 HSI				
	2-way, 4:1 HSI				

Table C.3: Program simulated power and energy consumption, excluding IO

		Post Route	Final chip
ucbqsort-fast	4:1 HSI		
	2:1 HSI		
ucbqsort	8-way, 4:1 HSI		
	2-way, 4:1 HSI		
compress	8-way, 4:1 HSI		
	2-way, 4:1 HSI		
convolution	8-way, 4:1 HSI		
	2-way, 4:1 HSI		

D

ASIC synthesis tools user guide

As part of (**G**.6), all TCL scripts and the whole project flow are documented. In different README files and comments, every function and design decision is documented. To get a new student started with the project, without needing any knowledge of the underlying software, a global user guide was written. The user guide explains all environment variables and aliases used throughout the different software.

The final part, ‘Full design flows’, lists different possibilities for starting up a new project, converting the full RTL to a GDSII layout and perform all verification steps. These simple commands are all that is necessary to get started.

The documentation is created in and added to the bitbucket project. All source and comments are available.

Starting a new project

Start with downloading all scripts necessary for an ASIC synthesis run.

```
    git clone https://lvbremen@bitbucket.org/lvbremen/asic-synthesis.git .
```

Then download the ρ-VEX project. The branch `asic-lennart` is a special ASIC synthesis ready version of the core. It contains proper reset signals and a High Speed Interface for communicating with large blocks of external memory.

```
    git clone https://lvbremen@bitbucket.org/lvbremen/rvex-asic.git --branch
asic-lennart IP/rvex-asic
```

To initialise all tools, a configuration script should be loaded. Any configuration script is located in the 'CONF' directory and initialises some required environment variables, sets up design files and pre-compiles some macro commands used throughout the project. Any time a terminal is opened, this configuration file must be sourced again. For the default design, run `source CONF/full.conf`.

Downloading simulation scripts

(The following is only necessary for simulation and can be skipped for synthesis.)
Download tool binaries to the folder `IP/rvex-asic/tools`. all binaries can be downloaded from the rVEX site: - rvex-elf32-32bit - rvex-3.43 - vexparse

Compiling libraries

- All default libraries required by ModelSim, Design Compiler and Encounter can be compiled with the macro command `make_lib`.
- All memory libraries definitions require minor fixes, run these using `fix_mem`.
- All libraries are required to be compiled for Design Compiler, do this using `extract_db`.
- For dynamic power and rail analysis, Power Grid Libraries are required. If power analysis is not necessary, this library can be omitted. The required commands can be run with `par_pgl`. Note that these include detailed SPICE simulations and can therefore take quite a long time. `par_pgl_nohup` allows for detaching the command line.
- For testing the ρ-VEX behaviourally, compile the testbench with `comp_rvex` and run a simulation program using `sim_rvex`. By default the program 'ucbqsort_fast' is run. This program outputs 'ucbqsort_fast: success' if completed successfully.
- For an ASIC version, some changes are made in the code and behavioural parts are swapped for hardware parts. To test these changes, run similar commands `comp_asic` and `sim_asic`.

Synthesising the ASIC

After all libraries are set up correctly, Synthesising the ASIC consists of several steps. More specifically, these steps include synthesis in Design Compiler, Place and Route in Encounter and

verification in Calibre.

Synthesis in Design Compiler

Synthesis can optionally be performed with switching activity aware power optimisations. This will result in a (situationally) power efficient design. If not, the following command can be skipped. Creating a switching activity (SAIF) file is as easy as running a simulation while saving all registers outputs. This can be done with the macro command `sim_asic_saif`. Note that this does require the simulation binaries to be set up correctly.

After either performing or skipping power annotation, the actual synthesis can start. This can be run using the macro command `syn`. As this command can take quite a while (several hours), it can also be run using `syn_nohup`, which allows for running in the background and detaching the command line. Synthesis should complete without errors.

To verify the synthesised design, the commands `comp_syn` (compile synthesis testbench) and `sim_syn` (simulate synthesis testbench) can be run. Note that after synthesis, some timing aware components are not entirely correct yet. Therefore, the testbench will run in UART mode using a slightly lower clock frequency. This will be fixed in Encounter.

Place and Route in Encounter

Similar to synthesis, PAR can be run using either `par` or `par_nohup`. Note that these commands will take approximately two to three days (on 2 CPU's) to complete.

