FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Visa Seppänen

PARTIALLY RECONFIGURABLE SDR SOLUTION ON FPGA

Master's Thesis
Degree Programme in Computer Science and Engineering
August 2019

Seppänen V. (2019) Partially reconfigurable SDR solution on FPGA. University of Oulu, Degree Programme in Computer Science and Engineering. Master's Thesis, 46 p.

ABSTRACT

Software-defined radios (SDR) have become more common in order to answer the increasing complexity of wireless communication standards. The flexibility offered by SDR technology in return makes it possible to create and implement even more complex standards so there exists a mutual evolution cycle. One of the technological opportunities pursued on SDR is changing the waveforms on the fly.

The standards challenge the SDR development. Computing throughput needs to be high enough, the end product has to be energy efficient, and all of this must be accomplished as cheaply as possible.

SDRs have a wide range of implementation opportunities from complete software designs to more hardware oriented with higher level software control. The extreme ends of these approaches suffer from energy dissipation and design cost issues, respectively. The compromises include application specific architectures and reconfigurable hardware. Solutions vary from software to hardware between cases and depending on the needs. This thesis concentrates on investigating partial reconfigurability on a field-programmable gate array (FPGA) in an SDR application.

Based on the results, partial reconfigurability is an attractive mean to bolster SDR functionalities. Although the energy efficiency of the employed FPGA solution is inferior to using an application-specific integrated circuit (ASIC), the flexibility and cost of design set them apart. This study focuses on partial reconfiguration on Xilinx FPGA devices but it may show benefits for other devices that can utilize partial reconfiguration on their designs.

Keywords: software-defined radio, partial reconfiguration, field-programmable gate array, waveform

Seppänen V. (2019) Osittain uudelleenohjelmoitava ohjelmistoradio FPGA-piirillä. Oulun yliopisto, tietotekniikan tutkinto-ohjelma. Diplomityö, 46 s.

TIIVISTELMÄ

Ohjelmistoradiot ovat yleistyneet entistä kehittyneempien langattomien kommunikointimenetelmien myötä ja tarpeesta vastata näiden vaatimuksiin. Samalla ohjelmistoradioiden joustavuus mahdollistaa uusien ja kompleksisempien standardien kehittämisen. Tätä voi pitää molemminpuolisena kehityssyklinä. Aaltomuotojen nopea vaihtaminen lennosta ohjelmistoradion ollessa käytössä on yksi kehityksen alla oleva teknologia.

Kommunikointistandardit haastavat ohjelmistoradioiden kehityksen erilaisilla vaatimuksillaan. Esimerkiksi laskentatehon tulee olla korkea, lopputuotteen energiatehokas ja tämän tulee tapahtua mahdollisimman edullisesti.

Ohjelmistoradioiden toteutukset vaihtelevat aina vahvoista ohjelmistopohjaisista arkkitehtuureista enemmän laitteistoon tukeutuviin versioihin. Ääripäissä tässä spektrissä ohjelmistoihin perustuvat toteutukset eivät ole riittävän energiatehokkaita ja laitteistoratkaisujen hinnat nousevat helposti korkealle. Keskitien ratkaisuja ovat sovelluskohtaiset arkkitehtuurit ja uudelleen ohjelmoitavat laitteistot. Implementaatiot vaihtelevat ohjelmisto-laitteisto skaalalla riippuen tarpeesta ja tilanteesta. Tämä opinnäytetyö keskittyy tutkimaan osittaista uudelleenohjelmoimista FPGA-piireillä ohjelmistoradion yhteydessä.

Tulosten perusteella osittainen uudelleen ohjelmointi on houkutteleva tapa tehostaa ohjelmistoradioita. Vaikka FPGA-piirien energiatehokkuus ei ole yhtä hyvä kuin ASIC-toteutusten, niiden joustavuus ja suunnittelukustannukset ovat paremmat. Vaikka tämä työ keskittyy osittaiseen uudelleenohjelmointiin Xilinxin FPGA-piireillä, voi siitä olla hyötyä muissa tutkimuksissa ja laitteissa.

Avainsanat: ohjelmistoradio, osittainen uudelleenohjelmointi, FPGA, aaltomuoto

TABLE OF CONTENTS

FOREWORD

This thesis was made to research the partial reconfiguration of Xilinx FPGAs with SDRs by the request of Bittium Wireless Oy. The work was supervised by Prof. Olli Silvén from the University of Oulu with Dr. Konstanting Mikhaylov acting as the second examiner, and by Jussi Liedes from Bittium Wireless Oy.

The technology of FPGAs and partial reconfiguration was very interesting and researching it opened my eyes to more opportunities on this field. The potential harmony between software and hardware can unlock many doors in the future even though the society drifts ever so slightly more and more on the shoulders of software. Nothing however can work without hardware and it is crucial to research its capabilities to create even more efficient applications that utilize the full potential of the hardware design. It is hard to distinguish a specific line between these two platforms or to juggle between them, but I love standing in that crossroads.

I would like to express my gratitude to my supervisors, who gave me such a superb feedback on my work. There were disagreements, but mostly these comments kept me pushing for the better. Special thanks go to my friends and closest ones, who helped with ideas and gave support when I needed it.

Oulu, 5.8.2019

Visa Seppänen

ABBREVIATIONS

ASIC	application-specific integrated circuit
ASIP	application-specific instruction-set processor
CR	cognitive radio
DSP	digital signal processor
FFT	fast Fourier transform
FPGA	field-programmable gate array
GPP	general-purpose processor
GPU	graphical processing unit
ICAP	internal configuration access port
ILA	integrated logical analyzer
IOB	input/output block
IP	intellectual property
ISE	integrated synthesis environment
LUT	lookup table
OFDM	orthogonal frequency-division multiplexing
PHY	physical layer
PRR	partially reconfigurable region
PSK	phase shift keying
QAM	quadrature amplitude modulation
RM	reconfigurable module
RTL	register transfer level
RX	receiver
SDR	software-defined radio
SODA	signal-processing on-demand architecture
TX	transmitter

1. INTRODUCTION

Wireless communication is the backbone of modern technology, which is always under intense development and research. More and more complex communication standards are being developed in order to increase overall throughput and reliability. This can be seen from the increasing number of specifications and reports over time as Figure 1 shows. These next generation standards demand greater performance from the devices that were suitable for the earlier generations. New technologies must be created in a fast pace in order to answer this growing demand. Radios that implement these standards must be able to work with multiple life cycles of those standards. Reworking every device's hardware that utilizes radio technology between every new finding in wireless communication is not a feasible method. With SDRs, updating becomes straightforward and flexible.

SDR is a radio type which has part of its functionalities realized via software instead of hardware. The development of SDR goes back to 1970's and 80's when the radio known as SpeakEasy was made, but Joseph Mitola III who started working on SDR in 1990's is considered as the father of this technology [1]. SDRs gave access to new possibilities with their flexibility and capabilities compared to their hardware based counterparts.

The most notable features in SDRs are interoperability and quick configuration for needed task. With SDRs, communication between different systems became much more straightforward and if some technology were not supported, it could be easily added to SDR's repertoire [1]. The ability to reconfigure existing SDRs with new waveforms and solutions also prolongs the lifetime of SDR equipment, which has especially monetary value [2].

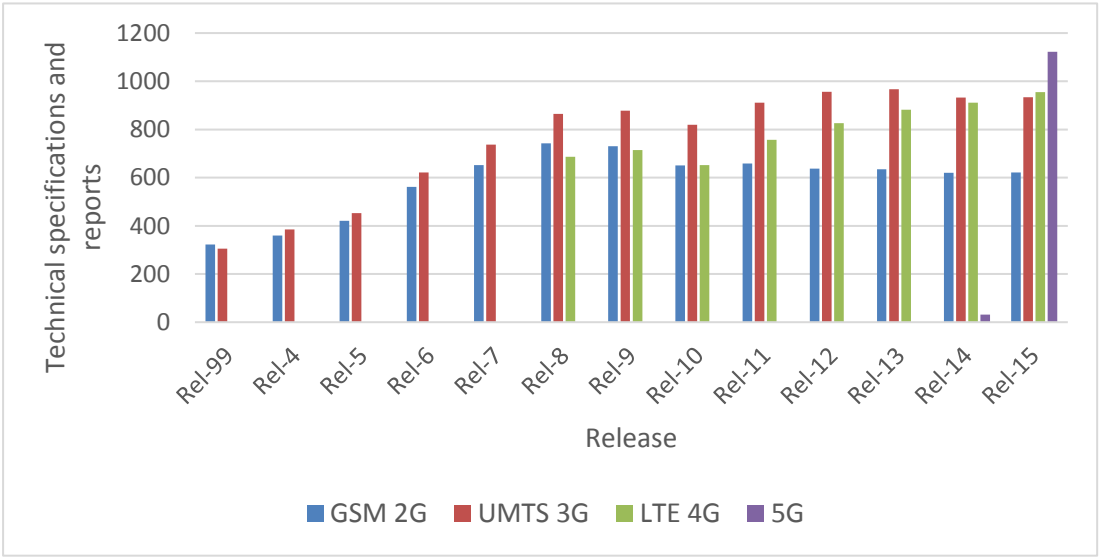

Figure 1. Difference in the number of specifications and reports between different wireless communication generations provided by 3GPP.

Radios based on hardware solutions do not have the same flexibility that SDRs offer. The current fast development of communication technology requires fast adaptation to new standards and the ability to support multiple different

communication standards on the same equipment. SDRs though can also lack behind in this pace if their reconfigurability is not improved and efficiency increased. Different devices like FPGAs are reconfigurable, but they can be optimized by introducing other technologies and solutions on them just like overclocking a processor on a laptop to enhance its performance.

SDR is not a new invention, but evolving computing power in technology makes the SDR implementations viable in commercial products. Development around SDRs focuses on different aspects of this technology in order to create the best possible end product. In [3] the focus is on lowering power consumption while still maintaining the advantages of software implementations like support for multiple different protocols. On the other hand [4] shows advantages on combining different architectural approaches in order to create a high-performance SDR where those approaches support one another.

Balancing between different attributes of a device and finding the right solutions to satisfy each need that have been established is a key to a successful product. This is also true for SDR. The attributes like size or power consumption cannot be considered as independent variables but always affect each other, for example increasing computational power via adding hardware increases the size and bill of materials.

There are different ways to realize SDRs utilizing hardware and software with their distinct approaches to different issues. In the past ASIC was the main hardware technology but has since been replaced by FPGAs with their flexibility, which is a main driving force for SDR [5]. FPGAs are a feasible platform for these kinds of applications, where the FPGA includes none or little hardware solutions for a radio but offers flexible solutions for a software implementation of such.

Different architectures have different advantages and combining these into hybrid solutions can bring the best attributes forth but in the meantime raise complexity and costs. General-purpose processors (GPP), application-specific instruction-set processors (ASIP), and hardware acceleration all have their pros and cons. Combining these and introducing new techniques can mitigate the drawbacks while maintaining high performance. Reconfiguring, would it be fine- or coarse-grained, for example makes it possible to reutilize same resources for different tasks and this way reduce size and power consumption.

Software reconfiguration is normally perceived as slow full reconfiguration. There are however ways to enhance this process like speeding up the configuration process or reconfiguring only parts of the device. There are multiple ways to implement these enhancements and the architecture may support some of those better than others. In the product considered in this thesis, there was a need for a fast waveform swapping without disturbing the rest of SDR's functionalities running on the same FPGA device. Partial reconfiguration, for instance, holds a lot of potential in this regard. The main focus is to research if with our given equipment an SDR system with partially reconfigurable waveforms is possible. If this functionality is viable, then it will be implemented to the product.

The advantages and disadvantages of partial reconfiguration and comparison between other possible approaches shall be discussed before the possible implementation. If this technology proves to be beneficial to the product, other uses than reconfiguring just waveforms may also be researched.

1.1. Structure of the thesis

This thesis is structured as follows. Chapter 2 goes over general SDR information and compares few different approaches. FPGA reconfiguration is examined in more detail in Chapter 3, and from those methods Chapter 4 focuses on partial reconfiguration. Chapter 5 presents the implementation of partial reconfiguration on Xilinx FPGA. The results and hardships are discussed in Chapter 6. Chapter 7 wraps all up and looks into some future work.

2. SOFTWARE-DEFINED RADIOS

Radio technology is the base of all wireless data communication. Advancements and development in this field have been very important for reliable and fast communication all over the world.

SDR architecture can be separated into hardware and software sections where the dividers are the analog-to-digital and digital-to-analog converters as can be seen in Figure 2. On the analog side, computing has been realized by hardware and on the digital side processes are done by software. Data flows through these parts in both ways, which are divided into transmitter (TX) and receiver (RX) components [6]. How these parts are realized is dependent on individual projects and their developers. Figure 3 shows a little bit more detailed block design of the architecture used in the context of this thesis.

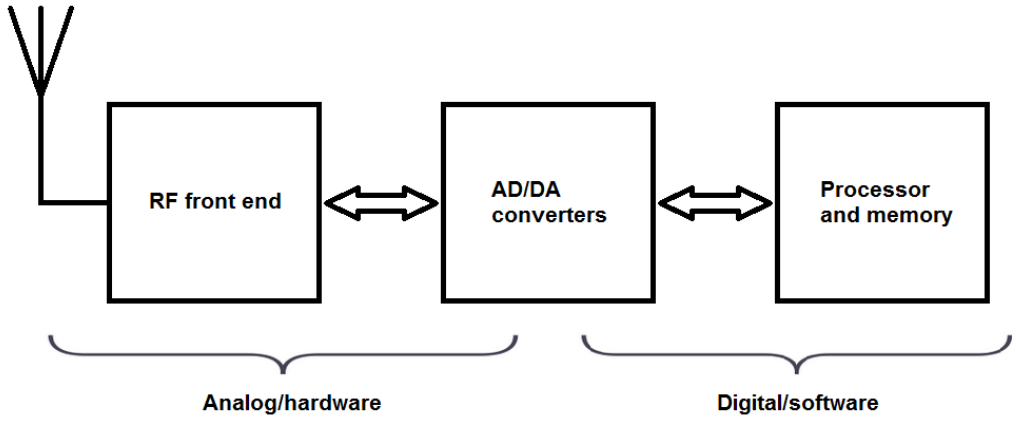

Figure 2. Simple SDR architecture.

