Optimizing FPGA Resource Allocation in SDR Remote Laboratories via Partial Reconfiguration

Zhiyun Zhang

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Electrical Engineering

University of Washington

2024

Committee:

Rania Hussein

Tai-Chang Chen

Program Authorized to Offer Degree:

Electrical and Computer Engineering

University of Washington

Abstract

Optimizing FPGA Resource Allocation in SDR Remote Laboratories via Partial Reconfiguration

Zhiyun Zhang

Chair of the Supervisory Committee:

Rania Hussein

Department of Electrical and Computer Engineering

Software-Defined Radio (SDR) remote laboratories provide engineering students in wireless communications and radio frequency courses with hands-on experience using SDRs. These devices are renowned for their flexibility and reconfigurability via software. However, cost-effective SDRs often feature lower-end System-on-Chips (SoCs) with Field Programmable Gate Arrays (FPGAs) capable of parallel data processing but limited in hardware resources for running complex digital signal processing algorithms. This limitation restricts their use to simpler FPGA-based tasks, reducing their operational flexibility. Partial Reconfiguration (PR) offers a compelling solution to these limitations by dynamically allocating hardware resources based on the system's operational mode, enhancing the FPGA's functionality. PR allows the execution of complex programs by enabling independent modifications, recompilation, and reconfiguration of specific FPGA regions without requiring a full project recompile. This process

streamlines modifications, shortens development cycles, and enables rapid iteration, significantly improving conventional FPGA programming techniques. This thesis investigates the implementation of PR within SDRs, specifically on Red Pitaya's SDR platforms which, to the best of our knowledge, have not been previously used for PR. It also explores the integration of this implementation into the existing structure of an SDR remote laboratory. Experimental results demonstrate notable improvements in hardware efficiency, including reductions in logic resource utilization and total power consumption, compared to traditional FPGA reconfiguration methods. These findings underscore the potential of PR in advancing the field of SDR.

Acknowledgements

First and foremost, I would like to express my deepest appreciation to my advisor, Dr. Rania Hussein. Her steadfast encouragement and support have been pivotal to my journey. Joining her research laboratory opened doors to invaluable resources and the opportunity to collaborate with like-minded peers, enriching my academic and personal growth immeasurably. I am profoundly grateful for her visionary guidance. While the path she charted for me initially seemed uncertain to me, every step she orchestrated has led me toward success, laying the foundation for my future endeavors.

I would also like to extend my heartfelt appreciation to Dr. Tai-Chang Chen for serving on my thesis committee and for being an exceptional professor throughout my academic journey. His classes have been incredibly enriching, and his patience and exemplary teaching methods have been instrumental in fostering my interest and motivation to pursue a career in electrical engineering. I would also like to extend my gratitude to ECE faculty and staff for the support for my educational journey at the University of Washington.

I am grateful to all members of the RHL-RELIA research team, with a special mention of Marcos Inoñan. His insight into the topic of Partial Reconfiguration sparked my interest, leading me to pursue it as the subject of my master's thesis. Before delving into Partial Reconfiguration, Marcos generously shared his extensive knowledge of radios and signal processing with me, a contribution that has been invaluable.

Many thanks to Dr. Pablo Orduña, co-founder of LabsLand and our industry partner for the RHL-RELIA project. Dr. Orduña's mentorship and provision of essential resources and tools have been crucial in transforming the RELIA project from a theoretical concept to an implemented solution.

I cannot conclude without expressing my profound gratitude to my family and friends, whose unwavering support and constant encouragement have been indispensable throughout this journey. Their belief in me has provided immense motivation and strength, allowing me to navigate challenges with resilience.

This work is conducted as a part of the RHL-RELIA project, which is funded by the National Science Foundation's Division of Undergraduate Education, under Grant No. 2141798.

TABLE OF CONTENTS

TABLE OF FIGURES

Chapter 1. INTRODUCTION

Partial Reconfiguration (PR) stands out as a sophisticated technique that boosts the functionality of Field Programmable Gate Arrays (FPGAs) by optimizing their hardware resource utilization. This capability is particularly crucial for Software-Defined Radios (SDRs), which often require extensive hardware resources for complex digital signal processing tasks. As SDRs have gained widespread interest from both the academic and industrial sectors due to their superior flexibility and programmability over conventional radios, the integration of PR can significantly enhance their functionality. These advantages allow students and professionals in radio and wireless communication to rapidly prototype and develop radio systems. However, despite a decrease in costs over the past two decades—with prices now ranging from 200 to 3000 USD—the most affordable SDRs often come with FPGAs that struggle with complex tasks due to limited capabilities. This section will explore several key concepts essential to understanding this thesis and discuss the motivations behind exploring the integration of PR into an existing SDR remote laboratory.

1.1 Software-Defined Radio (SDR)

Traditional radios, known as hardware-defined radios (HDRs), rely on fixed hardware circuits to process signals. These devices typically incorporate various hardware components, including filters, to directly manipulate analog signals. Once manufactured, these circuits are not easily modifiable, limiting the flexibility of HDRs. In contrast, SDRs digitize analog signals at the outset, employing general-purpose processors or FPGAs to conduct digital signal processing (DSP) tasks [1, 2]. Figure 1 provides a visual comparison of the internal structures of an SDR and an HDR receiver [3]. Conversely, the signal path architecture of SDR and HDR transmitters

is mirrored, reflecting the reverse flow in their processes.

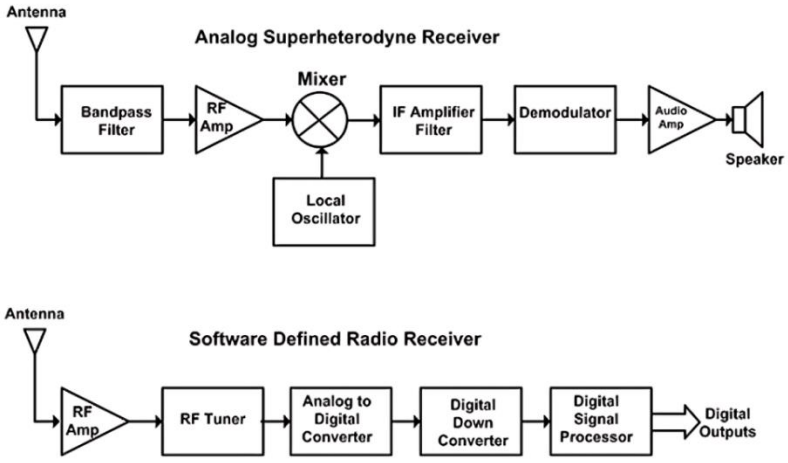


Figure 1. A comparison of an HDR receiver (Top) versus an SDR receiver (Bottom).

This distinct characteristic endows SDRs with unparalleled flexibility and reconfigurability, as they can be reprogrammed using software tools. One such tool is GNU Radio, a user-friendly application that enables users to assemble signal processing blocks into flowcharts for programming SDRs [4]. This capability accelerates prototyping and reduces costs relative to the traditional HDR approach, which necessitates dedicated hardware circuits for each functionality. The inherent adaptability of SDRs, facilitated by software tools, marks a substantial advancement over the more rigid, hardware-centric design of traditional radios [5, 6].

1.2 Zynq 7000 SoCs

Many SDRs on the market feature the hybrid reconfigurable Zynq-7000 All Programmable (AP) SoCs from Xilinx, which combine a Processing System (PS), typically running Linux, with Programmable Logic (PL), in the form of an FPGA [7]. The PS, powered by an ARM Cortex A9 processor, interacts with memory and peripherals via the Advanced eXtensible Interface (AXI). Meanwhile, the PL, leveraging the 7-series FPGA architecture, enables parallel data processing and supports partial reconfiguration [8].

Most economically priced SDRs, often priced below 1,000 USD, tend to employ the Z-7010 and Z-7020 SoCs. These models offer minimal hardware resources, making them unsuitable for large-scale programs. Given the demanding nature of digital signal processing tasks common in SDR applications, these two SoCs fall short for complex signal processing due to their limited number of hardware resources. Figure 2 offers a comparative analysis of the FPGA hardware resources between the Z-7010 and the Z-7100, the latter being a more advanced model within the Z-7000 family. It shows that the Z-7010 possesses only about 6% of the logic cells and 4% of the DSP Slices found in the higher-end Z-7100 model [9].

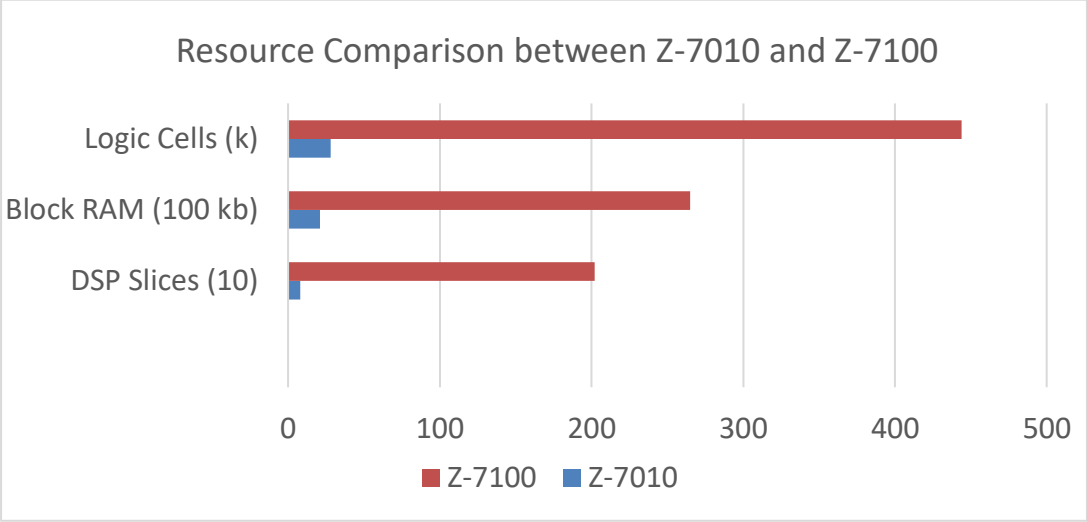


Figure 2. Comparison of FPGA hardware resources between Z-7010 and Z-7100 SoCs within the Z-7000 series, demonstrating the Z-7010's significantly lower logic cells, block RAM, and DSP slices.