If at any stage optimisations are required, the extra commands can be run. To optimise placement, `setenv ITERATE_PLC x` with `x` any number of optimization cycles. After that, run `par_opt_plc` to perform the optimisation. Similarly, `setenv ITERATE_CTS x && par_opt_cts` and `setenv ITERATE_DRC && par_opt_drv` can be run to optimise Clock Tree Synthesis or post-route Design Rule Violation (including timing) optimisations respectively. `setenv DESIGN_RESUME x && par_resume` can be run to resume the design stage after an optimisation, or just after any design stage. `x` should be any stage where to resume, numbered according to the script files in the 'PAR/BIN' directory.

After PAR, simulations can be run using `comp_par` and `sim_par`. These simulations do run at full clock speed and using the High Speed Interface simulation. All should complete successfully.

After PAR the most accurate power numbers can be calculated. Again optionally switching activity can be used for even more accurate (situationally) power numbers. Use `sim_par_saif` to generate the SAIF file. Open Encounter and run the script `source PAR/BIN/power_analysis.tcl` for performing the power analysis. Power numbers are outputted to the directory 'PAR/POW'.

Verification in Calibre

To verify the design, several steps are needed. First of all, all simulations must run successfully. All log files (in 'SYN/LOG' and 'PAR/LOG') should not contain any errors and minimal warnings. Post-synthesis setup timing (in 'SYN/RPT/{design}/timing_extended.rpt') must be met. Post-route

setup AND hold timing (in 'PAR/RPT/{design}/timeDesign') must be met. Then the following verifications can be performed in Calibre, to ensure manufacturability as well.

Design Rule Checks are performed to see if any defects can occur during manufacturing. These include wire spacing, minimum-cut vias, wire antennae, metal density and many more. Run all checks at once using `cal_drc` or `cal_drc_nohup`. Results are available in the 'CAL/RPT' directory.

Layout Versus Schematic checks if the PAR'ed design is actually the same as what was simulated using the netlist. Any inconsistencies like unconnected wires, overlapping/shorting devices and correctly connected bulk, power and ground pins (ERC) are checked using LVS. Run all checks using `cal_lvs` or `cal_lvs_nohup`. These results are also available in the 'CAL/RPT' directory.

Making code changes

- Changes in memory instances, compiled with the Faraday Memaker, require the command `fix_mem`. This fixes some minor bugs in the output files.
- Changes in any .lef (Library Exchange Format) file, require re-extraction of .db files. These are a compiled format necessary for Design Compiler. The .db files can all be extracted simultaneously using `extract_db`.
- Any change in source code (including functional changes of libraries) require simulations to be re-run for verification. Starting with the `check_asic` command. This compiles all code and checks for ASIC inconsistencies (like sensitivity lists). All errors must be fixed and all warnings should at least be checked and fixed as much as possible. ASIC simulation (using `comp_asic` and `sim_asic`) must still complete successfully.

Full design flows

Minimal flow

```
    git clone https://lvbremen@bitbucket.org/lvbremen/asic-synthesis.git .
    git clone https://lvbremen@bitbucket.org/lvbremen/rvex-asic.git --branch
asic-lennart IP/rvex-asic
    source CONF/full.conf
    make_lib
    fix_mem
    extract_db
    syn
    par
```

Full flow with simulations, power analysis and Calibre checks

```
    git clone https://lvbremen@bitbucket.org/lvbremen/asic-synthesis.git .
    git clone https://lvbremen@bitbucket.org/lvbremen/rvex-asic.git --branch
asic-lennart IP/rvex-asic
    source CONF/full.conf
```

Copy binary folders (rvex-elf32-32bit, rvex-3.43, vexparse) to 'IP/rvex-asic/tools'.

```
make_lib
fix_mem
extract_db
par_pgl
comp_rvex
sim_rvex
check_asic
comp_asic
sim_asic
sim_asic_saif
syn
comp_syn
sim_syn
par
comp_par
sim_par
sim_par_saif
```

Open Encounter (for example with `par_nolog`), open the powerplan (for example with `source ../DB/{design}_pplan.enc`). Run early rail analysis using `source ../BIN/rail_analysis.tcl`. Optionally, a DRC can be peformed here (using the GUI).