From a financial standpoint, manufacturing hardware for different standards is a very high cost endeavor. With SDR, only one type of hardware is needed since the different standards can be implemented through software. Development is also made easier on SDR platform by just reprogramming the system according to requirements [1]. Of course, these advances can be achieved only if there are multiple different configurations, waveforms, and modulations using this software. Radio relying on hardware solutions that has been tailored to one specific usage can still out perform and be more cost-efficient than an SDR, which also has been tailored to only one configuration. The advances of an SDR system rely heavily on its flexibility and agile development of multiple standards.

The changing of communication standards and how it happens can be considered as one of the main variables when comparing SDRs with each other [2]. On one end are those that take long time to load a new waveform algorithm and disturb the functionalities of the SDR while doing so. On the other end waveforms are loaded instantaneously and there are little or no downtime caused by this. Also the amount of different modulations and software that the SDRs can support will set them apart from each other.

Reconfiguring used platform with new software to work with different standards takes away the need to replace hardware in order to update outdated equipment.

Loading new software into, for example, multiple base stations is much faster and cheaper than changing each of them with new hardware. Fast reconfigurability of the SDR creates also the possibility to intercept other wireless communication because its modulation and frequency can be changed while searching the right configuration. This is especially beneficial for military use.

Software-defined and all radios in general can be stationary or portable depending on the use. Stationary ones do not have such strict specifications like size or power usage that portable devices do have. Mobility is very important especially in military use where troops might move long distances without logistics. Two most important factors that have impact on mobility with SDR are size and power usage. Smaller devices tend to weigh less so they are easier to carry, and low power usage gives longer uptime and less need of a recharge. These two attributes also go hand in hand since they are both mainly affected by the amount of used hardware. With less needed hardware and resources, the physical size of the device decreases. There would also be fewer components to power up which lowers power consumption.

There are many ways to decrease the amount of used hardware from the final product for example cutting functionalities from the design and favoring software solutions over hardware ones. Providing fewer functions is however not optimal since, as stated earlier, SDR excels when it is flexible and agile and cutting functionalities would hinder these attributes. This project is also long in its development so direct changes to hardware would be costly. This basically leaves utilizing different solutions on the already established platform. Some of these could provide answers to lowering hardware usage, enable fast waveform switching, and also bring other benefits.

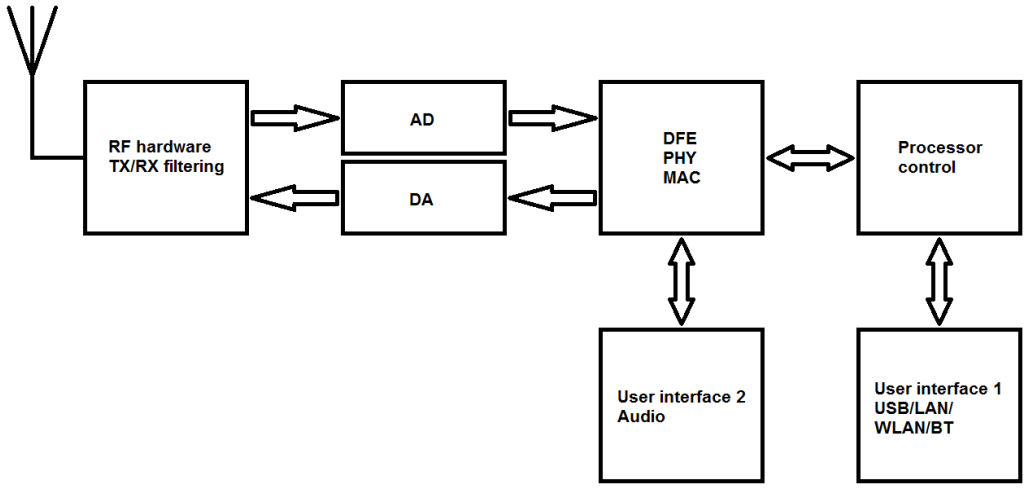

Figure 3. Simplified block design of the SDR under development.

2.1. Different SDR architectures

Since SDR is a major factor in communication technology, it is only logical that advancements on this field are developed all the time. Any improvements that can help, for instance, the performance, manufacturing, or costs should be looked into and implemented if they are viable ones. This has led into plethora of ways to create

an SDR device. Table 1 presents a few different possible SDR platform technologies and provides simple comparison between them [7].

Table 1. Comparison of SDR design approaches

	Performance	**Flexibility**	**Power consumption**	**Complexity**	**Cost**
GPP	Low	High	High	Low	Low
DSP	Moderate	Moderate	Moderate	Low	Moderate
FPGA	Moderate	High	Low-Moderate	Moderate	Moderate
ASIC	High	Low	Low	Low	Moderate
Hybrid	High	High	Low-Moderate	High	High

The main principle is the same: using software to implement the components that normally would be created via hardware solutions. The means, however, can be drastically different from another. The distribution between hardware and software, techniques to create needed functionalities, additional solutions to enhance the performance and other little things can make SDRs very different from each other. There are arguments for and against each approach, but these devices should be created requirements and preconditions in mind that has been made for the final product. Some techniques may work for one project but not for the other.

Deciding a proper architecture for the SDR product is a fundamental choice that affects every aspect of that product, like the distribution of functionalities on hardware and software. Each approach, be it software- or hardware-heavy solutions, has its advantages and disadvantages. The example architectures show that introducing multiple different solutions to the product and using hybrids of those solutions can bring a plethora of advantages. Figure 4 shows a rough estimate how performance and flexibility could be changed. However, there might also be trade-offs. Not all solutions work together or the chosen architecture does not have support for those techniques. Finding the right balance in everything brings about the best end product. Introducing excessively different methods increases complexity and therefore production time and costs.

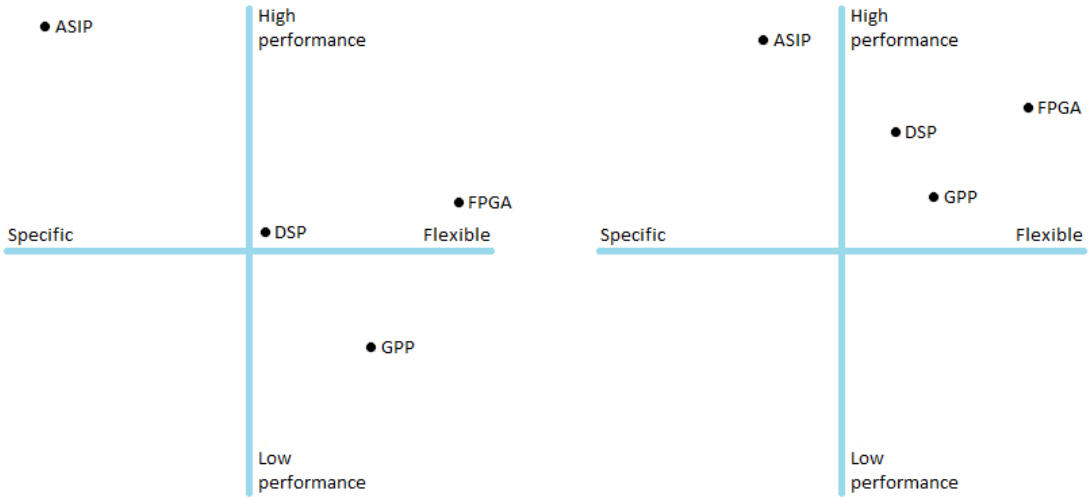

Figure 4. Rough comparison between before (left) and after introducing different methods to boost the capabilities of different platforms.

Comparing different approaches and architectures is not as simple as looking their numbers. Every single one can be configured for special need and performance as well as power efficiency can be optimized through different methods. These technologies are highly adaptive and should be compared with their capabilities in mind. The next four examples are meant to show how the shortcomings could be tackled. In the end, it is the responsibility of the system designers to come up with the best solutions for their needs.

2.1.1. SODA

Signal-processing on-demand architecture (SODA) is a fully programmable architecture that supports SDR. Lin et al. [3] try to accomplish an SDR with low power consumption without sacrificing flexibility or performance with this architecture. SODA is software-heavy and this decision is explained with support for running different protocols, faster time to market, flexible prototyping, and updating even after deployment and higher chip volumes.

SODA has multiple processing elements and uses hybrid combination of different digital signal processor (DSP) architectures in order to achieve the high throughput that wireless communication demands. This is called algorithm-architecture matching where different beneficial methods between algorithms and architectures are combined. It is done in order to accomplish something greater than what these techniques could achieve separately on their own. Matching like this is common in SDRs. Different approaches are used to support each other and reduce the impact from drawbacks.

The processing elements are linked together and with system controller and global memory through shared bus as seen in Figure 5. Idea of using multiple different units for processing could be expanded by adding more of these units. This could also work for waveform switching by having different units for different waveforms and change between them when needed.

There are trade-offs with more complicated solutions than what SODA utilizes. They might bring computational advances but at the same time would increase the complexity and needed hardware. For this reason, these solutions are discarded in this proposed architecture.

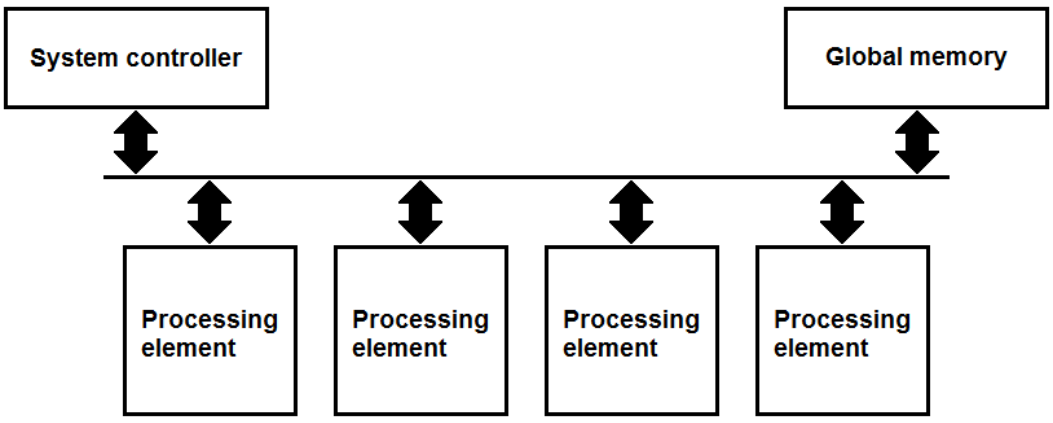

Figure 5. System architecture of SODA.

There are multiple different techniques used in order to achieve the wanted low-power design: exploiting more vector logic on DSP, having fewer ports and clustering memories for processing elements, and using smaller instruction logic. The assembly code is also optimized by hand. This kind of customization does bring all kinds of benefits, in this case lower power consumption and higher performance. However, too specific customization can also undermine flexibility. If the architecture is extremely optimized, it might not support as wide selection of different standards as less optimized one or the support is not as good.

Focusing on software solutions however does give more room for optimization without sacrificing too much flexibility. Hardware techniques could bring same kind of results or even better, but modifying hardware is much more complicated compared to software.

2.1.2. Sora

GPP focused approach is given with an SDR named Sora. GPPs are commonly known as the computer microprocessors. As an advantage over FPGAs and DSPs, which have been the dominant solutions for SDRs, GPPs are more familiar and easier to use from the standpoint of developers. A major drawback however is GPP's limited performance. Sora is designed to tackle this problem and offers a high-performance SDR with a GPP architecture [4].

Wireless communication requires high throughput and precise timing for it to function properly. GPP is not originally designed for this kind of performance. In order to overcome this obstacle, Sora utilizes tailored radio control board, multicore architecture with its multiple features, and dedicated cores for real-time tasks. Communication functionalities rely on software as much as possible in order to achieve a flexible SDR. This makes it possible to use simple and generic hardware designs. Communication between memory and radio control is done with high speed PCIe bus in order to address the high throughput requirements. Figure 6 illustrates Sora's system architecture in a simplified manner.

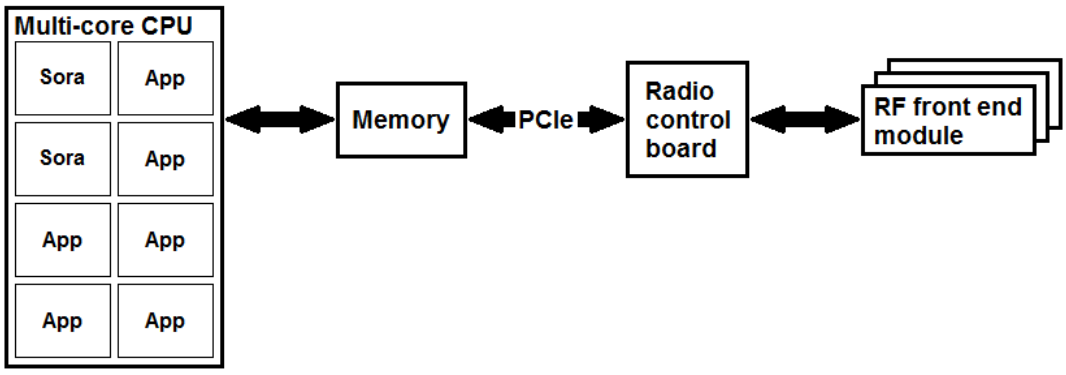

Figure 6. System architecture of Sora.