1.3 Partial Reconfiguration (PR)

Partial Reconfiguration (PR) stands as an advanced approach in the field of FPGA technology, particularly addressing the hardware limitations of lower-end SoC models like the Z-7010 and Z-7020 used in affordable SDRs. PR allows for the dynamic modification of specific sections of the FPGA without needing to reconfigure the entire FPGA [10]. Different

functionalities can be loaded and executed on-the-fly based on the current requirements of the application [11-13]. In contrast, traditional FPGA programming approaches demand that all functionalities be pre-loaded onto the FPGA, leading to underutilization of the device's hardware resources for the majority of the operational period.

Moreover, the utilization of PR accelerates the development process and iteration cycles for FPGA-based projects. Unlike traditional FPGA programming, which demands a complete synthesis, implementation, and configuration cycle for any change, PR modules streamline this process. Utilizing PR, updates to a designated partial reconfigurable FPGA portion require only that specific slice to undergo the processes of re-synthesis, re-implementation, and re-configuration, while the remaining FPGA portions are unchanged. This localized approach to reconfiguration bypasses the time-intensive full device recompilation, enabling rapid prototyping and testing of new functionalities or improvements. This reduction in development time is advantageous in research and development settings, where swift adaptation and testing of changes are crucial. PR not only enhances the functional capabilities of FPGAs in SDRs but also facilitates a degree of development flexibility not possible with conventional FPGA programming techniques. This feature, often overlooked by academia and industry, holds significant potential and has found applications across diverse domains, including aerospace, automotive, and machine learning.

Figure 3 illustrates a basic application of PR in a calculator program capable of addition and subtraction, with inputs from onboard switches and outputs on LEDs. The program comprises two parts: a static region, in green, and a partial reconfigurable region, in pink. The reconfigurable region allows for swapping between an adder and a subtractor module based on the required operation. Loading the adder module enables addition, while loading the subtractor

module facilitates subtraction. Utilizing PR conserves FPGA resources by only requiring the presence of one operational module in the FPGA at any given time, instead of both.

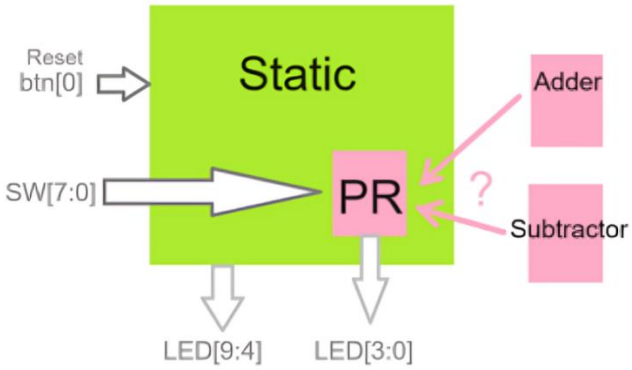


Figure 3. Diagram of a partial reconfigurable calculator application, showcasing a static region (green) for constant functionality and a reconfigurable region (pink) for interchangeable addition and subtraction modules, demonstrating efficient FPGA resource usage through dynamic module swapping.

1.3.1 Dynamic Partial Reconfiguration (DPR)

Dynamic Partial Reconfiguration (DPR) represents a specialized variant of PR. It enables the partial reconfiguration of an FPGA while the static region of the program continues to execute. Implementing this method necessitates meticulous program design to prevent the main program from processing incorrect data originating from the partial reconfigurable regions (PRRs) during their update. This precaution helps avoid unexpected behavior in the overall system.

1.3.2 Dynamic Partial Reconfiguration Case Study: Automative Headlights and Robotic Navigation System

In a study conducted by researchers at Carnegie Mellon University, two applications were developed to compare dynamic versus static FPGA mapping [14]. The first, an interactive system for automotive headlights, enables users to select among four environmental settings to adjust the

system's functionality accordingly. Additionally, a navigation application for robotic systems was developed to leverage partial reconfiguration. The findings revealed that DPR significantly enhanced efficiency, reducing power consumption by 28% to 30% and decreasing logic resource usage by 2.5 to 3.2 times. This efficiency was attributed to the capability of running complex applications on smaller FPGAs by dynamically adjusting resources, contrasting with the less efficient approach of using a larger FPGA to accommodate all modules statically.

1.4 Remote Laboratories

Traditional engineering courses often require students to participate in in-person laboratory sections for hands-on experience with hardware, a practice that comes with several drawbacks. Firstly, it may exclude students with physical or mental challenges that hinder their ability to attend lab sections, limiting their access to equal educational opportunities. Secondly, the fixed timing of lab sections may not accommodate all students' schedules, particularly those with daytime employment or occasional conflicts, potentially forcing them to drop the course or fall behind [15]. Lastly, educational institutions and instruction teams face the challenge of maintaining a large number of physical lab kits. These kits are frequently underutilized, vulnerable to accidental damage, or, in some cases, complete destruction. Given that in-person lab sessions typically occur only for a few hours each week, the majority of the time, these resources remain unused. The issue of maintenance and care is exacerbated when they are costly and fragile [16].

Remote laboratories address these drawbacks effectively, enhancing educational equity by enabling students from all backgrounds to access physical lab kits from anywhere [17-19]. Their flexibility ensures that physical lab resources are available outside traditional in-person lab hours, broadening accessibility [20-22]. Furthermore, by housing physical lab kits in a remote

setup, educational institutions and instructional teams alleviate the burden of constant maintenance and calibration, streamlining the management of these resources.

Despite initial skepticism about the effectiveness and efficiency of substituting in-person laboratories with remote laboratories, research has demonstrated their comparable performance, with some studies even reporting enhanced performance through remote lab usage [23-29]. For instance, a study in an FPGA course at the University of Washington revealed that students using a remote lab not only performed better but were also more likely to complete assignments, attributed to the greater flexibility offered by remote lab environments [30].

1.5 Implementing Partial Reconfiguration in an SDR Remote Laboratory

Although it serves as an efficient method to decrease reconfiguration time and optimize hardware resource usage in SDRs, to the best of our knowledge at the time of writing, there are no SDR remote laboratories that provide students with the opportunity to investigate and apply PR on SDRs. To fill this gap, this thesis documents our efforts in assessing the impact of PR on cost-effective SDRs and delves into the potential for its integration with an existing SDR remote laboratory setup. By doing so, our aim is to make this valuable technique accessible to more students, enabling them to harness PR in practical applications and propose a pathway for integrating advanced FPGA functionalities into the FPGA and SDR curricula.

1.6 Contributions of the Thesis

This thesis offers several contributions to the development and optimization of SDR remote laboratories, advancing the integration of PR into this emerging field. Firstly, it investigates the practicality and advantages of utilizing PR in enhancing the digital signal processing aspects of SDRs, providing a foundation for further exploration. Next, the thesis

proposes two architectures aimed at integrating PR functionality within the infrastructure of an existing SDR remote laboratory, thereby expanding the laboratory's capabilities. Finally, it delves into the process of selecting an appropriate SDR platform suitable for PR, discussing how the selected SDR benefits from the application of PR in terms of performance and flexibility.

This work pioneers new ground in an area where existing research is sparse, marking a step forward in the understanding and application of PR within SDR contexts. Furthermore, the thesis acknowledges the limitations of the current study and outlines potential directions for future research. It highlights areas where subsequent investigations could not only validate and extend the findings of this thesis but also explore new avenues for enhancing the utility and efficiency of SDR remote laboratories through advanced PR techniques. This perspective aims to inspire continued innovation and deeper inquiry into the integration of PR in SDR technologies.

Chapter 2. PREVIOUS WORK

There are several projects that have explored performing Partial Reconfiguration (PR) on Software-Defined Radios (SDRs), and a few have attempted to integrate PR into remote laboratories [14, 31-36]. Three notable projects are summarized below.

2.1 Implementing Hardware Multitasking with Partial Reconfiguration in a Remote Laboratory

In the first project [37], a research team from the University of Brasov designed and implemented an online digital hardware design remote laboratory that utilizes a Web application with a user interface, a Web server responsible for scheduling and managing hardware devices, and a network of ZedBoard, a popular FPGA development board. This virtual lab allows up to 4 users to share the same development board simultaneously through DPR. This approach improves resource sharing effectiveness and provides students with practical experience in digital hardware design and the use of hardware for running mathematical models of renewable energy sources. The lab's effectiveness in enhancing learning outcomes and resource utilization was also evaluated in the paper, demonstrating technical feasibility and positive educational impacts.

2.2 Reconfiguration Time with Partial Reconfiguration in SDRs

One of the first experiments in applying PR on SDRs in [38] utilized the USRP E310 SDR. The research team's findings, presented in their paper and at a conference, revealed that employing PR in their FPGA design significantly accelerated the reconfiguration process. Specifically, they found that performing a PR was over four times quicker than the conventional

full reconfiguration approach, cutting down the reconfiguration time from 143 milliseconds to 33 milliseconds.