Open Encounter and open the final design (`source ../DB/{design}_fixed.enc`) and run both rail analysis using `source ../BIN/rail_analysis.tcl` and power analysis using `source ../BIN/power_analysis.tcl`.

```
cal_drc
cal_lvs
```

E

Auxiliary scripts

A lot of tasks within the project are very repetitive or time consuming. To relieve the designer from this work, a few auxiliary scripts have been created and added to the project. The scripts used to achieve some of the results discussed in this work are added here.

The reference extraction, in Appendix E.1, lists every cell in the hierarchy of the design. It creates a summary of the accumulated cell area for each component, such that the designer can quickly identify large consumers. Furthermore, for each hierarchical instance, the top 10 largest elements are separately reported. This script was used to allow the use of a smaller prototyping chip area and reduce the manufacturing costs. The script allowed prediction of the impact of different ρ-VEX configurations, as discussed in Section 6.3.

The Power Grid Library is required for (**G**.5a) and (**G**.5b) and discussed in Section 9.4. To generate it, several SPICE files as well as quite a bit of configuration is necessary. Furthermore, due to a small bug in the software, the antenna cells have to be removed from the circuit libraries. To get to the correct configuration consisted of a lot of trial and error. To prevent a future designer from having to do all this work again, all steps are collected in the script as shown in Appendix E.2. The script automatically generates the library.

The final script, in Appendix E.3, is used to run the many different simulations as presented in Chapter 10. By running more simulations, the functional test coverage is improved. As each of these simulations can take a day to complete, it would require a lot of interaction from the designer. By using this automated batch script, the designer is relieved from the simple task of starting up the many different simulations and listing the results.

E.1. Reference extraction

```
1  #!/usr/bin/tclsh
2
3  # High hitter percentage of maximum
4  set highHitter            0.80
5  set highCount             10
6  # List of designs and cells to ignore when listing
7  set ignore                {SZKA65_128X32X1CM2 SZKA65_256X32X1CM2 IVDDIO
       ↪ IVSSIO IVDD IVSS}
8
9  set RPT                   "$::env(PROJ)/SYN/RPT/$::env(TOPLEVEL)$::env(OPT)
       ↪ _mapped"
10 set REFERENCE_FILE        "${RPT}/references.rpt"
11
12 set regexToplevel         {-{77}(.*?)-{77}}
13 set regexDesigns          {Design: (\S+)[\s]*\n[^\n]*\n[^\n]*\n-{77}([^-]*)-
       ↪ {77}}
```

```
14 set regexCellReferences {(\S+)[ ]+?\S+\s+?(\d+\.\d+)\s+?(\d+)\s+?\d+\.\d
   ↪ +[^\n]*?\n}
15 set regexDsgnReferences {^(\S+)\s+?(\d+\.\d+)\s+?(\d+)\s+?\d+\.\d+[^\n]*?$
   ↪ }
16
17
18 set fp [open $REFERENCE_FILE r]
19 set referenceContent [read $fp]
20 close $fp
21
22 set toplevel [regexp -inline -- $regexToplevel $referenceContent]
23
24 if {[string length $toplevel] == 0} {
25     puts "ERROR: File did not contain any reference information, or is
       ↪ formatted differently..."
26     return
27 }
28 set topInner [lindex $toplevel 1]
29
30
31 proc reformat {instanceList} {
32     foreach {match instance size count} $instanceList {
33         set tot [expr $size*$count]
34         lappend reformatList [list $instance $size $count $tot]
35     }
36     return $reformatList
37 }
38
39 proc extractReferences {inner dir designs} {
40     variable regexCellReferences
41     variable regexDsgnReferences
42     variable highHitter
43     variable highCount
44     variable ignore
45
46     # Extract all default library cell information
47     set cellResult [regexp -all -inline -- $regexCellReferences $inner]
48     # Extract all custom design components (next hierarchy level)
49     set dsgnResult [regexp -all -inline -lineanchor --
       ↪ $regexDsgnReferences $inner]
50     foreach {match design size count} $dsgnResult {
51         # Find design component in full list, to extract inner
52         set designInner [findDesignInner $design $designs]
53         if {[string length $designInner] == 0} {
54             puts "ERROR: design reference fault, could not find $design"
55             exit
56         }
57         # Recursively apply this function to inner design, traversing the
           ↪ hierarchy
58         file mkdir $dir/$design
59         extractReferences $designInner $dir/$design $designs
60     }
61
62     # Create a newly formatted list, easier to work with
63     # Also removes unwanted results
64     set allResult [reformat [concat $cellResult $dsgnResult]]
```

```
65
66      # Make a summary of all components in this design
67      set maxSize 0
68      set fp [open "$dir/summary.csv" w]
69      puts $fp "Instance\tCell␣area␣\[um^2\]\tCount"
70      foreach result $allResult {
71          puts $fp "[lindex␣$result␣0]\t[lindex␣$result␣1]\t[lindex␣$result␣
                ↪ 2]"
72          if {[lindex $result 0] in $ignore} {continue}
73          set maxSize [expr max($maxSize,[lindex $result 3])]
74      }
75      close $fp
76
77      # Sort component to get high hitters on top
78      set allResult [lsort -real -decreasing -index 3 $allResult]
79
80      # Make a separate summary of all 'high hitter' components
81      set i 0
82      set fp [open "$dir/high-hitter.csv" w]
83      puts $fp "Instance\tCell␣area␣\[um^2\]\tCount"
84      foreach result $allResult {
85          if {[lindex $result 0] in $ignore} {continue}
86          # List the $highCount largest elements, or $highHitter percentage
                ↪ of maximum
87          if {[expr $i < $highCount] || [expr [lindex $result 3] > [expr
                ↪ $maxSize*$highHitter]]} {
88              puts $fp "[lindex␣$result␣0]\t[lindex␣$result␣1]\t[lindex␣
                    ↪ $result␣2]"
89          }
90          incr i
91      }
92      close $fp
93  }
94
95  proc findDesignInner {search designs} {
96      foreach {match design designInner} $designs {
97          if {[string equal $search $design]} {
98              return $designInner
99          }
100     }
101     return ""
102 }
103
104
105
106 set refDir "$RPT/references"
107 file delete -force $refDir
108 file mkdir $refDir
109
110
111 set designs [regexp -all -inline -- $regexDesigns $referenceContent]
112 puts "Desings"
113 foreach {match design designInner} $designs {
114     puts "Design␣$design"
115 }
116
```

```
117 # Start with extracting toplevel, recursively extract all
118 extractReferences $topInner $refDir $designs
```