The custom radio control board has a large memory that can store multiple precalculated waveforms. Different standards have generally predetermined response frames so calculating most of the contents of those frames in advance of transmission is possible. All of this precalculation results in a faster response time, which is quite

complicated to achieve with GPPs, because of internal bus latency. Using radio control board's memory bypasses this latency problem.

Sora accelerates the capabilities of its GPPs even further to meet the timing and throughput requirements. This is done via lookup tables (LUT), data-parallelism, partitioning processing over multiple cores, and dedicating cores for specific tasks like software radio tasks.

This architecture shows that GPPs are also capable to work as basis for SDR systems when the right techniques are used. In the end, SDR realized this way is comparable to a commercial device.

2.1.3. Hardware accelerated FPGA

SDR solution proposed in [8] is the closest one of these examples to this thesis with the use of FPGA and partial reconfiguration. The main objective of this architecture is to provide building blocks that are flexible and programmable for the ever-changing wireless networks of the future. Reasoning for using FPGA comes from the fact that traditional ASIC architectures are not flexible and GPPs lack the performance. GPPs can be enhanced like [4] shows in order to boost their capabilities but even if coupled with graphical processing units (GPU) they would lack behind in the future.

The proposed architecture is a hybrid model of an FPGA with hardware accelerated functionalities. It creates a flexible platform with the computational capabilities comparable to an ASIC. The architecture includes processing units that are tailored for different tasks and are able to compute these tasks faster than their software counterparts.

Different wireless communication standards require different operations. FPGAs can be easily reconfigured again to compensate other standards, however not everything has to be reconfigured for this. Partial reconfiguration can be applied to SDRs based on this architecture to extend and improve the switching between different processing streams and communication standards.

Controlling the processing units and communication between them can be a challenging task. This with the partial reconfiguration creates an extremely complex platform. Working product with these techniques can be a powerful one, but the complexity creates a difficult development environment.

2.1.4. Reconfigurable ASIP

SDRs are also possible with ASIPs even though they might be considered overly static for this job. Vogt et al. [9] show an instruction level flexible and high performance architecture. Flexibility is accomplished via leaving channel code configuration dynamically reconfigurable, which can load new processes and programs that can compute different communication standards.

Multiple techniques are used to bring various benefits like lowering power consumption and used area while still maintaining great throughput and precise timing.

This architecture shines best with its performance when compared to other solutions but lacks behind with restricted flexibility. This is a common relation

between these two attributes: higher efficiency sacrifices flexibility and vice versa. Introducing different methods to compensate the faults is a typical practice.

2.2. Development of software-defined radios on FPGA

In the past, several ways to create SDRs have been utilized, mainly ASICs, DSPs, GPPs, and FPGAs [10]. Of these the FPGA has been the dominant one for the last years with its configurability. A hybrid solution with DSP further increases the advantages by FPGA and DSP compensating the drawbacks of each other. Before this, DSP was coupled with ASICs in order to achieve same results, but FPGA has since replaced ASICs [5].

FPGAs are quite inefficient when talking about utilizing logic. ASIC solutions are superior in this manner and are also faster and more power efficient. However, the reconfigurability of FPGA sets them apart. There exists a great amount of different standards that SDRs need to support. The technology behind FPGA has introduced an easy way to create new devices and update old ones for these standards. With ASIC, the modern communication solutions would require much more logic area and would rank costs much higher than with FPGA, but this is only true with low volumes. If the intention is to manufacture a lot of ASIC based SDRs, then the costs can be lower than with FPGA in the long run [11]. However, the quantity that these ASIC based systems need to be made in order to get these financial benefits grows all the time. This is due to the increasing computing power of FPGAs and technological advancements.

DSP and GPP perform well with narrowband signals and their inability with wide ones have been tried to improve by adding other solutions into their architecture like hardware acceleration. In the end this has a snowball effect of too complex design which leads to longer development and higher costs. Realizing some of the functions on FPGA or ASIC can share the load off from these processors and boost the capabilities of both.

2.3. Motivation and research

This thesis is targeting an SDR product, where the waveform physical layer (PHY) blocks are located on an FPGA. Higher protocol layers and user interface are located on the ARM processor cores. The FPGA has also other functions and interfaces in addition to the waveform PHYs. There are three different waveforms referenced as A, B, and C, of which A is the most complex and C is the least complex. In addition to that, there exists two variations of the SDR product: one with a smaller FPGA with a single active waveform and one with a larger FPGA with two waveforms simultaneously active. These waveforms are required to be swappable between the earlier mentioned A, B, and C. If the FPGA has two active waveforms, they can be different for example A and B or they can be both the same at any given time.

The main target for this thesis is to enable fast waveform swaps without disturbing the other functionalities of the SDR running on the same FPGA device. In the case of two parallel waveforms, while one of them is being changed, the other one should remain active and operational.

All this could be done by just fitting each waveform parallel on a large FPGA, but this is far from optimal. It would waste resources, increase the physical size and

would actually work against the idea of a flexible SDR on an FPGA. In conclusion this method is not considered as an option. Waveforms should be implemented on the FPGA in some efficient way that should provide the wanted fast switching of waveforms and at the same time could also bring other benefits like efficient resource usage and power consumption. These advantages that will be discussed later in this thesis should support the idea of a flexible and competitive SDR product.

There are ways to configure the FPGA in such a manner that its resources can be utilized in different ways at different times without any direct interference with the configured fabric. The device can change its functionalities by itself depending on the situation like the end user choosing a different mode. The device reconfigures itself according to the new mode and after a fast operation on the FPGA it is ready to be used with this mode without any other action by the user than just turning a knob.

3. RECONFIGURING FPGA

The main advantage of an FPGA is its reconfigurability. The circuit can be fully reconfigured after the initial application has been implemented on it. This removes the application and substitutes it with a new one. Full reconfiguration, however, is not the most efficient way to update the design or develop on an FPGA.

There are different ways to partition the resources on an FPGA for distinct needs and then reconfigure only those units while preserving the other sections that do not need reconfiguring. The main advantage this style brings is less of required resources because different functionalities can occupy the same area on various times when they are needed. These also accumulate into benefits on other fields like cost and power consumption for example. Depending on the method how this reconfigurability is established may also make development and updating faster since only parts of the whole board needs to be worked upon.

This thesis goes through the usage of partial reconfiguration, hardware acceleration, and fine- and coarse-grained logic in order to realize the wanted fast waveform switching and also looks into other possible benefits that these methods might provide. The product is developed on Xilinx Zynq Ultrascale+ FPGA device, which might affect the research. Full reconfiguration is also discussed briefly mainly as a counterpart and benchmark for the other methods.

3.1. Full reconfiguration

FPGAs with their high reconfigurability are an essential part of modern computing hardware. Full reconfiguration of the FPGA device makes it possible to create completely different applications on the same hardware with little effort. However, using only the full reconfiguration of the device is in itself a slow way to work with an FPGA. As the name implies, the full reconfiguration wipes the whole device clean and configures an all new implementation on it. This can be an all-new feature or just a small update on the original application. The time from synthesis all the way to the reprogramming of an FPGA can be more or less the same in both these cases even though the smaller change can be negligible when compared to the full new application.

Full reconfiguration can be considered as the traditional way of working with FPGAs. This method is used as the reference level when comparing the various procedures and their advantages and disadvantages.

3.2. Hardware acceleration

SDR has all or a significant part of it realized as software. Some functions and operations are however not ideal on software and would greatly benefit from hardware solutions. This kind of hardware acceleration could bring wanted computational throughput by optimizing some parts of the SDR by utilizing hardware computation that DSP, FPGA and ASIC can give. This reduces the flexibility that the software solutions have but finding a good balance between hardware and software can make the required fast waveform switching possible through faster computation.

In some cases, SDR might need a lot of computational power to operate on the wanted level. There might exist over 20 different standards that the SDR has to support with a wide variation between those standards from quite simple ones to very complex solutions. Going over these standards fast and loading those in to and out from an active stage asks a considerable amount of computing [12].

The SDR product considered in this thesis is already using FPGA based hardware acceleration to speed up operations and balance performance. Without hardware acceleration the software processes would be under considerable load, which would demand a more powerful processor and increase power consumption. It could be possible to expand hardware acceleration in order to switch waveforms fast enough as the specification demands. With multiple waveforms, however, hardware acceleration does not solve the problem of fitting each waveform on the FPGA and is actually detrimental in this cause. Moving more and more functionality into hardware will of course require more space on the FPGA, which can already get scarce in the case of two active waveforms of type A.

Traditional solution is to create different implementations and bitfiles with every possible combination of waveforms and use full reconfiguration with hardware acceleration to load those on the FPGA when needed. This however does not take away the drawbacks of rebooting and interfering with other parts of the device. In the products under development, the FPGA includes also other functions and services in addition to the waveforms so full reconfiguration would mean a full reboot of the product, which takes time. Bitfiles for full reconfiguration also increases the size of non-volatile memory to fit all those different bitfiles on the device. This case of needed memory is discussed in more detail later in this thesis.

3.3. Fine- and coarse-grained logic

Fine- and coarse-grained logics are not reconfiguring methods, but they do introduce different ways to improve the functionality and potential of an FPGA. In fine-grained logic, the resources are separated into smaller parts and each of them can compute different tasks at the same time. Coarse-grained logic creates larger units from the resources of an FPGA and these units are given defined tasks. For this reason, it is not as flexible as fine-grained with smaller tasks, but coarse-grained outperforms fine-grained logic when the given tasks are more suitable for the larger logic units. Especially if only the output is required and the operations between input and output are more or less meaningless. FPGAs can have both fine- and coarse-grained logic in them [13, 14]. Figure 7 shows how there can be fine-grained logic and various coarse-grained logic blocks on the same fabric and that the coarse-grained blocks are basically composed of fine-grained logic.

Partitioning logic into fine- and coarse-grained can be divided into different levels. Behavioral level partitioning happens before synthesis while register transfer level (RTL) and gate-level, which are considered structural partitioning, happens after synthesis. Behavioral level partitioning is considered superior when working with larger designs [13].

In coarse-grained logic, the advantages can only be achieved if the given application is using the coarse-grained logic units. These units can only execute the tasks that they are designed for and if those tasks are not needed then those logic units are not being utilized and are wasted. This creates a dilemma where precise coarse-grained logic units are indeed faster and more efficient on their tasks, but

fine-grained logic can work on multiple different applications. This leaves the question what percentage of fine- and coarse-grained logic should there be on an FPGA for the best performance.

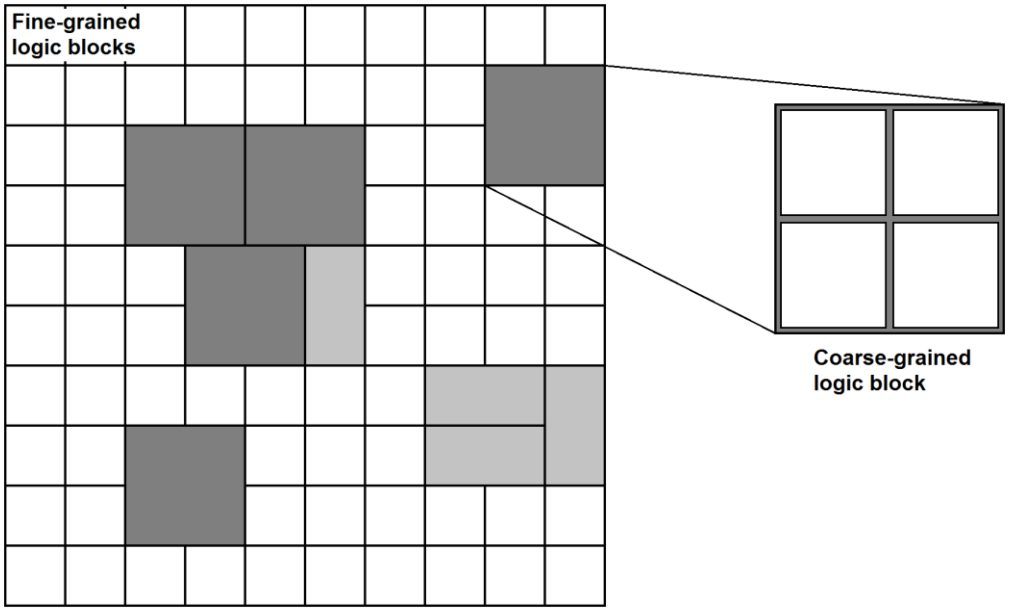

Figure 7. Fine- and coarse-grained logic blocks on a 10x10 grid.

Different waveforms can differ from each other quite a lot, which creates a problem with coarse-grained logic. Some logic units may work for one waveform but not for the others if the units are too specific. Taking into consider how this could be detrimental to the original goal of the product under development, there are no real advantage over the other opportunities researched in this thesis.

There is also a problem on how to lay these coarse-grained units on the FPGA to maximize their usefulness. In [14] the interface between these units and the rest of the hardware is explored on a broader scope. Part of the summary includes that the most efficient way is to use square shaped units that are next to each other in the middle of the board. This grants easier access for the applications to utilize the embedded blocks and minimizes the connections that are needed between the blocks.

These findings can be beneficial also on partially reconfigurable FPGAs implying that the reconfigurable regions should be square-like, close to each other and positioned as middle as possible. Maximizing these attributes while staying within given constraints can come out as taxing and take too many work hours to be beneficial but keeping these findings in mind during the research and development may bring desired benefits.