2.3 Reducing Resource Use and Power Consumption with Dynamic Partial Reconfiguration in SDRs

In a different study in [39], the research team evaluated DPR in SDRs. They established five distinct communication technology chains for FPGAs: Wi-Fi, Bluetooth, 2G, 3G, and LTE. They assessed the feasibility of DPR on these SDRs with a focus on resource utilization and power consumption. The findings were encouraging; the testing demonstrated that DPR could reduce FPGA area utilization by 10.19% and lower the total power by 76.71% compared to systems not utilizing DPR.

As demonstrated by the aforementioned projects, PR can reduce hardware resource utilization across various applications on different SDR platforms. However, we are not aware of any remote laboratories that currently facilitate the execution of PR on remote SDRs. Additionally, we found no resources or literature on implementing PR on the Red Pitaya, a popular and affordable SDR platform that will be discussed in detail in the following section. These gaps underscore the unique nature and academic contributions of this thesis.

Chapter 3. METHOD

3.1 Partial Reconfiguration FPGA Design Flow

For Zynq 7000 Series FPGA design utilizing Vivado Design Suite, the official development software for Zynq FPGAs, the common design process for developing an FPGA program without incorporating partial reconfiguration can be summarized as follows [40]:

1) Develop the FPGA program by either writing scripts or utilizing Intellectual Property (IP) blocks, which are then interconnected in a block design. Running an FPGA program on a physical device necessitates both source files, written in Hardware Description Language (HDL) that outlines all the logic and circuitry, and constraint files that provide essential pinout details and other synthesizer-required information for mapping the circuit onto the FPGA.

2) Synthesize the project to identify any syntax errors and convert it into a "netlist", which enumerates the necessary circuit components.

3) Implement the synthesized design on a chosen chip as specified by the programmers.

4) Generate a bitstream, translating the implemented design into a .bit file that can be loaded onto an FPGA.

5) Program the FPGA by configuring it with the generated bitstream from the preceding step.

The design process for developing a partial reconfigurable FPGA program with Vivado Design Suite is similar to that of standard, non-partial reconfigurable programs, with the inclusion of extra steps during the development, synthesis, and programming stages, as shown in Figure 4 [41].

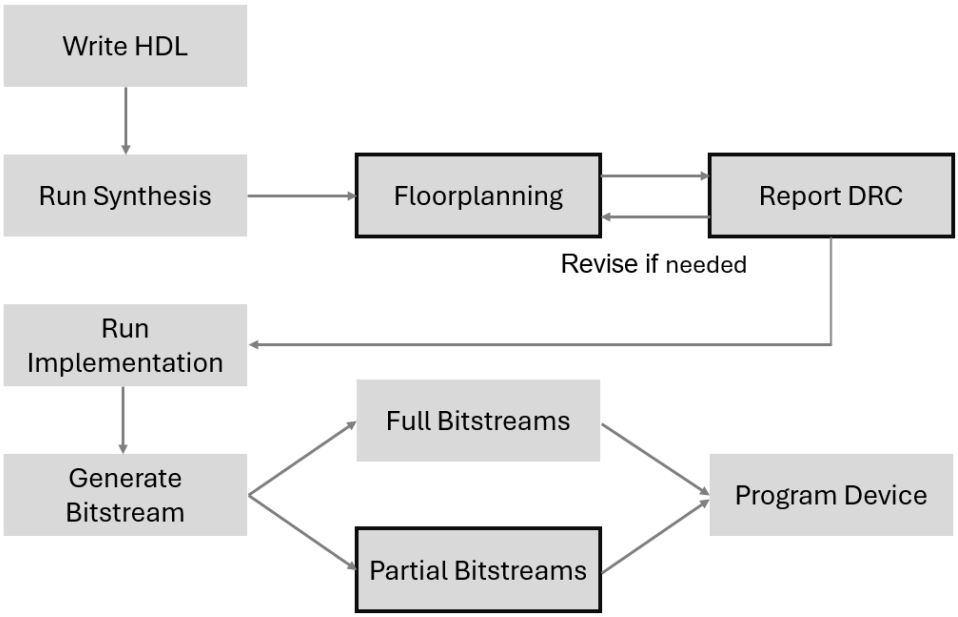


Figure 4. Design Flow for partial reconfigurable FPGA programs using Vivado Design Suite. Steps specific to PR are boxed.

Before sending the circuit design to the synthesizer, users need to activate "Enable Dynamic Function eXchange" from the Tools menu. Subsequently, a partition can be created by right-clicking on any HDL source file and selecting "Create Partition Definition". Upon naming this specific partition, users may then access the "Dynamic Function eXchange Wizard" via the left navigation pane. Within the wizard, it's possible to associate multiple source files with a partition. These source files then become children of the design project and are designated for programming the partial reconfigurable region (PRR) defined during the synthesis step. The design can then be synthesized.

Once the project has been successfully synthesized, users need to establish a Pblock for each partition by utilizing the Floorplanning feature [42]. This process involves carefully selecting specific regions on the FPGA that will accommodate the partial reconfigurable modules. Subsequently, the floorplanning must undergo a Design Rule Check (DRC) to ensure that the

design complies with the Dynamic Function eXchange (DFX) design rules. Following this, the project can be compiled into bitstreams [43].

Once the bitstreams have been generated, users will find multiple full and partial .bit files within the runs folders or their subfolders. Each of these files can be utilized to program the FPGA. However, attempting to program FPGAs with partial bitstreams before the FPGA is programmed using a full bitstream through Vivado's Hardware Manager will result in an error and fail to work.

3.2 Assessment of Partial Reconfiguration on a Zynq Development Board

Before delving into Partial Reconfiguration on an SDR, we conducted feasibility tests using a development board called the Blackboard, developed by Real Digital [44]. This board incorporates a Zynq Z-7007S SoC from the Zynq 7000 family, alongside various peripherals such as LEDs, switches, XADC, HDMI, and PMOD ports. While the Z-7007S SoC is slightly less powerful than the Z-7010 SoC typically found in more affordable SDRs, it still mirrors most of their capabilities at a reduced cost, making Blackboard a cost-effective choice. This approach not only minimizes potential waste should PR not be viable for SDRs but also facilitates a deeper understanding of the Zynq architecture, thanks to the comprehensive tutorials provided by the manufacturer.

3.2.1 Simple PR Calculator Program

The first evaluation of PR utilized a simple 8-bit calculator program capable of executing addition, subtraction, multiplication, and division on two 8-bit operands. The operands were input using onboard switches, with the calculation results being displayed on the onboard 7-segment displays.

During the Floorplanning process for the partial reconfigurable module, we faced several challenges typical of working with PR. The PRR needs to be carefully sized—small enough to conserve hardware resources, yet large enough to accommodate the largest PR module and allow for the inclusion of potentially larger future modules. Initially, we selected a small region that seemed sufficient for the largest existing PR module, as depicted in Figure 5. However, upon running the Design Rule Check (DRC), we encountered multiple errors indicating that the designated PRR was too small to house all the necessary hardware resources, as Figure 6 illustrates. This is contrary to what was indicated in the property window of the selected PR region, as Figure 7 shows.

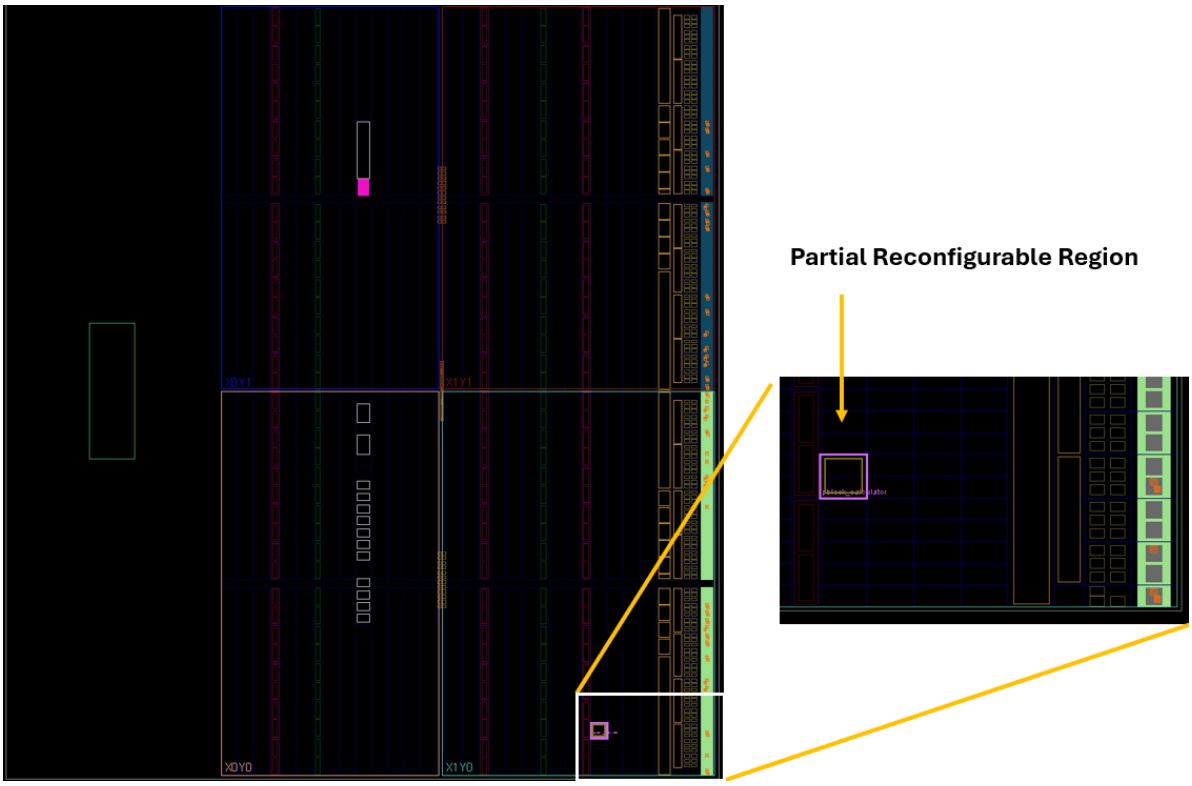


Figure 5. Initial Floorplanning for the partial reconfigurable calculator modules.