E.2. Automated Power Grid Library generation

```
1
2  ## Created at : Fri Jun 23 2017
3  ## Created by : Lennart van Bremen
4
5
6
7
8  #
       ↪ ------------------------------------------------------------------------
       ↪
9  # Initialize libraries in batch mode (which is highly recommended...)
10 #
       ↪ ------------------------------------------------------------------------
       ↪
11 # If the initialization has not been run yet, do it now.
12 # This allows the optimization scripts to be run standalone or during any
13 # regular PAR flow.
14 set init 0
15 if ![info exists PAR_INITIALIZED] {
16     set init 1
17 } else {
18     if !${PAR_INITIALIZED} {
19         set init 1
20     }
21 }
22 if ${init} {
23     source $::env(PROJ)/PAR/BIN/1-init.tcl
24 }
25
26
27 #
       ↪ ------------------------------------------------------------------------
       ↪
28 # Configure PGL library generation
29 #
       ↪ ------------------------------------------------------------------------
       ↪
30
31 # One of tt, ss, ff
32 # For most devices, there are more options (sf, fs etc.), however not for
       ↪ all.
33 # Limit to these three to keep it relatively simple...
34 if ![info exists corner] {
35     set corner tt
36     # One of typical, rcmin, rcmax, cmin, cmax
37     set qrcCorner typical
38 }
39
40 printHeader "PGL generation: configuring SPICE simulation for corner ${
       ↪ corner}"
```

```
41
42 # The spice corners select which devices actually get imported from the
       ↪ models,
43 # this setup should be enough (reading more slows down simulation)
44 set spiceCorners " \
45 ␣␣␣␣␣␣␣␣${corner}_tn_ll_rvt12 \
46 ␣␣␣␣␣␣␣␣${corner}_tn_ll_hvt12 \
47 ␣␣␣␣␣␣␣␣${corner}_tn_ll_lvt12 \
48 ␣␣␣␣␣␣␣␣${corner}_ll_diode \
49 ␣␣␣␣␣␣␣␣${corner}_ll_res \
50 ␣␣␣␣␣␣␣␣${corner}_ll_io25od33 \
51 ␣␣␣␣"
52
53 # Define our SPICE setup
54 # Spice models contain all devices necessary for simulation. Correct
       ↪ devices
55 # are selected using the spice corners.
56 set spiceModels "$::env(DK_BASE)/Models/BB/l65ll_v181.lib"
57 # The subcircuits define our standard cell library components, in spice
58 # netlist format.
59 set coreSubcktFiles "$::env(DK_IP)/cir/uk65lscllmvbbr.cir \
60 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣$::env(DK_IP)/cir/uk65lscllmvbbh.cir \
61 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣$::env(DK_IP)/cir/uk65lscllmvbbl.cir"
62
63 set padSubcktFiles  "$::env(DK_IP)/cir/${::padLib}.cir"
64
65
66 # Ensure our PGL folder exists
67 file mkdir $::env(PROJ)/PAR/PGL
68
69
70 # # Setup cells for macro library generation
71 # set cellList [getIPCells ${::allLibs}]
72 # set cellListFile $::env(PROJ)/PAR/PGL/stdcells.list
73
74 # # Write all cells to a file, one cell per line. Not strictly necessary,
       ↪ but
75 # # makes reading somewhat easier.
76 # set cellList [string map {" " "\r\n"} ${cellList}]
77 # set fp [open ${cellListFile} "w"]
78 # puts ${fp} ${cellList}
79 # close ${fp}
80
81 # The antenna cells give some problems with spice simulation, some device
       ↪ seems
82 # to not be available. Remove it from the subcicuit file. (We could remove
       ↪  it
83 # from the simulated list, however even at reading Voltus crashes...)
84 set coreSubckt ""
85 foreach coreSubcktFile ${coreSubcktFiles} {
86     set coreSubcktFileNew $::env(PROJ)/PAR/PGL/[file tail ${coreSubcktFile
          ↪ }]
87
88     set fp [open ${coreSubcktFile} "r"]
89     set coreSubcktData [read ${fp}]
90     close ${fp}
```

```
91
92      # Find the first occurence of an antenna cell
93      set antennaStart [string first ".subckt ANT" ${coreSubcktData}]
94      # If there is no antenna cell, we can simply use the original file
95      if {${antennaStart} == -1} {
96          lappend coreSubckt ${coreSubcktFile}
97          continue
98      }
99      # Find the next subcircuit that does not belong to the antenna cell
100     set antennaEndMatch [regexp -inline -start ${antennaStart}+1 -indices
            ↪ {\.subckt [^P]} ${coreSubcktData}]
101     set antennaEnd [lindex [lindex ${antennaEndMatch} 0] 0]
102
103     # And remove the antenna data
104     set coreSubcktData [string replace ${coreSubcktData} ${antennaStart} $
            ↪ {antennaEnd}-1]
105
106     # Write back the result
107     set fp [open ${coreSubcktFileNew} "w"]
108     puts ${fp} ${coreSubcktData}
109     close ${fp}