3.4. Partial reconfiguration

Partial reconfiguration is a technology that makes it possible to reconfigure certain physical areas on an FPGA after its initial full configuration without disturbing the operation of the other areas on the same FPGA [15]. Without partial reconfiguration,

the functionalities of the hardware cannot be changed without full reconfiguration of the FPGA, which takes time and resources depending on the device.

Partial reconfiguration makes it possible to change the functionality of a specific area on an FPGA while the rest of the device is running. This reconfigurable area can be used for different functions, which reduces the total amount of logic resources, therefore decreasing the size of the device, and doing this on the fly is faster than rebooting the system in between. Testing and performance are also improved when there is less downtime. Testing may also gain further benefits from this kind of fragmentation. If there are development changes only on these reconfigurable parts, it might be enough to verify the functionality of those and not the whole system, speeding the testing process.

Reconfiguration done on the fly while the FPGA is operational is called active or dynamic partial reconfiguration. When the device is inactive in shutdown mode, partial reconfiguration is still possible and it is called static partial reconfiguration [15, 16]. This thesis focuses on the former.

Xilinx offers two different ways to implement reconfiguration on their hardware. These are difference-based and module-based partial reconfiguration. The former is used in smaller changes while the latter can be used to change larger modules [16, 17].

As the name implies, the difference-based method keeps only track of the differences between configurations and uses this information to achieve reconfiguration. Thanks to this, while using this method the generated partial bitstreams are quite small because they do not need to store all of the device information. Difference-based partial reconfiguration is useful for very small changes like altering LUTs or memory blocks.

Module-based approach alter larger area. The reconfigurable modules (RM) can be swapped into and out of their designated area on the FPGA changing the operations on the fly. Figure 8 shows how four different modules can utilize the same resource space compared to an implementation without any partial reconfiguration and how this makes it possible to use smaller FPGA. However, switching between different modules demands more from the design and brings more restrictions to the project than smaller changes that can be done with difference-based reconfiguration.

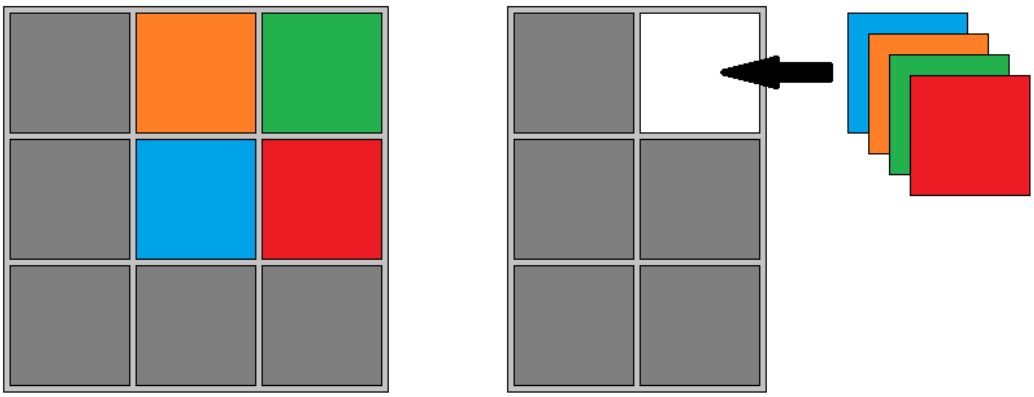

Figure 8. Larger and smaller circuits without and with partial reconfiguration respectively.

The waveforms considered in this thesis occupy remarkably large areas on the Ultrascale+ FPGA devices used in the project so module-based reconfiguration fits well to that. Difference-based reconfiguration might work in the case of similar waveforms by changing the functionalities between them, but in this case the waveforms differ so much from each other that it is not a feasible method to realize the wanted fast switching.

Partially reconfigurable hardware brings opportunities for faster development, cutting bill of materials, and flexible troubleshooting. SDR can be implemented with several parallel waveforms on the fabric for different communication standards but this increases the size of the device and while one of the waveforms is in use, the others are idle and are just wasting resources and consuming power. This makes the waveforms of an SDR a desirable target for partial reconfiguration. Waveforms can occupy a significant amount of area on an FPGA and that can be seen in the Table 2.

In the context of this thesis, in one of the products ⅔ of the whole FPGA has been reserved for waveforms and ⅓ for other platform services. If the product needs different capabilities through different waveforms, which is the case in this thesis, partial reconfiguration is a viable mean to make those wanted waveforms into RMs and using only a part of the FPGA for all of them while releasing more resources for other applications.

Table 2. Budgeted distribution of resources for different parts on the FPGA with two parallel waveforms

	LUT	**FF**	**BRAM**	**URAM**	**DSP**	**MMCM**
Available on FPGA fabric	230400	460800	312	96	1725	8
Budget for Platform	76800	153600	104	32	575	6
Budget for WF1	76800	153600	104	32	575	1
Budget for WF2	76800	153600	104	32	575	1

3.5. Summary

FPGAs can be programmed repeatedly with different configurations and thus change the way they work. Changing the implemented configuration in order to enable new functionalities on the SDR is a powerful tool to extend the capabilities of the given platform. Reutilizing the same resources and the possibility to change different waveforms via software maintains the flexibility of an SDR and keeps the size of the device small. Mobility, in which power usage and size play key roles, is a very high priority with the final product of which this thesis is a part of.

Dynamic partial reconfiguration suits well to the needs of an SDR device residing on an FPGA. To enable fast switching between waveforms without partial reconfiguration, each waveform would need a dedicated and parallel implementation on the FPGA. This would require resources and thereby increase power consumption and bill of material costs. With partial reconfiguration waveforms can be partially reconfigured, which makes it possible to use the same logic resources for all of them. Different waveforms can be swapped quickly from and to an FPGA depending on the need without ever interrupting the running system. This bypasses the need to reboot the FPGA when reconfiguring different functionalities on it and therefore the whole SDR if other parts rely on services running on that FPGA. This lowers the downtime

from tens of seconds or even minutes of a full reboot to mere seconds of a partial reconfiguration.

There are also other advantages when partial reconfiguration is introduced to the design. It enables smaller hardware because the different waveforms do not need to fit on the hardware at the same time, which in turn reduces costs and power consumption. However, it is good to notice that this is true in the case where fast waveform switching is mandatory. The device can be rebooted and fully reconfigured using the same amount of hardware as with partial reconfiguration if there are not any time restrictions.

Partial reconfiguration can also make workflow and implementation of the research and development faster, because if the changes only affect the RMs and not the static portion of the FPGA, only the functionality of the RMs need to be verified again. The interface between static and partially reconfigurable area also needs to stay intact for this to be true since changes into the interface means changes to the static.

In the future, there could, and most likely will, also rise a need to implement new waveforms on the existing old product. If partial reconfiguration has already been successfully implemented on that product, it is more flexible to introduce a new RM to it than refactoring the design in order it to be compatible with this new waveform. Without partial reconfiguration, preparing the project for future updates like this would require much more flexibility from both the software and hardware of the device, which could create more complex solutions which in return takes more work hours and raises costs.

4. PARTIAL RECONFIGURATION

Due to the increasing number of gates, popularity, and therefore times of configurating FPGAs, partial reconfiguration holds a lot of potential. There have been coarse- and fine-grained implementations with different logics located on the reconfigurable area. A good example is given in [18], which goes over a comparison between 3G, LTE and WIFI standards on a Xilinx FPGA with different solutions. The study shows how dynamic partial reconfiguration lowers the needed space and power usage when these standards are made partially reconfigurable. However, this also creates some time overhead from the reconfiguration, which is not needed with full configuration with parallel modules since all of the modules exist on the FPGA already.

With SDR, partial reconfiguration also demands more memory with the static and all of the RMs compared to a full configuration with support for all of the required standards. However, in case of [18], the difference was found to be only 1.4 MB (3.8 MB versus 5.2 MB). This is heavily subject to how much the modules overlap each other with their logic and how the initial implementation is realized with partial reconfiguration. The required memory in the case of the this thesis is discussed later in the experimental implementation chapter.

Research presented in [19] explores the usage of a tool called PARBIT [20], which can create reconfigurable bitfiles for FPGA out from existing bitfiles. That research uses a term dynamic hardware plugin (DHP), which is equivalent to RMs discussed in this paper. As stated, PARBIT accomplishes the partial reconfigurability using earlier bitfiles, while Xilinx Vivado, which is used in this project, can be used to create a project around this concept where the completed implementation is already partially reconfigurable.

PARBIT brings the flexibility of partial reconfiguration to existing designs but does not help during the development of the bitfiles that it uses as input. Using PARBIT also creates its own restrictions and rules to synthesis, routing, and placement that must be taken into consider during development [19]. So this tool can be used on existing bitfiles in order to combine parts of them into a partially reconfigurable whole. However, the original files must be developed with the special restrictions in mind. Xilinx Vivado on the other hand gives a framework for partial reconfiguration in order to ensure its functionality in the final stages.

PARBIT, however, shows that partial reconfiguration can be implemented after the workflow of creating a bitfile, so it is not absolutely necessary to accomplish it during development. It must also be taken into notice that the introduction of PARBIT took place in the early 2000s, and technology on this area has since then gone a long way. This tool might still have some use, but for future developments it might already be obsolete.

4.1. Development of partial reconfiguration

Xilinx introduced partial reconfiguration support to its FPGA hardware in the 1990s with XC6200 but the popularity of this technology grew in the early 2000s when Virtex family line got its second member: Virtex-II [21]. With the next generations, Virtex-4, -5, and -6, came new architectural improvements to FPGAs and partial reconfiguration. These include the switch to LUTs for flexibility, which eased

connectivity and separating the silicon area into smaller clock regions so reconfiguration could take place in smaller areas. Table 3 shows the release times of the Virtex generations as well as the process technologies, which also shows the increased computing power, which in return enabled new techniques and solutions on the FGPAs [22, 23].

With the introduction of Xilinx integrated synthesis environment (ISE) Design Suite, also partial reconfiguration evolved. It started increasingly to rely on software while the older versions relied on specific hardware solutions and techniques that can be seen as primitive compared to today's standards.

When Virtex-7 series was introduced in 2010 and Vivado Design Suite in the following years to substitute ISE Design Suite, also partial reconfiguration went through significant development. Until 2017, partial reconfiguration was a separate tool within Vivado but has been since then included into the software itself and needed its own license until the release of version 2019.1 [24]. All of this shows that partial reconfiguration is constantly under development and is being optimized by Xilinx for better performance.

Table 3. List of Virtex family FPGAs

Virtex family	**Introduced**	**Process technology (nm)**
E	September 1999	180
II	January 2001	150
II Pro	March 2002	130
4	June 2004	90
5	May 2006	65
6	February 2009	40
7	June 2010	28
UltraScale	May 2014	20
UltraScale+	January 2016	16

With Virtex family, Xilinx also introduced integrated configuration access port (ICAP) technology which gave the FPGA a possibility to reconfigure itself. Configuration data could be read from an external memory through ICAP and loaded into the FPGA without any external controller. The device could now change its behaviour according to what is demanded from it all by itself. This technology is in a key position for a fast change of modules in partial reconfiguration solutions. Later in development other such ports were introduced on new devices: processor and media configuration access ports (PCAP, MCAP). These brought new ways to reconfigure bitstreams onto FPGA fabric.

Partial reconfiguration was less complicated but also more restricted with its capabilities during its earlier iterations. With new technology and solutions added on the Xilinx's devices came major advantages and improvements to partial reconfiguration design but also new restrictions. These new additions had to be taken into consider in the partial reconfiguration workflow so reconfiguring the FPGA on the fly would not interfere with these functionalities.

Good example is the requirements for the area where the partial reconfiguration would take place. On older iterations, this area would have to be specific size and shape like expand the whole height of the FPGA, but on present devices this area can be located almost anywhere and can be almost any shape. This however also raises

questions like where and how this area should be located to gain optimal resource utilization while on previous devices there was only a few options to choose from, which would streamline the workflow.

While Xilinx has been the forerunner on partial reconfiguration, other vendors have also been working on their own solutions like this. However, Intel's Altera is the only one of them still supporting partial reconfiguration. Reasons for this are mainly due to complicated development of such technology and limited usage of it in practical solutions. Xilinx and Altera are also the lead developers of large FPGAs that benefit more from partial reconfiguration. Smaller devices can utilize simpler methods to achieve similar kind of goals that partial reconfiguration brings to larger FPGAs.

Naturally smaller FPGAs are cheaper and easier to manufacture than larger ones. Creating hardware that is capable to utilize partial reconfiguration makes it possible to implement larger solutions on smaller FPGAs and thereby cut costs while still keeping the performance high.

4.2. Modulation and waveforms

Modulation and demodulation with coding and decoding are key functions of waveform PHY's TX and RX pipes respectively. Addition to these functionalities, there can be fast Fourier transform (FFT), synchronization, quantifiers and many more depending on the waveform that is being used. Figure 9 shows a basic block diagram for these pipes for orthogonal frequency-division multiplexing (OFDM) waveform [25]. These parts also have to be tuned for their specific waveforms, which makes them easily noninterchangeable between waveforms.