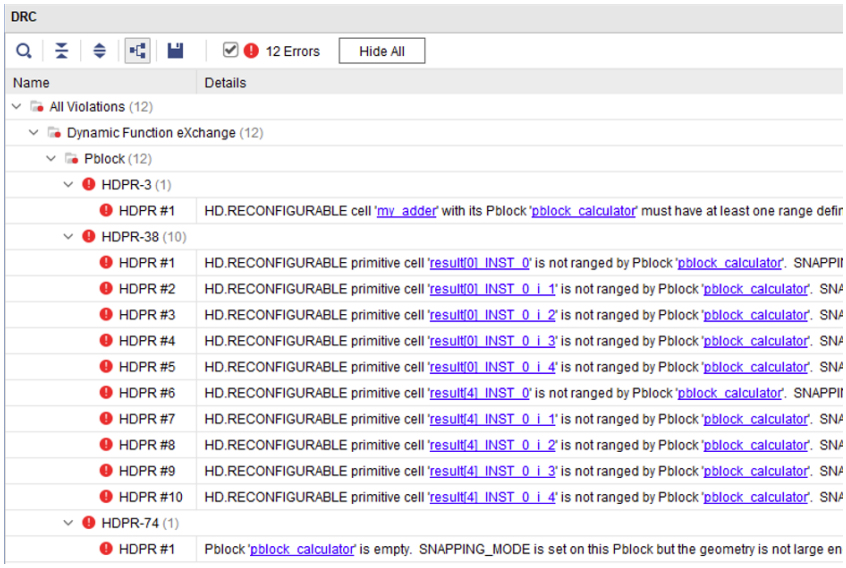


Figure 6. DRC rules were violated when the selected partial reconfigurable region (PRR) is not large enough to contain all the required hardware.

Pblock Properties

pblock_calculator

Physical Resource Estimates

Site Type	Parent	Child	Non-Assigned	Used	Available	% Util
Slice LUTs	8	0	0	8	16	50
LUT as Logic	8	0	0	8	16	50
Slice Registers	0	0	0	0	32	0
Register as Flip Flop	0	0	0	0	32	0
Register as Latch	0	0	0	0	32	0
F7 Muxes	0	0	0	0	8	0
F8 Muxes	0	0	0	0	4	0

Figure 7. The Pblock, based on its properties, was assumed to house enough hardware resources for the partial reconfigurable modules while allocating a Partial Reconfigurable Region (PRR).

All violations were resolved upon choosing a larger Pblock with additional resources. This Pblock was minimized to the smallest size possible without incurring any DRC violations. The final configuration of the Pblock is depicted in Figure 8.

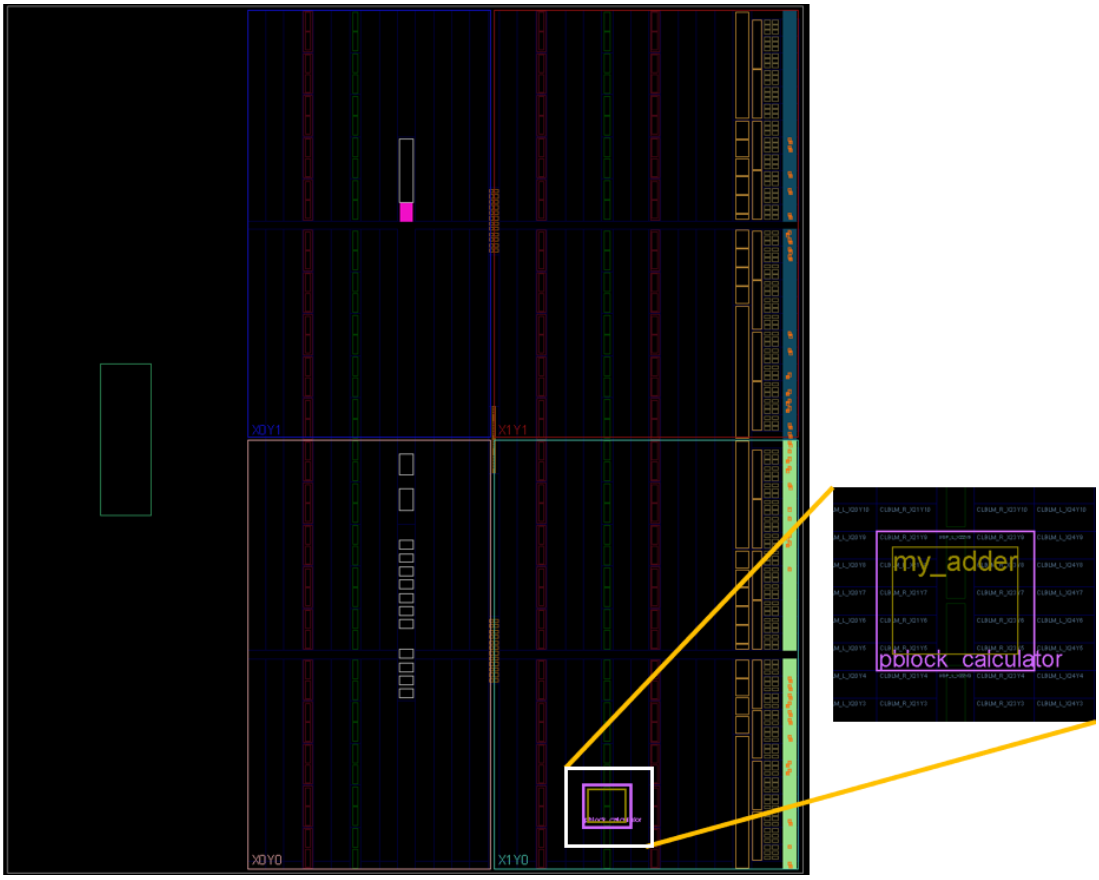
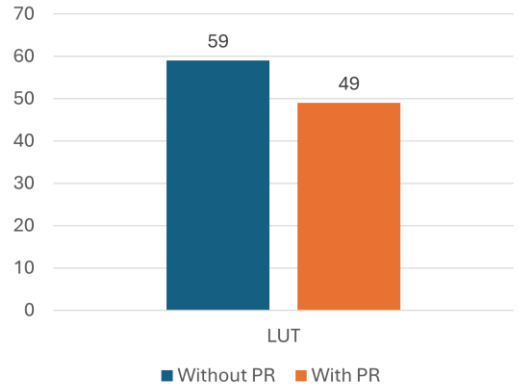


Figure 8. Final Pblock Configuration for the Partial Reconfigurable module. The size has been increased, and it adheres to all Design Rule Check rules.

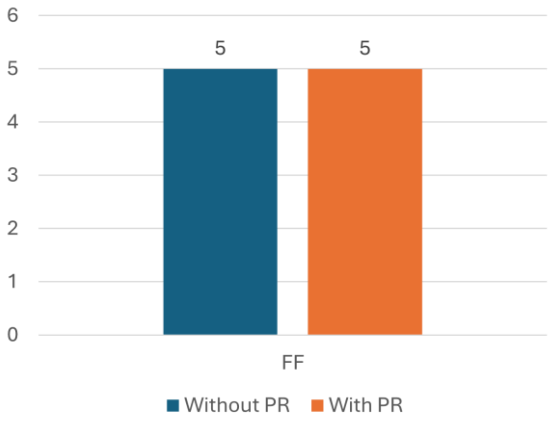
Additionally, a control group version of the calculator program was developed without utilizing PR. This version encompasses all four computation modules: adder, subtractor, multiplier, and divider, each designed to process calculations involving two 8-bit operands. Unlike the approach of swapping PR modules through Vivado's Hardware Manager, the functionality of this program is controlled by the states of two onboard switches. These switches are not employed in the previously mentioned PR version of the program.

After implementing both programs and generating their respective bitstreams, data concerning hardware resource usage and total power were gathered. Key hardware resources, including LUTs, FFs, BRAM, and DSP Slices, are compared across all case studies discussed in

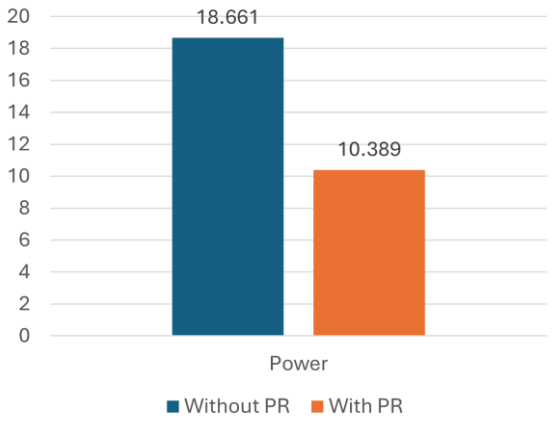
this paper. Figure 9 provides a graphical representation of these comparisons for the simple 8-bit calculator program on Blackboard.



(a)



(b)



(c)

Figure 9. LUT and FF utilization and total power comparison of the 8-bit calculator program on Blackboard. (a) LUT utilization. (b) FF utilization. (c) Total power.