110
111     lappend coreSubckt ${coreSubcktFileNew}
112 }
113
114
115 #
        ↪ ------------------------------------------------------------------------
        ↪
116 # Generate standard cell PGL library
117 #
        ↪ ------------------------------------------------------------------------
        ↪
118 printHeader "PGL generation: generating standard cell library for corner $
        ↪ {corner}"
119
120 # All (automatically recognized) power pins
121 # VDDA12 VDDA25 VCC VDDACI VDDANAI VDD VDDANA VDDIO PWR VDDANAC VBP
122 # All (automatically recognized) ground pins
123 # GNDA GND VSSACI VSSANAI VSS VSSANA VSSIO G0 G1 VSSANAC VBN
124
125 # Input all our settings for the generation of a standard cell PGL
126 # Start with setting up for core cells
127 set_pg_library_mode -celltype stdcells -current_distribution propagation \
128                     -filler_cells [getLibCellFillerEmpty] -decap_cells [
                            ↪ getLibCellFillerDecap] \
129                     -spice_models ${spiceModels} \
130                     -spice_subckts ${coreSubckt} \
131                     -spice_corners ${spiceCorners} \
132                     -extraction_tech_file $::env(PAR_QRC)/${qrcCorner}/
                            ↪ qrcTechFile \
133                     -power_pins {VDD 1.2 VCC 1.2 VDDIO_E 2.5 VDDIO_W 2.5
                            ↪ PWR 2.5 VDDA12 1.2 VDDA25 2.5 VDDACI 1.2 VDDANAI
                            ↪  2.5 VDDANA_N 2.5 VDDANA_S 2.5 VDDANAC 1.2} \
134                     -ground_pins {VSS GND VSSIO_E VSSIO_W G0 G1 GNDA
                            ↪ VSSACI VSSANAI VSSANA_N VSSANA_S VSSANAC} \
```

```
135                     -bulk_power_pins {VBP 1.2} -bulk_ground_pins VBN \
136                     -temperature 25
137
138 # And generate the core cell library
139 generate_pg_library -output $::env(PROJ)/PAR/PGL -library_prefix TT_core
140
141
142 # Set up for currently active pad cells
143 set_pg_library_mode -celltype stdcells -current_distribution propagation \
144                     -filler_cells [getLibCellFillerEmpty] -decap_cells [
                        ↪ getLibCellFillerDecap] \
145                     -spice_models ${spiceModels} \
146                     -spice_subckts ${padSubcktFiles} \
147                     -spice_corners ${spiceCorners} \
148                     -extraction_tech_file $::env(PAR_QRC)/${qrcCorner}/
                        ↪ qrcTechFile \
149                     -power_pins {VDD 1.2 VCC 1.2 VDDIO_E 2.5 VDDIO_W 2.5
                        ↪ PWR 2.5 VDDA12 1.2 VDDA25 2.5 VDDACI 1.2 VDDANAI
                        ↪  2.5 VDDANA_N 2.5 VDDANA_S 2.5 VDDANAC 1.2} \
150                     -ground_pins {VSS GND VSSIO_E VSSIO_W G0 G1 GNDA
                        ↪ VSSACI VSSANAI VSSANA_N VSSANA_S VSSANAC} \
151                     -bulk_power_pins {VBP 1.2} -bulk_ground_pins VBN \
152                     -temperature 25
153
154 # And generate the core cell library
155 generate_pg_library -output $::env(PROJ)/PAR/PGL -library_prefix TT_${
    ↪ ::padLib}
156
157 #
    ↪ ------------------------------------------------------------------------
    ↪
158 # Generate macro cell PGL libraries
159 #
    ↪ ------------------------------------------------------------------------
    ↪
160 # GDS to PGL does not seem to work...
161 # printHeader "PGL generation: generating macro cell libraries for corner
    ↪ ${corner}"
162
163 # Input all our settings for the generation of macro cell PGL
164 # set_pg_library_mode -celltype macros -current_distribution propagation \
165                     # -cell_list_file ${cellListFile} \
166                     # -gds_files [getExtendedFiles ${::allLibs} ".gds"] \
167                     # -gds_layermap "../PGL/gds.layermap" \
168                     # -spice_models ${spiceModels} \
169                     # -spice_corners ${spiceCorners} \
170                     # -extraction_tech_file $::env(PAR_QRC)/${qrcCorner}/
                        ↪ qrcTechFile \
171                     # -power_pins {VDD 1.2 VCC 1.2 VDDANAC 1.2 VDDANA 2.5
                        ↪ VDDA12 1.2 VDDA25 2.5} -ground_pins {VSS GND
                        ↪ VSSANAC VSSANA GNDA} \
172                     # -temperature 25
173
174 # And generate the libraries (one for each macro)
175 # generate_pg_library -output $::env(PROJ)/PAR/PGL -library_prefix TT
```