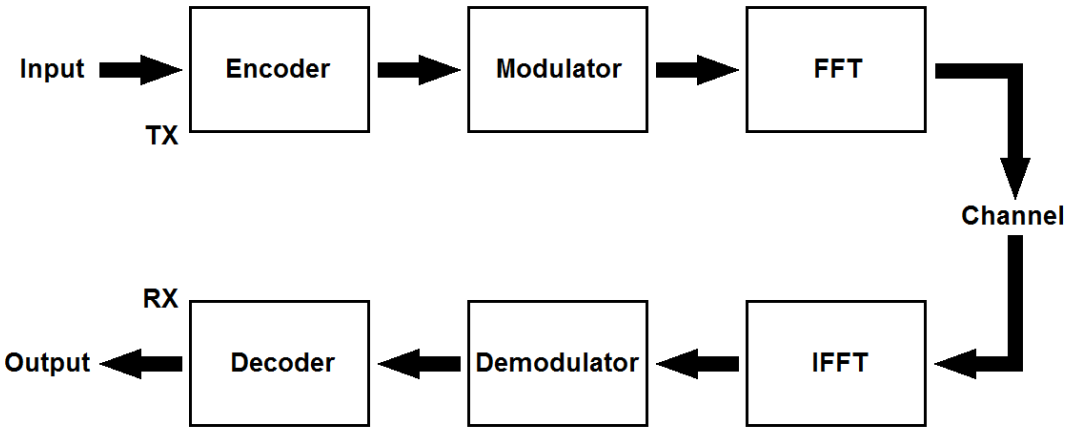

Figure 9. Basic TX-RX block diagram for OFDM.

Modulation in telecommunication can be analog or digital and there are many different types of modulations, but they can be divided into three different groups: amplitude, frequency, and phase modulation. In analog, these are called modulation and in digital they are called shift keying, but the quintessential meaning in both cases are the same.

In each different modulation, the corresponding value is modulated in order to carry the wanted information over radio signals. These modulations can be combined

into a more complex modulation that can carry more information over the same period of time. One of these modulations is quadrature amplitude modulation (QAM), which combines amplitude and phase modulations and is one of the most used modulation techniques in telecommunication this day. QAM itself can be expanded with various ways to increase its bitrate and reliability [26].

QAM can be separated into different categories based on the amount of constellation points the modulation uses. These are represented by a number like 16-QAM and 64-QAM, which can be seen in Figure 10. The more constellation points there are the higher bitrate can be achieved, for example with 16 constellation points one point can hold 4 bits, with 64 one point can hold 6 bits. This also establishes higher risk for errors through noise and other disturbances.

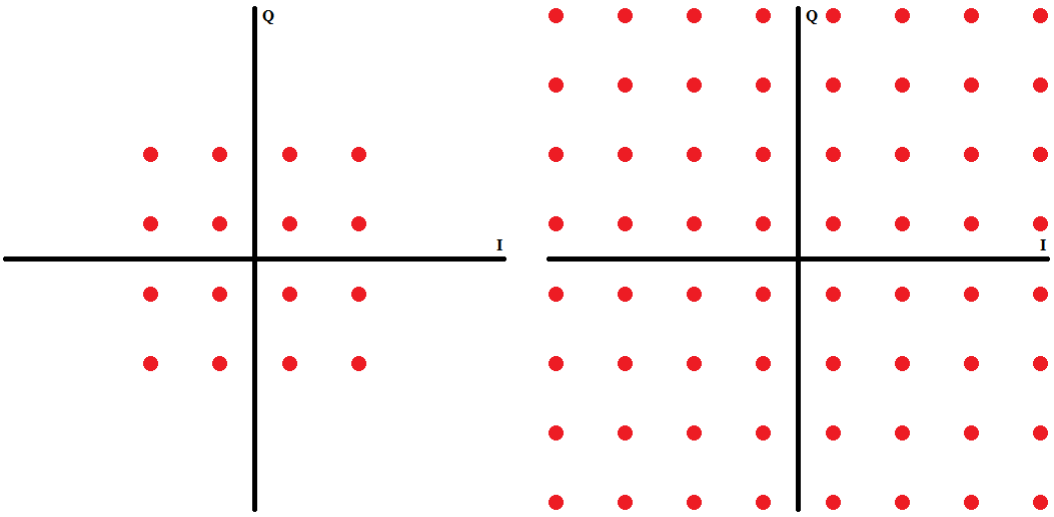

Figure 10. Constellation of 16-QAM (left) and 64-QAM.

Since different QAMs can modulate different number of bits into one symbol, the word lengths are different. The transmitted signal could still be demodulated with any QAM when the amplitude is normalized, but the result would be worthless with a wrong receiver because the symbols would be interpreted incorrectly. This means that different modulations need distinct PHYs to work properly.

Even when waveforms are using the same modulation technique on the base level, they may differ from each other so much that distinct TX and RX pipes are needed. In this thesis, there are three waveforms, A, B, and C, with different modulations and features. There are major differences between them so creating exclusive PHY for each of them is inevitable. This means designing a configuration with these waveforms in parallel is far from an optimal solution.

4.3. Waveform switching with partial reconfiguration

There are two kinds of controllers for partial reconfiguration: external, like a PC connected to the FPGA that has the partial reconfiguration modules, and internal, which resides on the FPGA itself [27]. This internal reconfiguration is mainly done through ICAP, but can also utilize PCAP or MCAP if needed [21]. In the context of this thesis, the development was done with an external controller but the final

configuration of the product can be thought to use the internal scheme, since there will be a separate processor, which will take care of loading binaries on the FPGA. The end user should be able to change the waveform on the FPGA without any external device that would be connected to the device.

In this project, waveform A has been designed to use the whole budget that has been given to it, waveform B around 50%, and waveform C 25%. The space would get scarce if all the needed waveforms had to be implemented on the silicon at the same time. The objective is not even to support a configuration where all of the waveforms would exist on the FPGA at the same time because of the lack of resources. Rigid solution without partial reconfiguration also does not help with the plan that these SDR products could support completely new waveforms in the future. Partial reconfiguration is considered as an option to solve the problem to fit all these and future waveforms on the device against the idea of creating different full bitfiles and using full reconfiguration.

Partial reconfiguration also brings benefits for downtime related to rebooting the device. Without partial reconfiguration, the FPGA would go through full reconfiguration, which also reconfigures and resets other vital services on that FPGA and therefore would require a complete reboot. Partial reconfiguration on the other hand only affects predetermined areas on the silicon and can happen while the device is running and the downtime of a waveform that is being reconfigured would be just mere seconds.

4.4. Restrictions and drawbacks on Xilinx FPGA

There are also drawbacks with partial reconfiguration. Adding more functionalities into a project increases the complexity, which in return increases the needed work hours, cost, and time to market. However, partial reconfiguration should bring advantages in a long run with its benefits and not only for this product but for next generations where it can be implemented with the help of this thesis. Before any of that, there are obstacles to overcome for partial reconfiguration to be put into practice successfully.

Modules that are made partially reconfigurable need a predetermined area, a partially reconfigurable region (PRR), from the targeted FPGA. This area can be anywhere on the FPGA as long as it contains enough resources for each reconfigurable part and it does not include any non-permitted parts. Some of these restrictions created by logical units are addressed in the experimental implementation chapter.

The design of the project and hardware may bring their own restrictions, as for example some part of the design must be located on one specific region on the silicon. This may occur because there are specific resources only on one part of the silicon or that the hardware has some special functionalities for something particular, for example debugging, so those parts of the design must be routed on that distinguished part of the FPGA if they are present.

The PRR should also be as simple as possible for better performance, for example a square. A more complex region requires more complex routing and placement to fit the needed structure inside it. These two factors, resources and shape, could create problems depending on the used FPGA and how its components are laid out. Trying to fit all the necessary parts into the PRR while at the same time trying to keep the

shape simple and not covering too much extra resources, ergo creating overhead, can be a taxing endeavor.

If the project has same intellectual properties (IP) that are situated on different places, for example on static and RMs, they must have different names for Vivado to implement them correctly. The IPs can still be the same, but their names must differ from each other. This increases the complexity because normally Vivado is able to use the same IP in every instance in the project if it just has the same name.

4.4.1. Overhead

The PRR that has been reserved for partial reconfiguration is permitted only for the RMs. Static components cannot occupy any of this area. This region is one or a combination of rectangles on the FPGA that will cover all the needed resources for the RMs. It is possible in theory to pick only the needed resources from the silicon to the PRR, but this would increase the complexity of the region and would make routing the device on the FPGA near impossible.

In reality, the area will cover more resources than what is needed depending on the layout of components on the FPGA. This is considered overhead of resources because the RMs do not use them and static implementation is not allowed to utilize them. In the contest of this thesis, this can be seen most clearly when comparing the usage of the least and most complex RMs on the FPGA, where the waveform C covers only a fraction of the area that waveform A takes, depicted in Figure 14 and Figure 12 respectively. The Figure 11 shows the layout with an empty RM, which Vivado refers as greybox, and Figure 13 with waveform B as the RM. In all of these figures, the orange part represents the static logic and the blue shows the used logic of an RM, residing inside the PRR, which consists of four rectangles. In Figure 11, the PRR is mainly black because the RM is an empty greybox module without any logic.

RMs can differ a lot from each other and one of them may require resources that the others do not and vice versa. In the worst case, the first RM utilizes a lot of LUTs and FFs but very little BRAMs and the second RM utilizes very little logic but a lot of BRAMs. If these kind of RMs are implemented on the same PRR, all of their needed resources must be included in it. This increases overhead and unused resources when one RM does not utilize all of the resources that have been given to it. The more similarities the RMs have with their resource utilization, the more advantage partial reconfiguration brings. This is one concept that is good to take into consider in early design steps when it is under discussion what functionalities are realized on what part of the device.

Different functionalities can be realized through multiple different means, for example FIFO buffers can be made through BRAMs or LUTs. Tweaking these resources and using different techniques between RMs can bring their utilization characteristics closer with each other.

Figure 11. FPGA layout with greybox RM.

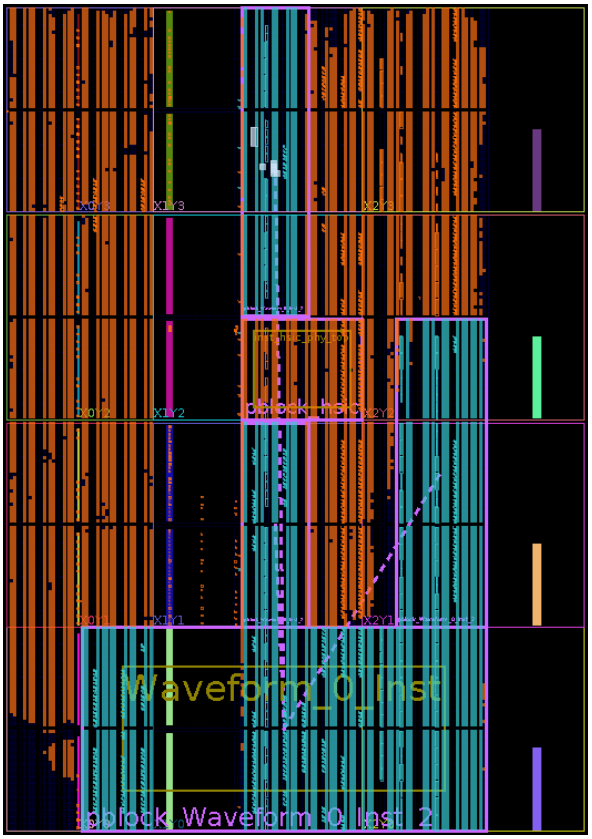

Figure 12. Utilization of PRR with waveform A.

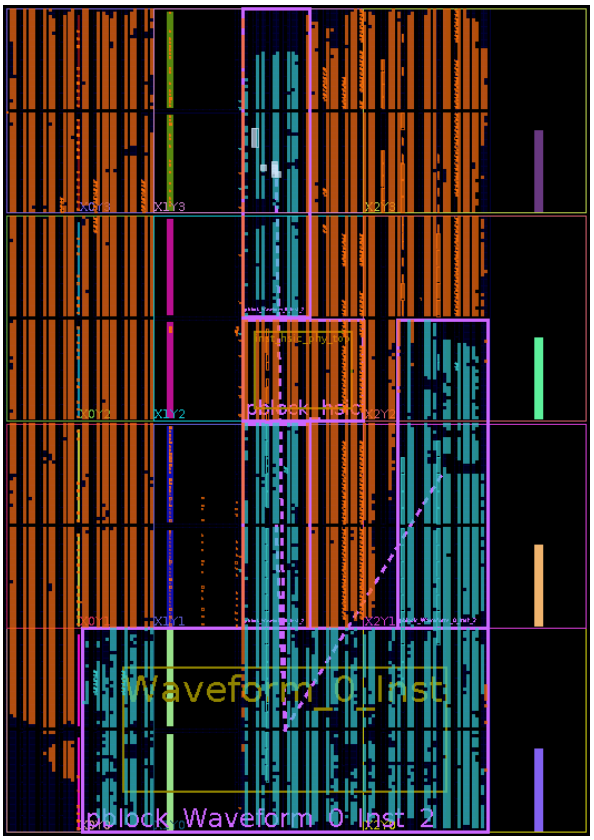

Figure 13. Utilization of PRR with waveform B.

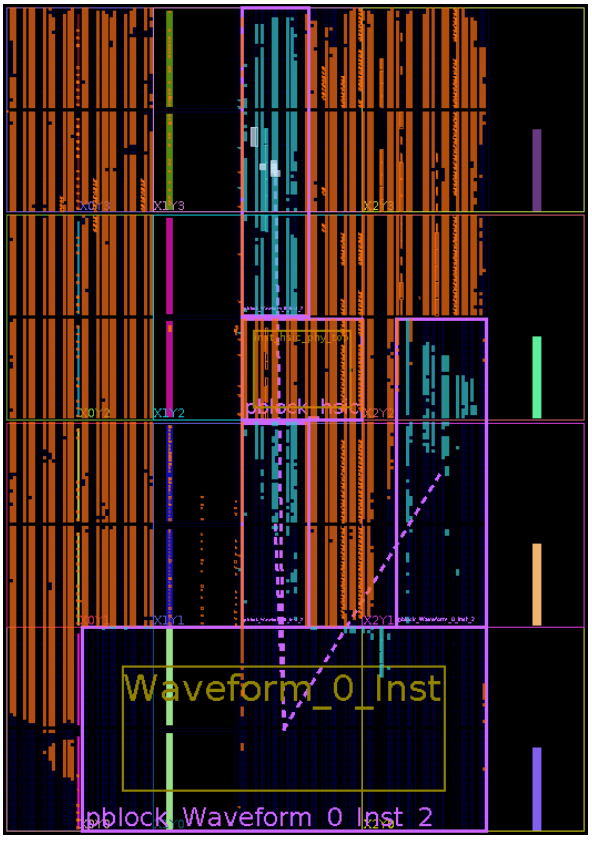

Figure 14. Utilization of PRR with waveform C.