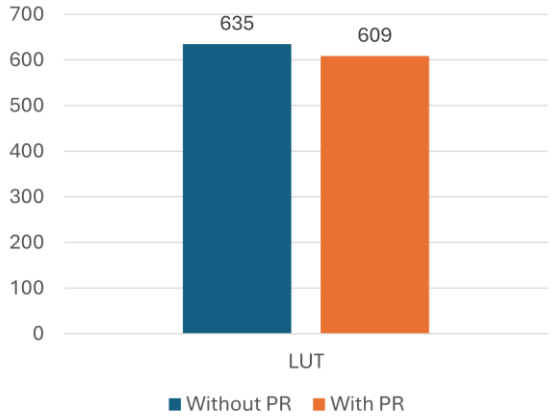
Although both programs exhibited nearly identical functionalities, enabling PR resulted in significant enhancements in terms of LUT utilization and power efficiency. Specifically, the program utilizing PR demonstrated a reduction of over 16.9% in LUTs, decreasing from 59 to 49, and achieved a notable decrease of over 44.3% in total power consumption, falling from 18.661 W to 10.289 W at the maximum theoretical clock speed. Both programs used 5 FF. Since BRAM and DSP slices were not utilized in these programs, comparisons and data for these components are not presented.

3.2.2 XADC PR Calculator Program

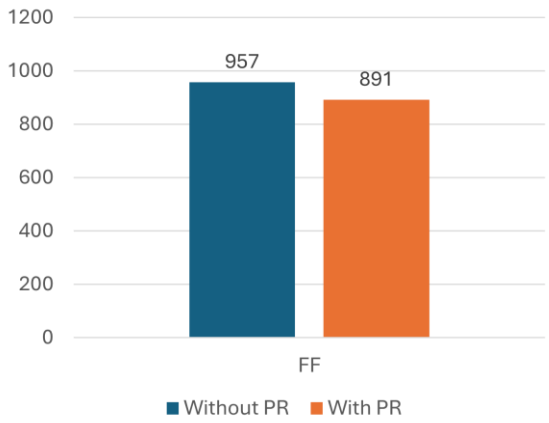
In another comparison designed to incorporate basic signal processing, programs were set to receive input from the onboard potentiometer connected to the XADC on the Blackboard, processing it through either a low-pass or high-pass filter. For the program with PR enabled, a PRR was allocated during floorplanning to accommodate either one of these PR modules at any given time. For the comparative non-PR program, we implemented manual mode switching by

writing to a register in the SoC, simulating a more SDR-like control mechanism where adjustments are predominantly made through register modifications rather than physical switches.

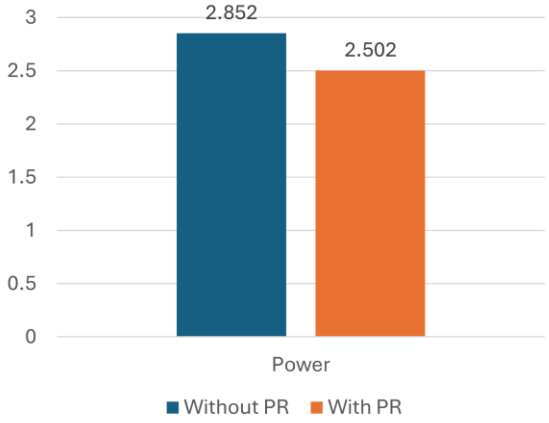
Data on hardware resource utilization and power consumption were once again gathered and are presented in Figure 10. In this set of comparisons, there was a notable reduction in LUT and FF utilization by 4.1% and 6.9%, respectively, alongside a 12.3% decrease in total power consumption. The diminished difference in comparison to the calculator program's results can be attributed to the smaller size disparity between the static and PR modules in this instance, coupled with the use of fewer PR modules—reducing from four in the calculator program to two here. This observation underlines a consistent trend identified through extensive PR evaluations: the presence of more PR modules per PRR leads to more savings in resources, particularly when all PR modules demand similar hardware resources. For this program, no reduction in BRAM and DSP utilization was noted.



(a)



(b)



(c)

Figure 10. LUT and FF utilization and power consumption of the XADC program on Blackboard. (a) LUT utilization. (b) FF utilization. (c) Total power.

Additional tests on the Blackboard yielded similar results to the XADC program, with average savings of 5-8% in LUT and FF utilization and a 10-18% reduction in power consumption. Encouraged by these PR evaluation results, we proceeded to select an appropriate SDR platform for further testing.

3.3 Selecting an SDR

Choosing the right SDR for evaluating partial reconfiguration is critical, as an unsuitable SDR can result in both wasted time and financial resources. The research group we are part of, RHL-RELIA, developed through a collaboration between the Remote Hub Lab (RHL) [45] and LabsLand [46], has successfully set up an SDR remote laboratory. Initially, this laboratory employed the ADALM-PLUTO (also known as PlutoSDR) developed by Analog Devices [47, 48]. The RHL-RELIA laboratory is structured based on the MELODY model, a framework for assessing the efficacy of SDR remote labs [49]. Despite the PlutoSDR's cost-effectiveness and inclusion of a Z-7010 SoC, early attempts at implementing PR revealed its limitations in user-friendliness for PR, primarily due to inadequate connectivity options [50]. The PlutoSDR's lack of connectivity features, such as an Ethernet port, and the requirement to disassemble the casing and alter the circuit board for programming with Vivado, posed challenges.

Following an unfruitful beginning with PlutoSDR, we established four criteria for an ideal SDR platform to facilitate PR effectively:

1) It should be equipped with a Zynq 7000 Series SoC, with a preference for the Z-7010 or Z-7020 models.

2) The platform should be cost-effective (under $750) and widely used.

3) It must be open-source, supported by an active user community, so we can get help when needed.

4) It must include an Ethernet port and microSD card slot. An Ethernet port facilitates remote configuration, while a microSD card slot enables the testing of various firmware versions and provides a straightforward means of device recovery in the event of a corrupt firmware.

Having ruled out various prominent and well-received SDRs, such as the USRP N210 [51], due to their prohibitive costs and limited connectivity features, we selected the Red Pitayas as the optimal platform for our needs. They are popular and met all four of the criteria we had outlined, making them the most suitable choice for our project [52-54]. During the evaluation stage, we worked with two Red Pitaya models: the STEMlab 125-14 [55] and the SDRlab 122-16 [56]. Table 1 showcases a side-by-side comparison of these models. The key differences between them are the SoC used, bandwidth, and RF channels' sampling rates and resolutions. The initial experimentation was conducted with the SDRlab 122-16, owing to its superior capabilities compared to the STEMlab 125-14. However, due to a scarcity of detailed information about the model, subsequent tests were carried out using the more popular and affordable STEMlab 125-14.

Table 1. Red Pitaya STEMlab 125-14 and SDR122-16 Specification

	STEMlab 125-14	SDRlab 122-16
SoC	Z-7010	Z-7020
Connectivity	USB 2.0, 1 Gb Ethernet	
Number of RF Input Channels	2	
Number of RF Output Channels	2	
RF Input Bandwidth	DC – 60 MHz	300 kHz – 550 MHz
RF Output Bandwidth	DC – 60 MHz	300 kHz – 60 MHz
Input Sampling Rate (MS/s)	125	122.88
Output Sampling Rate (MS/s)	125	122.88
Input Resolution (bit)	14	16
Output Resolution (bit)	14	

3.4 Integrating Red Pitaya into an Existing Architecture

Prior to conducting PR tests on the Red Pitaya, it was integrated into our pre-existing remote laboratory framework. This incorporation enhances the interoperability within our remote lab framework, allowing for diverse devices and software to work together [57-60]. Figure 11 illustrates the architecture of RHL-RELIA, highlighting its three main components: a server featuring a web application, a scheduler for hardware usage management, and a data exchanger for processing user configuration files and relaying resultant signals back to them; Raspberry Pis tasked with programming and data retrieval from SDRs; and the physical SDR devices themselves, housed in Faraday cages [48]. Currently, all hardware components are located within an RHL laboratory at the University of Washington.

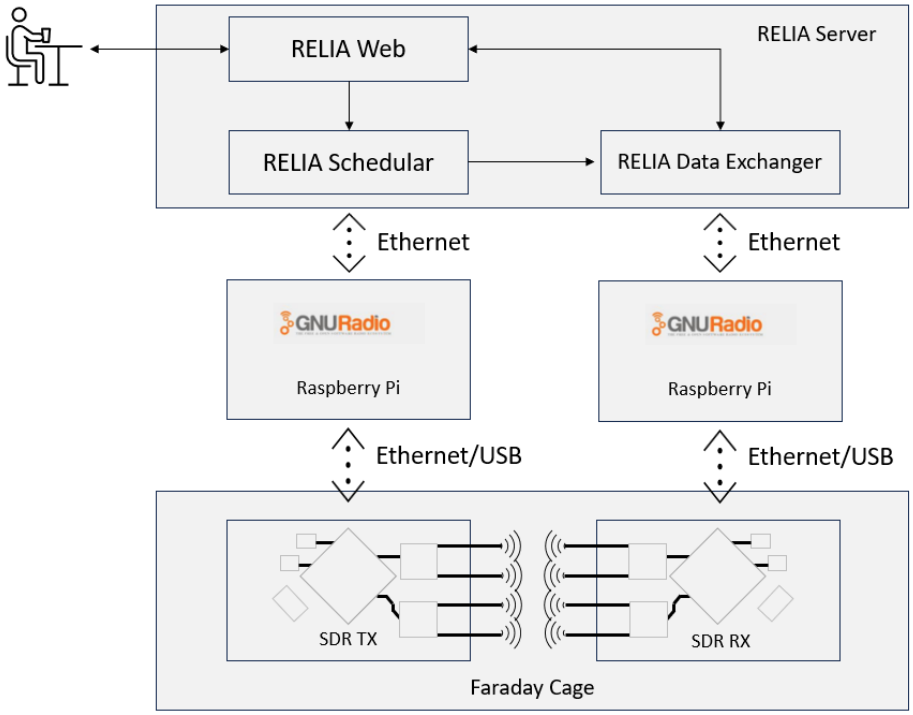


Figure 11. RHL-RELIA framework.