E.3. Batch simulation

```
#!/usr/bin/tclsh

# Not being in the correct project directory prevents correct configuration...
if ![string equal [pwd] $::env(PROJ)] {
    puts "ERROR: This script must be run from the base project directory."
    puts "If not, modelsim.ini variables are set incorrectly, causing trouble in other programs."
    puts "Aborting batch simulation..."
    exit
}

# Start with sourcing the project wide common functions and variables.
source $::env(PROJ)/BIN/common.tcl


# Configure stdout to immediately output, instead of line-buffering. Together
# with the switch >&@stdout of the exec command, this can be used to
# interactively run separate commands within this tcl script.
chan configure stdout -buffering none


# Configure which tests we want to run
set configurations  {par}
set laneConfigs     {0x0000 0x8880}
set examples        {ucbqsort compress convolution}

foreach configuration ${configurations} {
    printHeader "Compiling design configuration ${configuration}"
    # First make sure we compiled the correct design
    exec >&@stdout $::env(PROJ)/SIM/BIN/compile_${configuration}.tcl
    foreach laneConfig ${laneConfigs} {
        printHeader "Compiling for lane configuration ${laneConfig}"
        # Lane config is a separate compile option, we need to clean the
        # directory with every change.
        cd $::env(PROJ)/IP/rvex-asic/platform/asic-hsi-test/examples
        exec >&@stdout make clean
        cd $::env(PROJ)

        foreach example ${examples} {
            set resultFile $::env(PROJ)/SIM/LOG/${configuration}/${example}_${laneConfig}way.transcript
            # Skip simulations that are already complete, to save time...
            if [file exists ${resultFile}] {
               continue;
            }
            # Compile the current example design
            printHeader "Compiling test ${example} with lane configuration ${laneConfig}"
            cd $::env(PROJ)/IP/rvex-asic/platform/asic-hsi-test/examples
            catch {
                exec >&@stdout make ${example}.srec DEFS=LANE_CONFIG=${laneConfig}
```

```
49              }
50              # Sometimes we have to run twice...
51              exec >&@stdout make ${example}.srec DEFS=LANE_CONFIG=${
                    ↪ laneConfig}
52              cd $::env(PROJ)
53
54              # And simulate it in batch mode
55              set ::env(SIM_EXAMPLE) ${example}
56              printHeader "Running␣test␣${example}␣with␣lane␣configuration␣$
                    ↪ {laneConfig}␣in␣mode␣${configuration}"
57              exec vsim << "
58  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣do␣$::env(PROJ)/SIM/BIN/run_${configuration}.do
59  ␣␣␣␣␣␣␣␣␣␣␣␣␣␣quit␣-f"
60
61              putslog "Simulation␣finished!␣Moving␣files..."
62
63              # # Copy the result to a 'unique' name (unique per iteration)
64              # file rename -force $::env(PROJ)/SIM/LOG/${configuration}/
                    ↪ vsim.wlf    $::env(PROJ)/SIM/LOG/${configuration}/${
                    ↪ example}_${laneConfig}way.wlf
65              # The wlf is large and/or somewhat useless, delete it
                    ↪ instead...
66              file delete $::env(PROJ)/SIM/LOG/${configuration}/vsim.wlf
67              file rename -force $::env(PROJ)/SIM/LOG/${configuration}/
                    ↪ transcript ${resultFile}
68
69              putslog "Simulation␣results␣complete"
70          }
71      }
72  }
73
74  # Restore simulation variable to its default
75  set ::env(SIM_EXAMPLE) "sim"
```