The overhead can be minimized with different tactics, but there will always be some overhead on the area and it is not only a negative thing. The overhead gives more margin for the RMs in general, which helps the routing tool algorithm with laying the logic on the FPGA. Future bug fixes, feature updates, and even totally new waveforms would be easier to fit without changing the size or location of the PRR. The overhead can be seen as a design space for these additions and fixes. If the final product is implemented with tight PRR that is tailored for the original waveforms, only a little more complex waveform can be much harder to implement from this point forward. Also, the unused resources of the overhead do not waste any dynamic power.

In the end, it must be noticed that the final product with partial reconfiguration will still use less space and resources than fitting each waveform onto the board at the same time even when overhead is taken into consider.

5. EXPERIMENTAL IMPLEMENTATION

This thesis was part of a research and development project, where an SDR product with different waveforms was developed. Partially reconfigurating the used FPGA was chosen as a solution to tackle the problem of enabling fast waveform switching without simultaneously fitting all the waveforms on the given FPGA. The work was done on Xilinx MPSoC UltraScale+ devices with Vivado Design Suite software development tool, versions 2017.4 through 2019.1, which was run on a Unix-like operating system. Vivado's partial reconfiguration design flow was also tested on Windows based computers.

Secondary goals for partial reconfiguration were to reduce power consumption, make development more efficient, and to decrease the used hardware resources. Optimizing these factors was expected to result a lighter and more efficient product.

Vivado gives the possibility to manage the FPGA design project via TCL-scripts. Design project based on these scripts defines the source files and settings for synthesis and routing processes. TCL-scripts are very useful for version-control and testing when the test environment needs to recreate the design flow multiple times the same way. Focusing version-control on these scripts is much more effortless than managing whole Vivado project databases without them. During the development, also the partial reconfiguration design flow was implemented with TCL-scripts.

5.1. Vivado workflow

Vivado projects in default start without partial reconfiguration active so that has to be enabled manually. This action is a simple click of a button but irreversible, so it is good practice to create a copy of the project before this. After this, it is possible to define the wanted sources into partially reconfigurable modules. The following workflow is depicted in Figure 15.

There are restrictions to what is allowed on an RM and what is prohibited in addition to other design criteria. The list of these is quite extensive and specific, because there are differences between FPGA platforms and limitations in certain conditions. Some of these cases are covered in the next sections, but the guide for partial reconfiguration offered by Xilinx [24] goes over all of them in a more specific manner.

RMs are also required to have configurations, which can be made with Vivado's Partial Reconfiguration Wizard tool. These configurations determine how these modules function and how they are run. If there are multiple PRRs, these configurations also specify how the modules are paired and on which region they are located in each implementation. The Wizard tool is also used to determine possible greyboxes and the parent-child relations, which is discussed in more detail in section 5.2.

After this, it is possible to run synthesis, which transforms the design into a gate-level representation [28] and allows the creation of PRRs. These are drawn on the device fabric and must include all the needed resources while not including anything prohibited. Choosing these areas is analyzed in section 5.3. The modules are assigned their corresponding PRRs through Vivado's netlist.

Running implementation lays the design on the fabric and after that, it is possible to import bitfiles and other wanted files out of Vivado. The number of created bitfiles

depends on the earlier configurations and the number of RMs and PRRs. Section 5.6 covers this in more detail.

With partial reconfiguration, waveforms could be easily added or removed from the project. They could be combined with each other in any way in a platform with multiple parallel waveforms just by changing the configurations. They could be coupled even with greybox modules if needed.

Such flexibility does not work without partial reconfiguration. In order to create different waveform configurations in this manner, the project needs to be started all the way from the beginning with the new waveforms or remove the old ones and add the new ones after that. In any case, this means maintaining multiple different Vivado projects at the same time in order to create all the combinations compared to just one project with partial reconfiguration. This eases the maintenance and shortens the needed time drastically when only one project needs to be run compared to multiple.

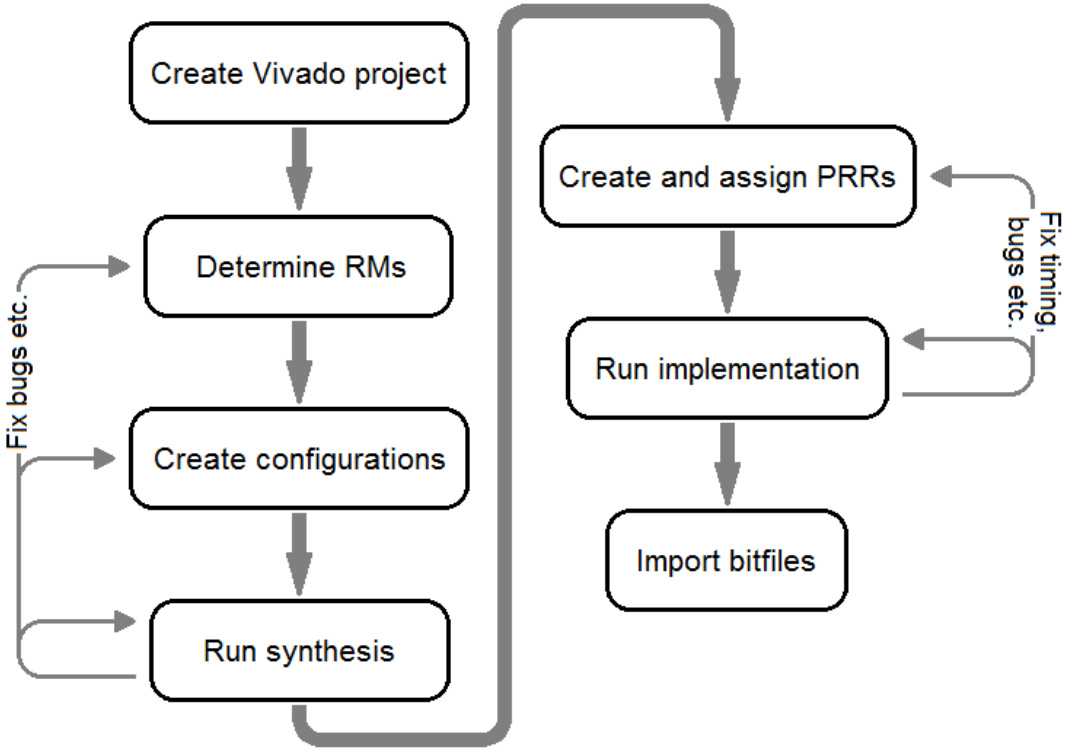

Figure 15. Creating partially reconfigurable Vivado project.

5.2. Module relations

When creating implementations for multiple RMs on a PRR, there needs to exist a parent-child relationship between the RMs. One of them must be chosen as a parent and the others are configured as children to that parent. The parent implementation reserves the preconfigured PRR for the RMs, implements the parent RM on that region and includes the static part of the application on the rest of the silicon. After a successful parent implementation, the children will be implemented on the

corresponding PRR. These tasks do not interfere with the static side after the parent implementation has been done.

Vivado allows the usage of a greybox as one of the RMs, which means an empty module. This module can be used as the parent module, which releases the other modules from the state of a parent and all of them can be realized as child modules. This can be seen as a benefit because if the parent module goes through changes, it forces the static side to also go through the workflow again even if the changes do not impact it directly. This cascades to all the children because they need to be implemented again for the new static side that was forced through the parent module.

The greybox module will not need any changes to it because it is an empty module. If the real RMs need any changes and the static does not, it can be left as is and only the RMs that are affected need to go through the workflow and they can just use the old static module as a reference for routing and such. This speeds up the workflow because working only with the RMs is much faster than routing the whole device repeatedly during development.

There are drawbacks in this parent-child relationship. This procedure can create problems if there are large differences between the RMs. The static side is optimized for the parent RM and the other RMs do not change the static implementation so the children have to work with more restrictions that have been created by the already made static side.

This is especially a major concern if a greybox module is chosen as the parent RM. In this kind of configuration, the static part of the implementation has minimum restrictions from the RM and therefore the usage of the silicon and routing may not work with the children RMs.

5.3. Selecting reconfigurable region

As stated earlier, RMs require an area from the FPGA that has been predetermined for them. These regions are referred as pblocks by Vivado and as PRRs in this thesis. Depending on the SDR variant, there were one or two PRRs, one for each simultaneously active waveform. According to the earlier research made during this thesis, these PRRs should be as simple as possible, include all the needed resources while not including any prohibited resources or solutions, should be situated near the middle of the device, while minimizing the overhead but still giving enough space for the routing to take place. A full Vivado project with layout implementation were first made without partial reconfiguration on the FPGA with two waveforms of type A to examine how Vivado lays this configuration on it. This layout was used as a starting point when determining PRRs for RMs.

During development, it was found out that the size of the created bitfile for each RM was subject to the determined PRR. Even though the areas covered almost the same amount of resources in two different implementations, the bitfile from the more complex PRR ended up being significantly larger. This is the result of how Vivado implements routing with partial reconfiguration. Xilinx FPGAs are divided into clock regions and the area for RM may go over multiple of these regions depending on the needed resources. If a PRR needs just some of the resources in a clock region, the entire region is added into the routing area that the RM can use. This does not increase the amount of resources in any way and static implementation can still use the other parts of this clock region that the RM does not use.

In the case of multiple different PRRs, there are corresponding routing areas for each of them. The routing areas are prohibited from crossing one another in Vivado. This means that other PRRs cannot use resources from a clock region that already has another PRR crossing it because both of them would try to include that clock region into their own routing area.

Vivado creates the routing area automatically based on the PRR and what clock regions it crosses. Through experiments, it seems like Vivado also makes the routing area into a rectangle if there is nothing in the way, for example another RM's routing area. If the PRR is really complex and crosses a lot of clock regions vertically and horizontally, this leads to including even more clock regions into the routing area and it will ultimately end up being remarkably large.

Figures 16 and 17 show an example of this behavior with 12 clock regions. The bitfiles that would be created according to the PRR in Figure 16 would be larger than the bitfiles from PRR A in Figure 17 although they cover the same logical resources. This is due to the larger routing area.

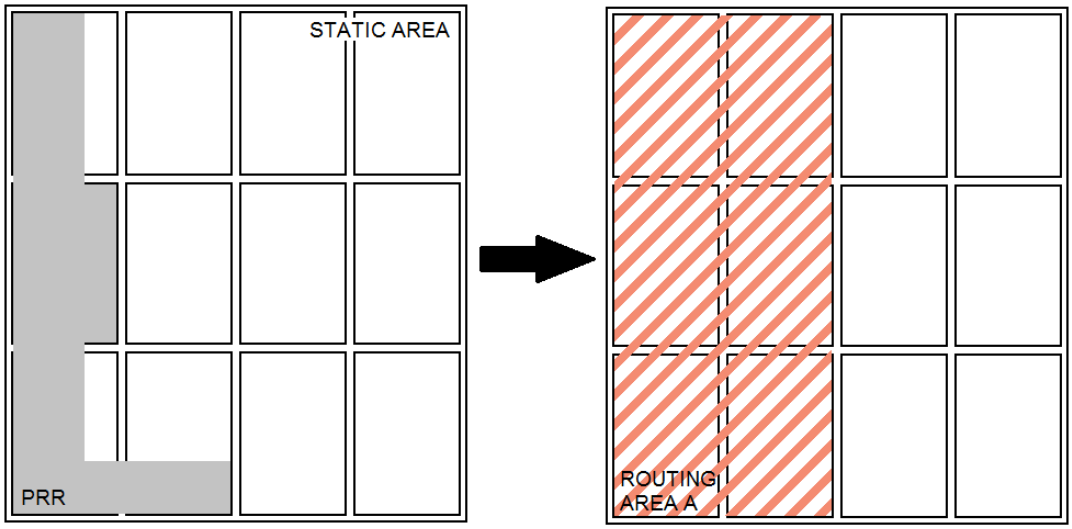

Figure 16. Creation of routing area with one PRR.

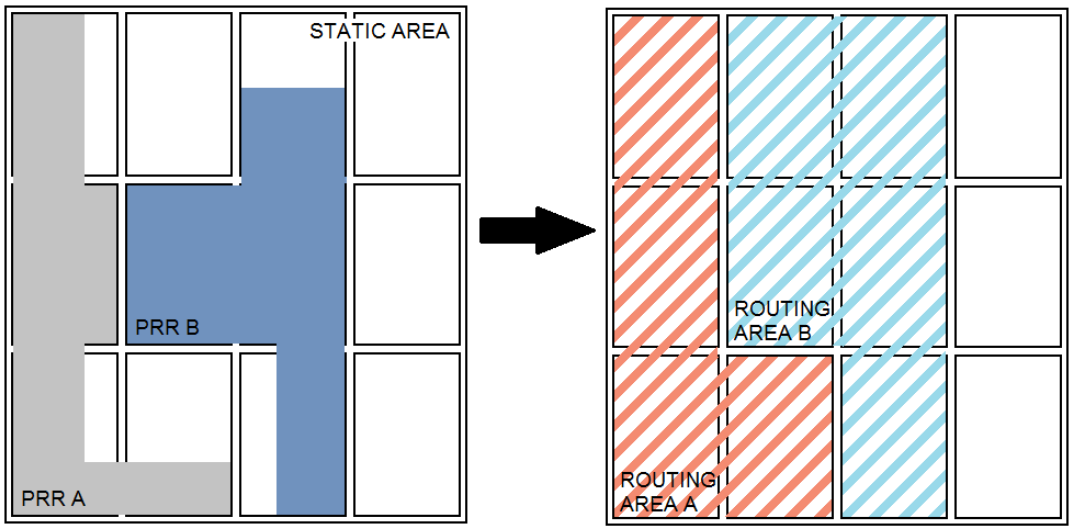

Figure 17. Creation of routing areas with two PRRs.