3.4.1 Redesigning Faraday Cages

Switching to a different SDR device necessitates the redesign of the existing enclosure, initially intended to house two PlutoSDRs—one for transmission and one for reception. Each case needs to accommodate two SDRs of the same model, with plans to encase them in a Faraday fabric on all sides except the top, which was to be covered with a semi-transparent Faraday mesh for better visibility under cameras. During our exploration of PR's viability on the PlutoSDR, we undertook the task of redesigning its Faraday cage. The initial version, created using Python, was overly large, making it unsuitable for printing on standard large desktop 3D printers, resulted in excessive printing time and filament use, and lacked aesthetic appeal [61].

The redesigned PlutoSDR case we developed addresses the aforementioned issues effectively [62]. A 3D model of this updated enclosure is illustrated in Figure 12, with a comparison of the new and original versions presented in Figure 13. This revamped case is more compact, securely holds the PlutoSDR with accurately measured walls, and significantly reduces both filament consumption and printing time—by over 37.4% and 64.5%, respectively, using identical slicing parameters. Figure 13 showcases a side-by-side comparison, featuring the original case in purple and the new case in light green.

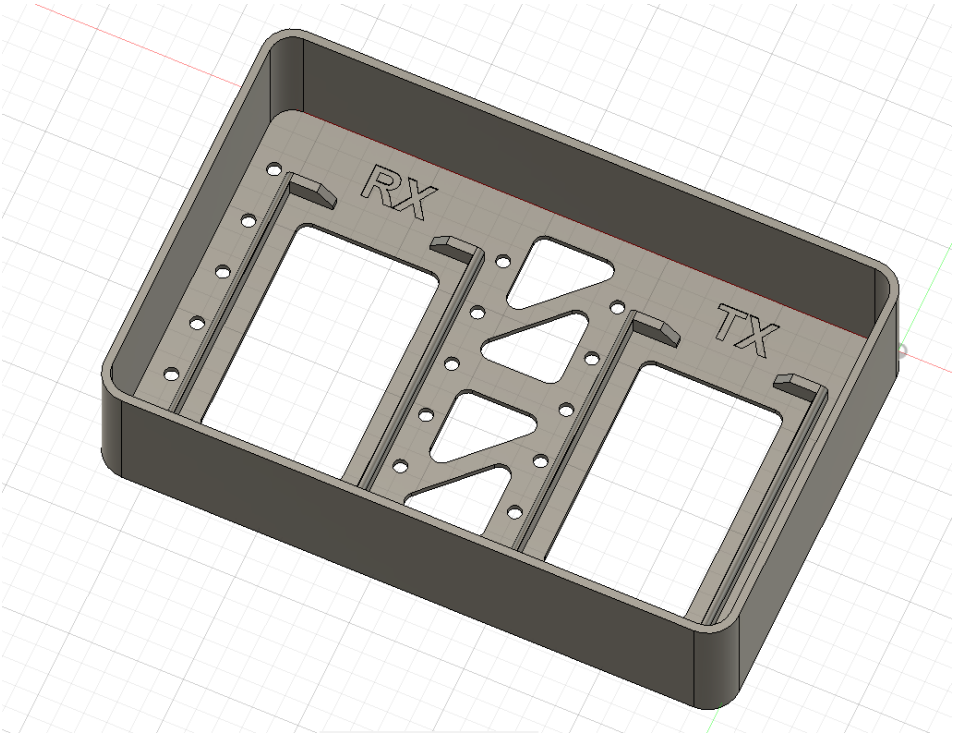


Figure 12. New Faraday cage design for PlutoSDRs. Rectangular and triangular cutouts are implemented to reduce filament usage.

Figure 13. Comparison between the old case in purple and the new case in light green.

Additionally, a Faraday cage was designed to accommodate two Red Pitayas, adopting the same design principles used for the PlutoSDRs, as shown in Figure 14. Given that Red Pitayas are more compact than PlutoSDRs, the resulting case is even smaller. Unlike the PlutoSDRs, Red Pitayas are delivered without any external casing, necessitating the inclusion of four precisely placed holes in the design. These allow the Red Pitayas to be securely fastened to the bottom of the cage using four M3 machine screws.

Figure 15 showcases a physical setup with two Red Pitayas, each enclosed in a case that is covered on all sides except the top. The Red Pitaya on the left functions as a receiver, while the one on the right serves as a transmitter, indicated by their differing antenna connections. Figure 16 depicts the completed installation of a Red Pitaya Faraday cage within a LabsLand Prism4 structure—a robust enclosure capable of housing 4 Raspberry Pis, 4 SDRs within 2 Faraday cages, a power supply, among other components. The Faraday cages' top is now fitted with a semi-transparent Faraday material, allowing the Red Pitayas to be visible through monitoring cameras mounted on the ceiling of the LabsLand Prism4 structures.

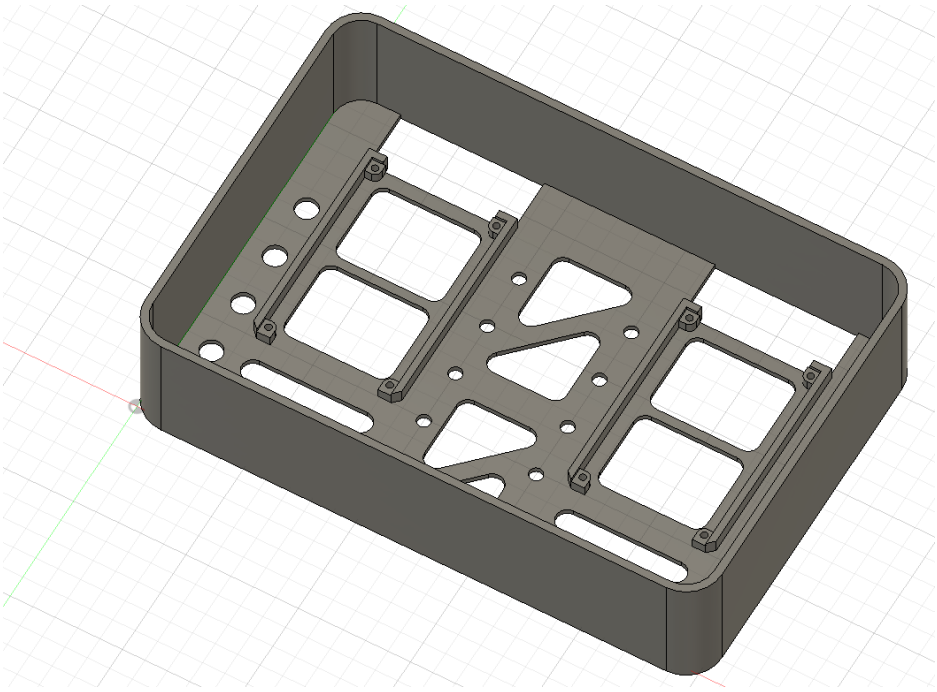


Figure 14. Faraday cage design for Red Pitayas. Rectangular and triangular cutouts are implemented to reduce filament usage.

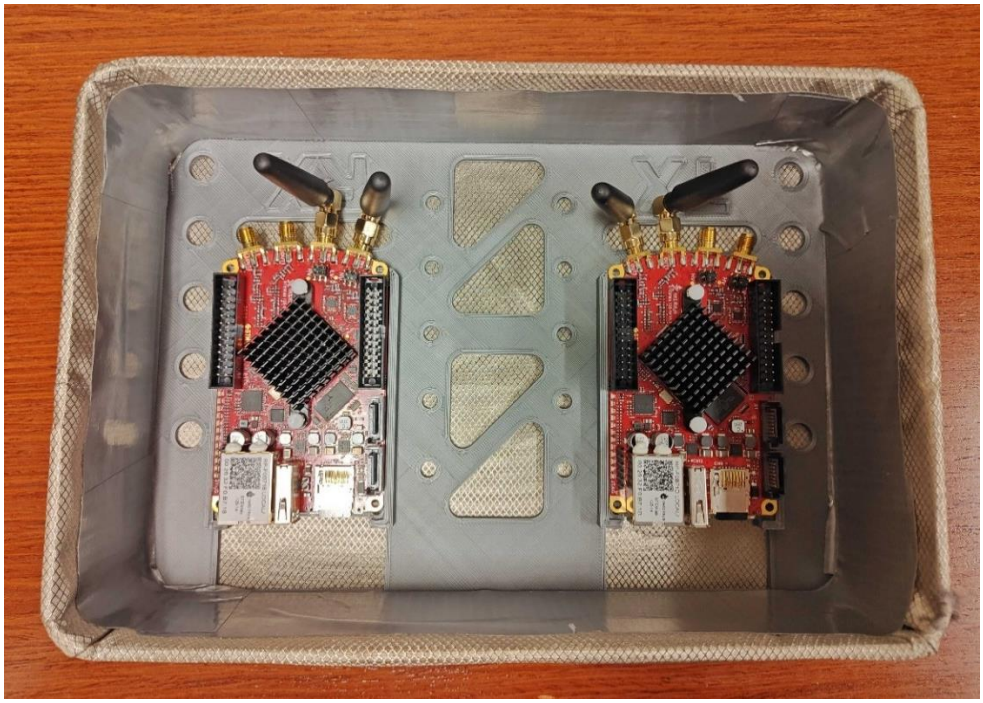


Figure 15. Faraday cage with 2 Red Pitayas installed. The cage is enclosed with Faraday fabric, except the top.