Author

Lennart van Bremen was born in Delft, The Netherlands, on January 31, 1993. Starting at a young age, he has always been passionate about software and electronics. All his programming knowledge has been self-taught from resources available on the internet. This knowledge is backed up by countless pieces of software and small games, still in use.

At the age of 17, his eager learning drive and interests in software and hardware allowed him to participate in the FIRST Tech Challenge, as part of a nine-headed team. The team constructed a robot that was able to perform several user-guided tasks, competing other teams in two on two matches. With very little help, he was able to program an AI capable of outperforming all other Dutch teams by a long shot. During the international competition organized in the Centennial Olympic Stadium in Atlanta, Georgia, the team battled through three days of matches. They missed the finals by just a few points, while still leaving many competitors behind.

In 2011, he continued applying his software knowledge with a job. He collaborated in the development of several websites as well as integrated solutions for other companies, including the Royal Dutch Airlines. In the year 2015 he quit his job to put all his effort in finishing his education.

In the same year, he acquired his B.Sc. in electrical engineering at the Delft University of Technology, with a minor in Artificial Intelligence. With this document, he strives to acquire his M.Sc. as well, concluding his study program with a collaboration project between the Computer Engineering and Circuits and Systems departments of the Delft University of Technology. After his study, he is ready to move to Eindhoven, where he has signed a contract with ProDrive to start working.