5.4. Vivado restrictions

There are restrictions to where the PRR can reside on the FPGA with Xilinx hardware. There are also restrictions what can be included to an RM.

Route-through signals are not permitted to go through an RM. During reconfiguration, such signal would be cut off and could create other problems and therefore it is prohibited. During development, this kind of a problem arose from a clock signal that was through-routed from waveform's in port to out port.

Few workarounds were explored for this case. However, these solutions were found impossible to implement by another restriction that prevents certain resources to exist on the RM's area, in this case input-output blocks (IOB). These blocks are allowed to be partially reconfigurable on UltraScale and UltraScale+ FPGAs in certain conditions, but those conditions could not be met in the project considered in this thesis.

The workarounds needed a mixed-mode clock manager (MMCM) primitive to reside in the PRR to work, but with the given Xilinx FPGA those reside next to IOBs. Trying to include an MMCM to the PRR would also include IOBs to the area, which is prohibited like stated earlier. This is also true for some other primitives like phase-locked loops (PLL) that were also considered as a solution to the original route-through problem. MMCM primitives were budgeted for the RMs, like shown in Table 2, but this kind of restrictions forced them to be physically located on the static side. These restrictions should be considered in the early phases of the FPGA's internal architecture planning in the project.

During development, any FPGA project needs debugging and logical resources for that, like integrated logical analyzers (ILA). Enabling debugging with partial reconfiguration needs extra steps when compared to without it. Xilinx recommends using specific naming style of ports in RM-static interface. When these ports and logical blocks like ILAs are present on RMs, Vivado automatically creates debug hubs and connections for the static side and RMs. This style of enabling debugging was chosen for this project for its simplicity after experimenting with debug bridges.

This method however forces the parent module to have these ports and logic for Vivado to go through these steps creating the debug hub also on the static side. If this hub is missing from the static side, but the child modules do have it, the connections will not occur between these hubs, which leads to a failing implementation [24].

Introducing partial reconfiguration into a Vivado project increases its complexity and managing it becomes more challenging. It is beneficial to maintain a design project without partial reconfiguration for comparison and debugging purposes, because the gained information from that could help with partial reconfiguration.

5.5. Power consumption

Using a partial reconfiguration solution decreases the amount of needed LUTs when compared to the full system implementation with parallel waveforms. Using the same resources for different tasks at different times makes it possible to use a smaller FPGA altogether, which reduces costs and power consumption. With partial reconfiguration, only the needed modules are loaded on to the FPGA for any given time, which lowers the amount of active LUTs and therefore decreases power consumption.

In [18], the difference between consumed power with RMs and a full implementation is significant. The smallest module's logic, 3G, consumes only 0.24 mW while having all the three standards on the FPGA at the same time ranks the number to 171.47 mW. The most complex module, LTE, consumes 128.8 mW. This shows the possible power savings that can be achieved with partial reconfiguration.

Figures 18-20 show that there is no actual difference in power usage between implementations when partial reconfiguration is enabled and when it is not in the case of only one waveform in each implementation in the SDR product this thesis focuses on. With partial reconfiguration, there can be seen a slight increase on the total consumption over the board, but that is expected with more complex design and overhead from the PRRs.

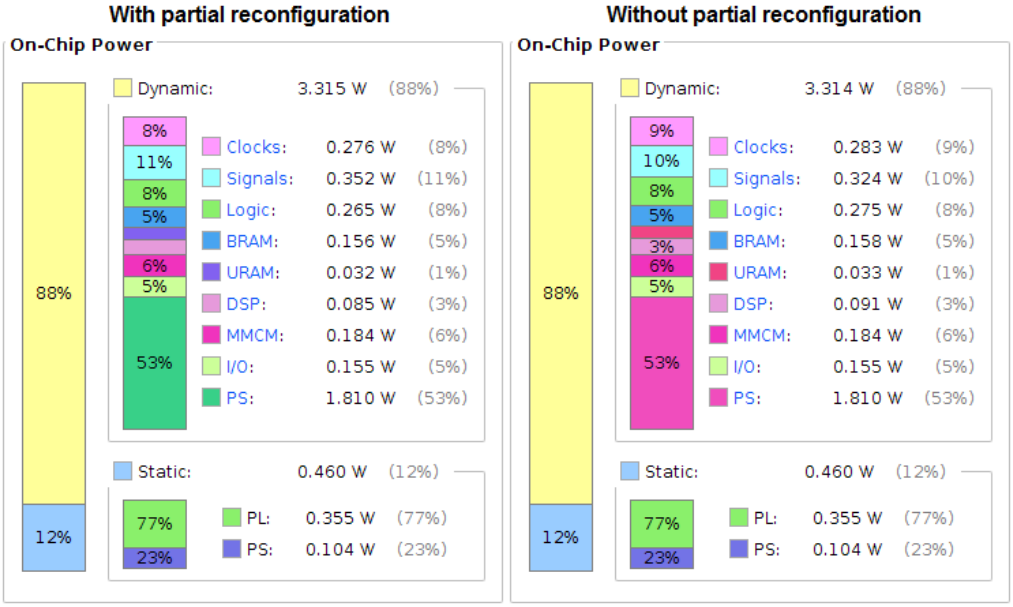

Figure 18. Power consumption with waveform A.

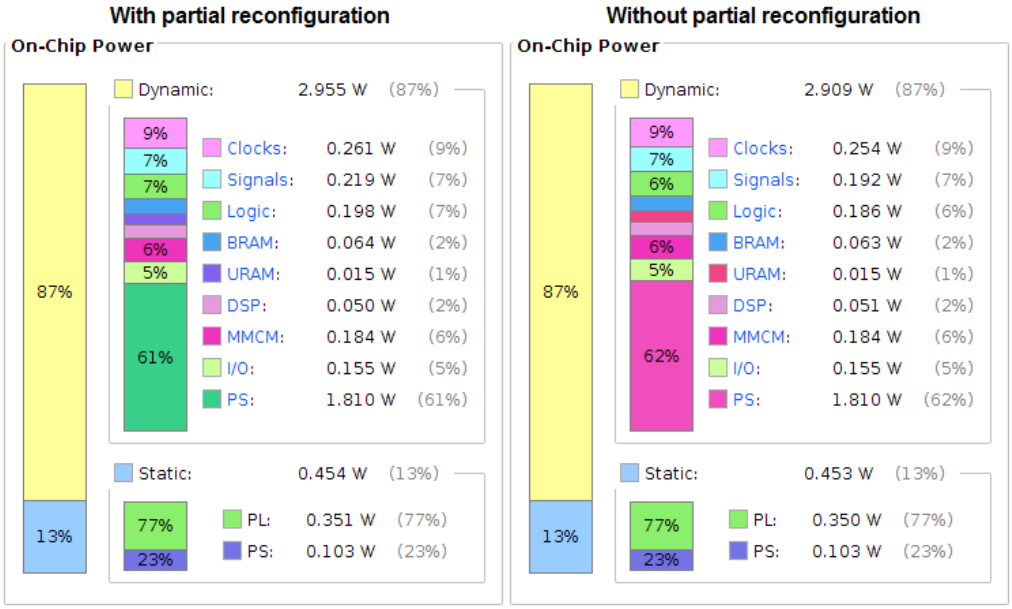

Figure 19. Power consumption with waveform B.

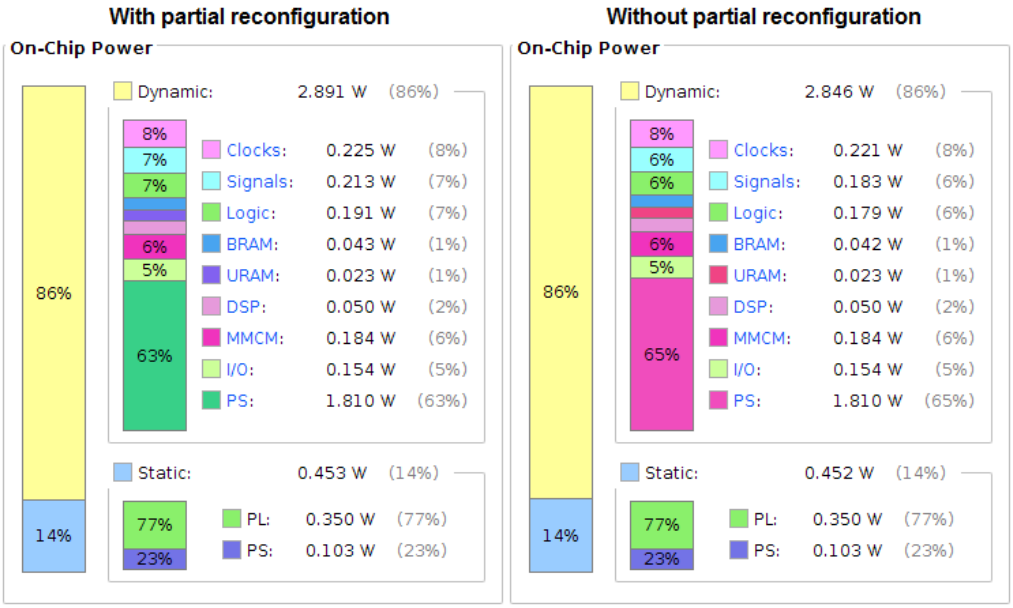

Figure 20. Power consumption with waveform C.

The final product will use multiple waveforms and without partial reconfiguration the right sides of these figures would see a drastic increase in the power consumption with multiple waveforms on a larger FPGA at the same time. With the partial reconfiguration enabled, this configuration already has all these three waveforms implemented as RMs, although only one active at a time, and the left side in these figures could demonstrate the power usage of the final product. These images were extracted from Vivado, which gives a power consumption estimate after a successful implementation.

5.6. Usage of memory

While partial reconfiguration decreases the need for hardware resources by enabling them to be used for different tasks at different times, it also lowers the combined size of generated bitfiles. This opens more space from the memory where the bitfiles are stored and from where they are loaded into the FPGA. There are two main factors that affect the needed non-volatile memory when comparing implementation with or without partial reconfiguration. These are the partial reconfiguration itself, the overhead and PRRs, and the needed bitfiles and the amount of them.

When comparing just the inclusion of partial reconfiguration in this project and the effect it had on the file size, the differences were quite negligible. Depending on what time during the development these sizes were compared, both solutions could take less space at one time or another while the difference never went over 0.5MB.

These variances were mainly caused by the amount of overhead and how the PRRs are configured, which were discussed earlier in this thesis. If the area is really complex and covers multiple different clock regions, the routing area can cover almost the entire FPGA. This can dramatically increase the size of partial bitfiles and if the routing area includes the whole FPGA, the size of the bitfile can even match the size of a full bitfile.

The number of bitfiles affects the needed memory in a much larger scale. Without partial reconfiguration and fast waveform switching, for this project there should exist full bitfiles for every combination of wanted implementations with different waveforms. With partial reconfiguration however, there only needs to be one main bitfile with all the static parts and partial bitfiles for the RMs. The full bitfile would also include the parent waveform for it to function properly. This means that the parent RM actually appears twice in the final implementation: once in the full bitfile and once as a partial bitfile.

In the case of the platform with one active waveform, without partial reconfiguration there would need to be three full bitfiles for all three waveforms and with partial reconfiguration one full bitfile and three partial bitfiles for RMs. The number of different bitfiles naturally increases with the platform with two active waveforms. The number of possible combinations can be calculated by combination with repetition

$$\left(\binom{n}{k}\right) = \binom{n+k-1}{k}, \tag{1}$$

where n is the number of waveforms and k is the number of active waveforms on the platform.

Therefore, with three possible waveforms there exist six different combinations with two active waveforms. This means without partial reconfiguration there would need to be six different full bitfiles. With it, there still needs to be one full bitfile, but with two active waveforms there exists two different PRRs. Those regions need different bitfiles from each other, so each RM needs a bitfile for each area. That means with three waveforms and two PRRs, six different RM bitfiles needs to exist in order to create every configuration combination.

The combined size of these generated bitfiles was without partial reconfiguration 85MB and with it 47MB. The combined size is cut almost in half and this is only with three different waveforms. When the product would be updated with support for new waveforms, in the case without partial reconfiguration, every new combination would need to be made into a full bitfile. With partial reconfiguration enabled, only the new RMs need to be added and they could be used to create new combinations with the old RMs. The number of full bitfiles would increase more rapidly, than the number of new RMs for partial reconfiguration solution, and that way the overall needed space for the bitfiles accordingly. The increase in number of bitfiles can be seen in Figure 21.

The impact of possible greybox RMs were left out from these calculations since there are no such thing when partial reconfiguration is disabled. Greyboxes are optional and are mainly used for development purposes if they are used in the first place. They can be used as the default RMs when the system is powered up if the design is able to do that with an empty module. One greybox was around 50kB in this project.

These calculations do not take into consideration if there is a difference between combinations like A-B and B-A. Meaning that if waveform A was in the first PRR and waveform B in the second, swapping these two between each other would have no difference. This might not be true in the final product.