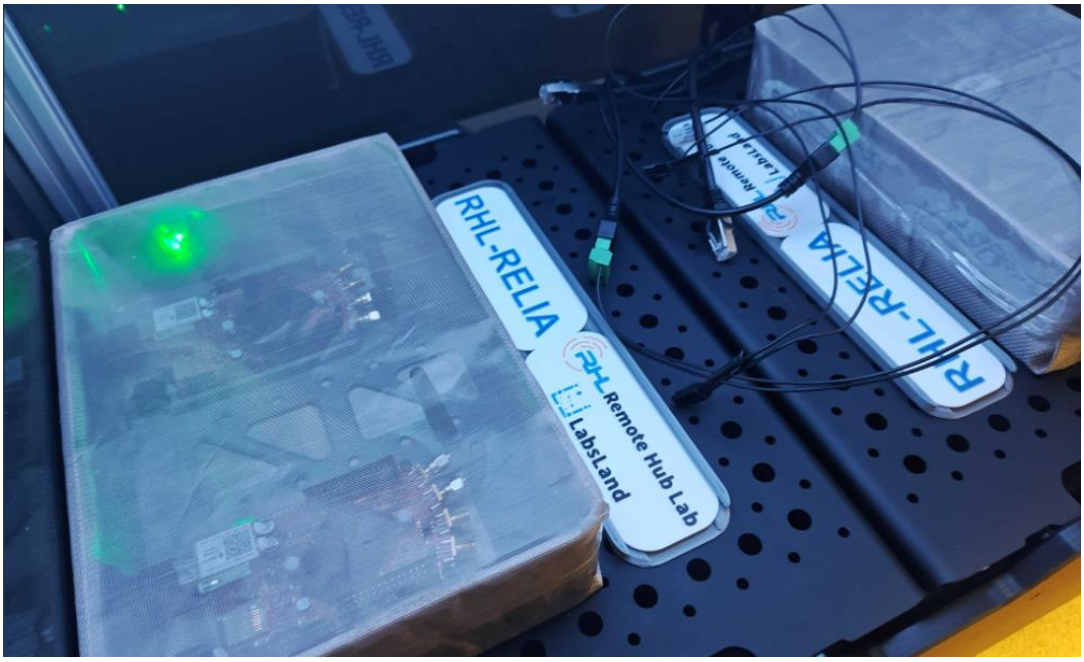


Figure 16. Faraday Cage Housing Red Pitayas within a LabsLand Prism4 structure.

3.4.2 Running GNU Radio Companion on Red Pitaya

In its present phase, RHL-RELIA allows users to upload GNU Radio Companion files (.grc) [63]. GNU Radio Companion (GRC) is a user-friendly graphical tool that enables users to easily configure a wide range of SDRs by assembling readily available or custom-created blocks into flowcharts. Our server then allocates an available SDR instance—a duo of SDRs within the same Faraday cage—and proceeds to program them. However, the factory image of Red Pitayas lacks built-in support for GNU Radio Companion, necessitating the use of a custom firmware to facilitate GNU Radio functionality on these devices. To evaluate this capability, several tests, including the transmission of a single-tone signal, were conducted. Figure 17 presents the flowchart of a GRC setup designed to emit a 100 kHz cosine wave, with a frequency plot of the transmitted signal depicted on the figure's right side.

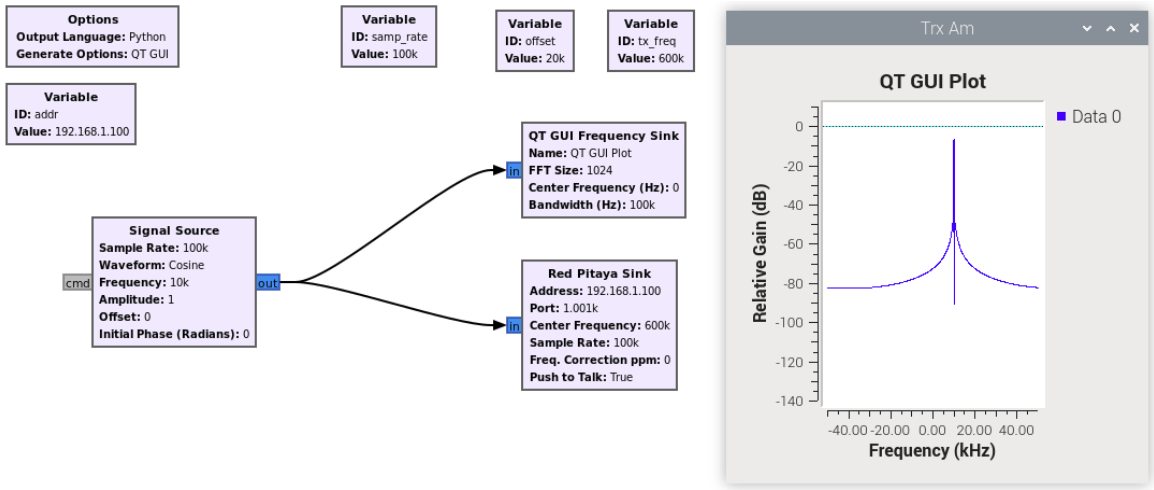


Figure 17. Transmitter program of a 10 kHz single tone cosine wave.

The GNU Radio receiver flowchart is displayed on the left side of Figure 18, while the corresponding received signal is illustrated on the right side of the same figure. The shapes of the transmitted and received signals are nearly identical, with a center peak at around 10 kHz and the received signal exhibiting a slightly lower gain, attributable to transmission loss.

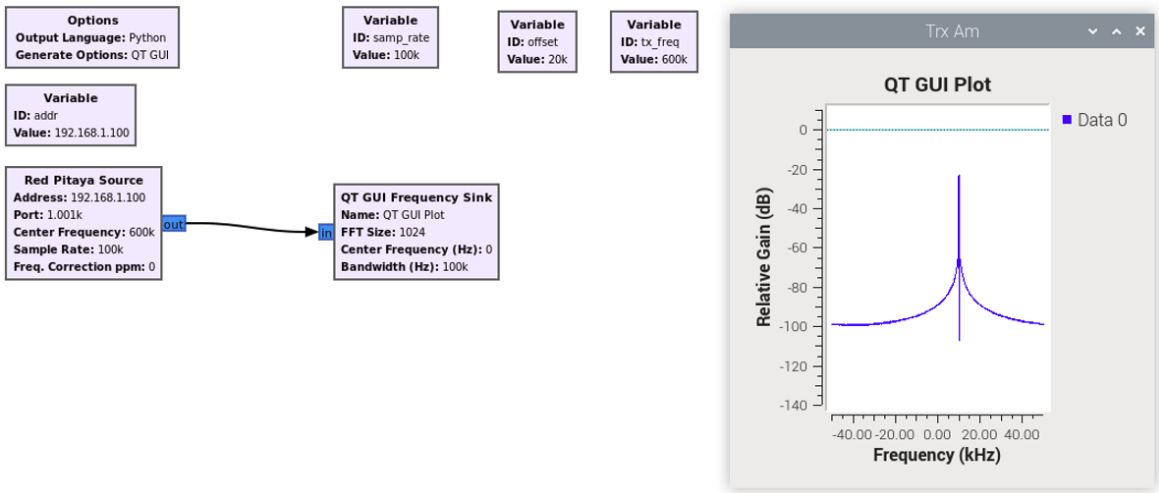


Figure 18. Receiver program of a 10 kHz single tone cosine wave.

3.5 Partial Reconfiguration Architectures

Having familiarized ourselves with the capabilities of Red Pitayas and successfully integrated them into our existing remote laboratory setup, we moved on to explore and devise PR

architectures suitable for Red Pitayas. These architectures could be applicable to other SDRs equipped with Zynq 7000 SoCs and Ethernet port. We developed two distinct architectures: one based on a User Interface (UI) and another leveraging command line interactions [64]. Each architecture presents its own set of advantages and disadvantages compared to the other.

3.5.1 User Interface (UI)-Based Architecture Overview

The user interface (UI)-based architecture aims to offer users an experience akin to that of the Vivado Design Suite. Within this framework, users will have the ability to upload both full and partial bitstreams of their FPGA programs to our web platform. Subsequently, our server will assign an available x86 computer to configure a pair of SDRs with Zynq 7000 SoCs. During their allocated session times, users can select specific partial bitstreams for programming the SDRs. This user interface is designed to mirror the Hardware Manager found in Vivado, as illustrated in Figure 19.

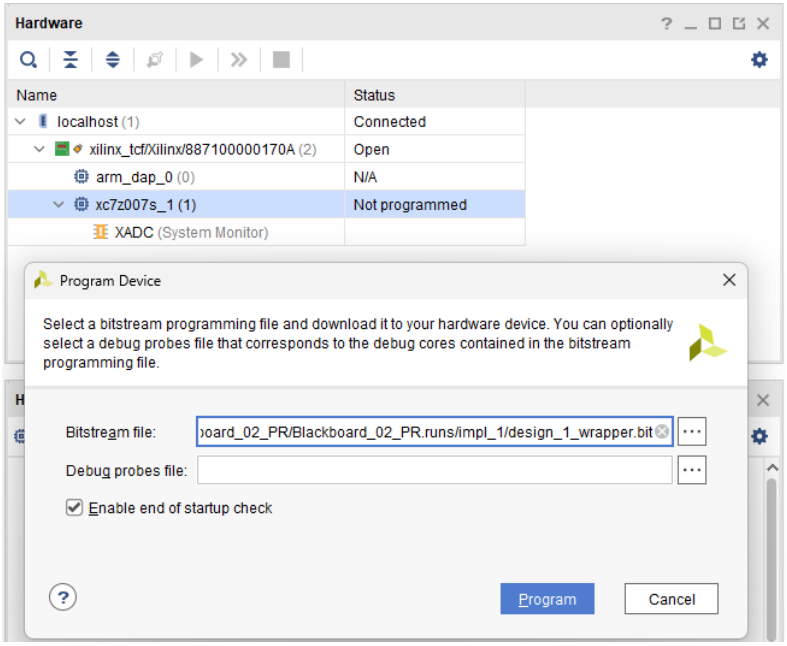


Figure 19. Hardware Manager within Vivado Design Suite. This interface allows users to choose a Zynq SoC and program it with bitstreams.