For example, there can be different frequency ranges defined for different RMs and there can be variance between register addresses in these waveforms depending

on where they should be located. If this possibility is taken into calculations, the number of files will stay the same with partial reconfiguration but increase without it.

In the case of three different waveforms there would exist nine different combinations between those waveforms if the order does matter which would mean nine different bitfiles. The number of bitfiles would grow exponentially. This means that without partial reconfiguration, in the final product the number of bitfiles would be somewhere between (1) and n^2.

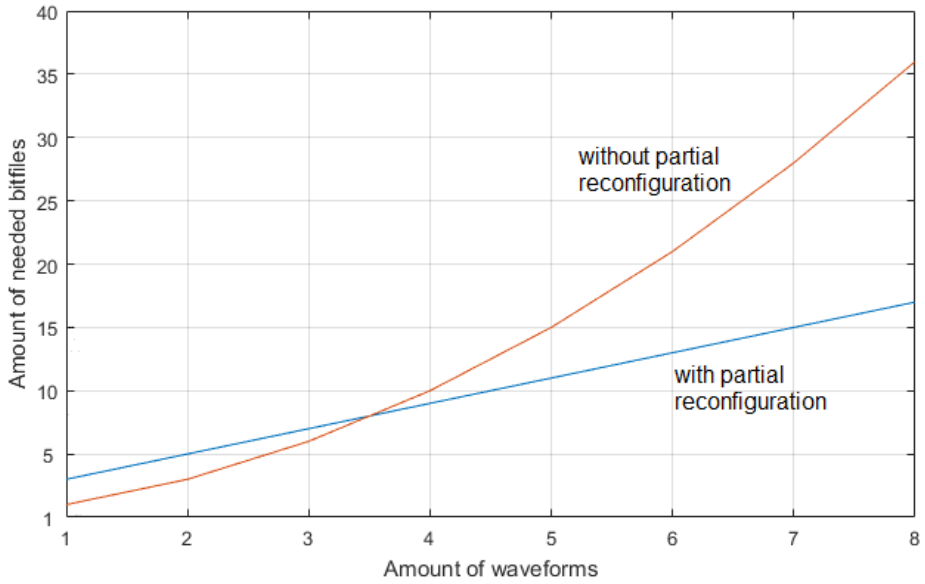

Figure 21. The number of needed bitfiles on the platform with two active waveforms depending on the number of supported waveforms.

5.7. Switching of waveforms

After a successful build was completed with partial reconfiguration, it was tested how the swapping between different modules happen in practice and how it would perform.

The results confirmed the studied benefits. Waveforms could be changed through the command prompt with simple commands and the change would take only few seconds. The connection with the FPGA stayed stable during the change although of course the waveform going through the change went down but was back online right after the reconfiguration. The system did not require any full reboots and could start to operate with the new configurations right after the change was complete. Older versions of the build without partial reconfiguration required a full restart of the system if the waveforms needed to be changed. This could take minutes, which is considerably longer than the few seconds of partial reconfiguration.

The functionalities were tested during the development but not with the final product. This thesis was completed before the benefits could be confirmed on the field on actual usage with the end user. The observed advantages during testing however proved that partial reconfiguration was capable of achieving the objectives that it was meant to overcome.

6. DISCUSSION

During this thesis and the project it was a part of, it was proven that partial reconfiguration could enable the wanted fast waveform switching. The partial reconfiguration would also bring other benefits along, mostly increasing the flexibility to add future waveforms to the product. However, advantages seldom appear without some disadvantages. Adding new features increase intricacy and takes time as well as resources to implement.

Implementing partial reconfiguration is most efficient and easiest when the research and development is started with that in mind. In the project this thesis is part of, partial reconfiguration was not a high priority from the beginning. The key driver for implementing it was raised later from the fast waveform switching time targets. The already chosen architecture did not prevent using partial reconfiguration, but it was not fully optimal either. This caused constraints on the manner how partial reconfiguration could be realized because the architecture could not be changed anymore without high costs. Older FPGA generations would also have a partial reconfiguration design flow that has matured longer providing more stable solutions. Examples from other designers would also be more plentiful.

There have been many projects with partial reconfiguration, but part of this thesis with two PRRs on the FPGA proved to be a quite troublesome task. There are many examples with two PRRs where the RMs on those areas are different from each other. In this project, we had three waveforms that needed to be implemented on these areas in any combination, which made this project rather unique. Coming up with a flexible solution that could go through every wanted waveform combination with ease proved to be a much more time-consuming endeavor than originally expected.

Vivado has a lot of configurations and rules that can be used to control how it executes different stages of the project like synthesis and implementation. Going through all of those and finding those that could be beneficial to partial reconfiguration would have been too time-consuming. Partial reconfiguration is only a part of the whole picture and most of these configurations are determined by other parts of the project creating restrictions for partial reconfiguration. There were a plenty of different corners that were not explored in the given time frame that could have yielded positive results.

Updates from Xilinx for their software raised its own problems. Vivado Design Suite tool is under a constant development and there were several new version releases during this project. Changing from a version to another had impacts on the project depending on the success of the build and how much the new version updated the software. This created a constant constraint on the project that was under development.

Xilinx has admitted that there can exist differences in the implementations between different operating systems from the same sources. This is a problem with Vivado in general and not just with partial reconfiguration. In this project, there were people who worked on Linux and others who worked on Windows and outcomes from those environments could differ even if the source codes were identical. This made that not all results were comparable with each other between these operating systems.

7. CONCLUSIONS

Wireless communication evolves constantly. Throughput and performance are improved after every new generation and standard. This also increases the complexity and requirements that the developers must take into consider when designing products that use these standards. New techniques and methods are also being developed in order to make it possible to realize these new products. Although the performance increases, the power consumption or size has to stay low.

This thesis focused on partial reconfiguration on Xilinx FPGA in order to realize fast waveform switching on an SDR product and explored other possible benefits as well. Partial reconfiguration was founded to be a compelling and versatile tool in the FPGA's repertoire. The results showed that switching waveforms with this method was possible and could carry it out in the wanted time frame.

The final and stable partial reconfiguration implementations for the given SDR platforms were not fully finished during work of this thesis due to other priorities in the project. However, the proof of concept with benefits were demonstrated and the SDR development project proceeds to utilize partial reconfiguration as one key feature of the SDR products.

For the future, partial reconfiguration could be expanded upon to achieve different goals. For example, Xilinx hinted about a possibility of creating a static implementation with a PRR but without the RMs. This implementation could be distributed without the risk of compromising the original RMs that could not be shared with other developers for a reason or another.

In the case of SDRs, the next logical step right now, which is under development on this field, would be cognitive radio (CR). These radios could automatically do what SDRs do right now. CR could listen to signals and change its configurations appropriately in order to interpret the given signals. It could also learn from its adaptations and change its functionalities accordingly [2]. Partial reconfiguration could be a powerful tool for this kind of radios as well, adding new ways how this automatic reconfiguration could occur.

8. REFERENCES

[1] Brannon B (2004) Software-defined radio. In: Dowla F (ed) Handbook of RF and wireless technologies. Burlington, MA, Elsevier: 133-179. DOI: https://dx.doi.org/10.1016/B978-075067695-3/50007-0.

[2] Ulversøy T (2010) Software defined radio: challenges and opportunities. IEEE Communications Surveys & Tutorials 12(4): 531-550. DOI: https://dx.doi.org/10.1109/SURV.2010.032910.00019.

[3] Lin Y, Lee H, Who M, Harel Y, Mahlke S, Mudge T, Chakrabarti C & Flautner K (2006) SODA: a low-power architecture for software radio. Proc. 2006 International Symposium on Computer Architecture. Boston, Massachusetts, USA, 89-101. DOI: https://dx.doi.org/10.1109/ISCA.2006.37.

[4] Tan K, Liu H, Zhang J, Zhang Y, Fang J & Voelker G (2011) Sora: high-performance software radio using general-purpose multi-core processors. Communications of the ACM 54(1): 99-107. DOI: https://dx.doi.org/10.1145/1866739.1866760.

[5] Cummings M & Haruyama S (1999) FPGA in the software radio. IEEE Communications Magazine 37(2): 108-112. DOI: https://dx.doi.org/10.1109/35.747258.

[6] Bambang R, Langi A, Kurniawan A, Marpanaji E, Mahendra A & Liung T (2015) Software architecture of software-defined radio (SDR). URL: https://www.researchgate.net/publication/268378153.

[7] Akeela R & Dezfouli B (2018) Software-defined radios: architecture, state-of-the-art, and challenges. Computer Communications 128: 106-125. DOI: https://dx.doi.org/10.1016/j.comcom.2018.07.012.

[8] Kazaz T, Praet C, Kulin M, Willemen P & Moerman I (2016) Hardware accelerated SDR platform for adaptive air interfaces. Proc. 2016 ETSI Workshop on Future Network Technologies. Sophia Antipolis, France, 1-10. URL: https://arxiv.org/abs/1705.00115.

[9] Vogt T & When N (2008) A reconfigurable ASIP for convolutional and turbo decoding in an SDR environment. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 16(1): 1309-1320. DOI: https://dx.doi.org/10.1109/TVLSI.2008.2002428.

[10] Safadi M & Ndzi D (2006) Digital hardware choices for software radio (SDR) baseband implementation. Proc. 2006 International Conference on Information & Communication Technologies. Damascus, Syria, 2623-2628 DOI: https://dx.doi.org/10.1109/ICTTA.2006.1684823.

[11] Jayaraman R (2001) (When) will FPGAs kill ASICs? 2001 Design Automation Conference. Las Vegas, Nevada, USA. URL: https://www.doc.ic.ac.uk/~wl/teachlocal/arch/killasic.pdf.

[12] Lau D, Blackburn J & Seely J (2005) The use of hardware acceleration in SDR waveforms. Proc. 2005 Software Defined Radio Technical Conference and Product Exposition. Orlando, Florida, USA. URL: https://www.wirelessinnovation.org/assets/Proceedings/2005/2005-sdr05-1-6-03-lau.pdf.

[13] Srinivasan V, Govindarajan S & Vemuri R (2001) Fine-grained and coarse-grained behavioral partitioning with effective utilization of memory and design space exploration for multi-FPGA architectures. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 9(1): 140-158. DOI: https://dx.doi.org/10.1109/92.920829.

[14] Yu C, Lamoureux J, Wilton S, Leong P & Luk W (2008) The coarse-grained/fine-grained logic interface in FPGAs with embedded floating-point arithmetic units. International Journal of Reconfigurable Computing 2008: article ID 736203. DOI: https://dx.doi.org/10.1155/2008/736203.

[15] Kao C (2005) Benefits of partial reconfiguration. Xcell Journal (55): 65-67. URL: https://www.xilinx.com/publications/archives/xcell/Xcell55.pdf.

[16] Lie W & Feng-yan W (2009) Dynamic partial reconfiguration in FPGAs. Proc. 2009 International Symposium on Intelligent Information Technology Application. Shanghai, China, 445-448. DOI: https://dx.doi.org/10.1109/IITA.2009.334.

[17] Lim D & Peattie M (2002) Two flows for partial reconfiguration: module based or small bit manipulations. Xilinx Application Note 290 (v1.0). URL: https://www.cs.york.ac.uk/rts/docs/Xilinx-datasource-2003-q1/appnotes/xapp290.pdf.

[18] Sadek A, Mostafa H, Nassar A & Ismail Y (2017) Towards the implementation of multi-band multi-standard software-defined radio using dynamic partial reconfiguration. International Journal of Communications Systems 30(17). DOI: https://dx.doi.org/10.1002/dac.3342.

[19] Horta E, Lockwood J, Taylor D & Parlour D (2002) Dynamic hardware plugins in an FPGA with partial run-time reconfiguration. Proc. 2002 Design Automation Conference. New Orleans, Louisiana, USA, 343-348. DOI: https://dx.doi.org/10.1145/513918.514007.

[20] Horta E & Lockwood J (2001) PARBIT: A tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (FPGAs). Technical report. Washington University, Department of Computer Sciene and Engineering. DOI: https://dx.doi.org/10.7936/K7T43R9B.

[21] Vipin K & Fahmy S (2018) FPGA dynamic and partial reconfiguration: a survey of architectures, methods, and applications. ACM Computing Surveys 51(4): article 72. DOI: https://dx.doi.org/10.1145/3193827.

[22] Xilinx (2009) Form 10-K. URL: http://edgar.secdatabase.com/1217/120677409001145/filing-main.htm.

[23] Xilinx (2016) Form 10-K. URL: http://edgar.secdatabase.com/2687/74398816000063/filing-main.htm.

[24] Xilinx (2019) UG909: Vivado design suite user guide: partial reconfiguration. Version 2019.1. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug909-vivado-partial-reconfiguration.pdf.

[25] Hill T (2015) Orthogonal polynomial modulation for higher data rates in 5G mobile communications. Master's thesis. University of Surrey, Faculty of Engineering and Physical Sciences. DOI: https://dx.doi.org/10.13140/RG.2.2.11454.54084.

[26] Poole I (n.d.) Quadrature amplitude modulation, QAM. Electronics Notes. URL : https://www.electronics-notes.com/articles/radio/modulation/quadrature-amplitude-modulation-what-is-qam-basics.php. Accessed: 11.3.2019.

[27] Kamaleldin A, Mohamed A, Nagy A, Gamal Y, Shalash A, Ismail Y & Mostafa H (2017) Design guidelines for the high-speed dynamic partial reconfiguration based software defined radio implementations on Xilinx Zynq FPGA. DOI: https://dx.doi.org/10.1109/ISCAS.2017.8050456.

[28] Xilinx (2019) UG901: Vivado design suite user guide: synthesis. Version 2019.1. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_1/ug901-vivado-synthesis.pdf.