3.5.2 UI-Based Architecture Hardware Setup

Figure 20 illustrates the hardware components and their connections required for the UI-based architecture [64]. Beyond the server, which hosts a web interface and incorporates both the RELIA Data Exchanger and the RELIA Scheduler, this setup necessitates the use of Raspberry Pis, computers based on x86 architecture, JTAG programmers, and SDRs with Zynq 7000 SoCs. In this arrangement, Raspberry Pis serve to capture signals from the SDRs and relay them back to the server for display on the web interface. An x86 architecture-based computer is essential for running Vivado and programming FPGAs, as ARM-based devices, including Raspberry Pis and the latest Mac computers with Apple silicon, are not supported by Vivado as of this writing. JTAG programmers facilitate the connection between the Zynq SoCs on the SDRs and the x86 computer and assist the programming of the FPGAs. These programmers are optional for SDRs equipped with USB ports that allow for direct FPGA programming.

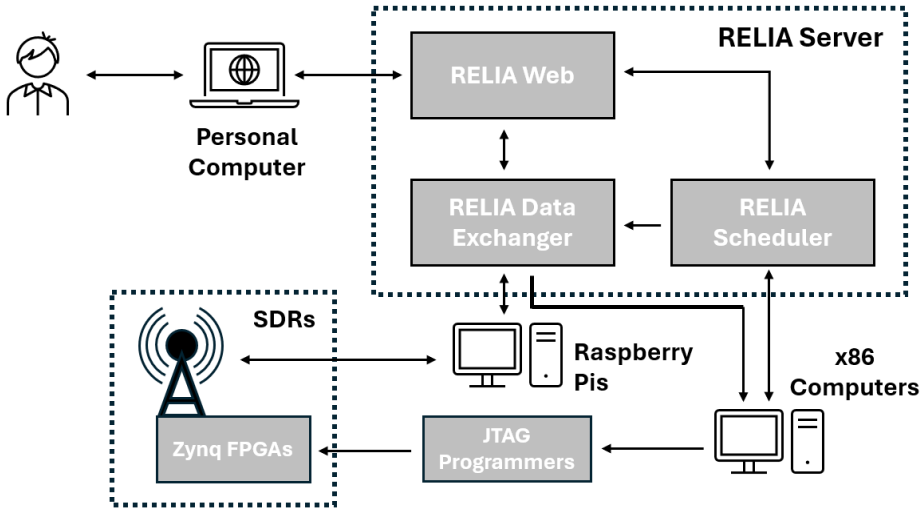


Figure 20. RHL-RELIA architecture for user interface (UI)-based FPGA PR.

For the JTAG programming interface, we opted for the JTAG-HS3 model. This choice was primarily influenced by its cost-effectiveness; at approximately $60 at the time of this

writing, it presents a significant saving over the manufacturer's own programmer, which is priced above $200 (the price is of June 2024). The primary limitation associated with the JTAG-HS3 is its unconventional 14-pin layout, whereas the Red Pitayas utilize a standard 6-pin JTAG configuration. Converters from 14 pins to 6 are notably expensive due to their rarity. To further reduce hardware costs, we employed a breadboard and Dupont cables to adapt the JTAG-HS3's 14-pin output to fit the 6-pin input on the Red Pitayas, as illustrated in Figure 21. The Red Pitaya is powered through a MicroUSB cable, while the JTAG-HS3 connects to a computer with an x86 architecture for the purpose of FPGA programming in Vivado. Ethernet cables link Raspberry Pis to Red Pitayas, allowing them to communicate via IP addresses, with data being exchanged through the SSH protocol. Note: the connection setup between Raspberry Pis and Red Pitayas is not depicted in the figure.

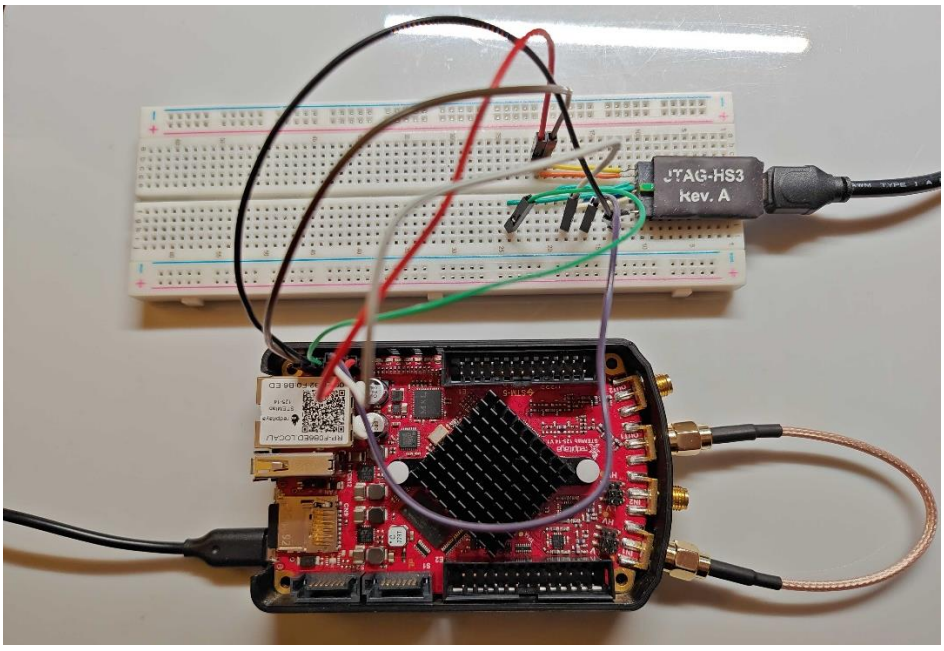


Figure 21. Red Pitaya and JTAG-HS3 programmer connection.

3.5.3 Command Line-Based Architecture Overview

The command line-based architecture enables users to program FPGAs through commands within a command console. Upon receiving both full and partial bitstreams, our server transfers these files to a Raspberry Pi. Users are then presented with a console window, where they can execute the command

```
cat bitstream.bit > /dev/xdevcfg
```

to program the FPGA. Prior to initiating PR, it's necessary to set the PR flag, signaling to the SoC that the forthcoming bitstream will be a partial one.

3.5.4 Command Line-Based Architecture Hardware Setup

The architecture based on command-line interaction necessitates a reduced number of hardware components compared to a setup centered around a user interface (UI). This simplification is depicted in Figure 22 [64]. By circumventing the need to incorporate Vivado for UI accessibility within our remote laboratory setup, we eliminate the requirement for both an x86 architecture-based computer and a separate JTAG programmer. Raspberry Pis serve a dual function in this streamlined approach, functioning as both the programmer and the receiver of data from SDRs. Raspberry Pis are interfaced with Red Pitayas via Ethernet cables, enabling communication through IP addresses. Data transfer is facilitated using the SSH protocol.

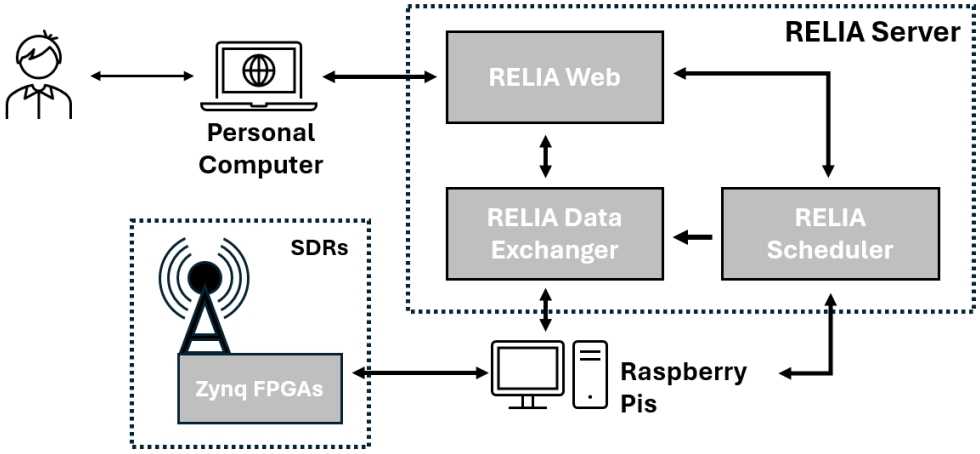


Figure 22. RHL-RELIA architecture for command line-based FPGA PR.

3.5.5 Comparison of UI-Based and Command Line-Based Architectures

The UI-based architecture and the command line-based architecture each offer distinct advantages and limitations. The UI-based approach, reminiscent of the Vivado Design Suite's Hardware Manager, provides an intuitive and visual interface, making it particularly accessible for users unfamiliar with command line operations. This ease of use, however, comes at the cost of requiring more complex hardware setups, including x86 architecture-based computers and JTAG programmers, to facilitate the connection and programming of FPGAs. On the other hand, the command line-based architecture offers a more streamlined and potentially faster workflow for those comfortable with console commands, significantly reducing the dependency on specific hardware by utilizing Raspberry Pis for direct programming. This method, while efficient for experienced users, presents a steeper learning curve and is less intuitive for beginners. Additionally, it lacks the visual feedback and ease of monitoring provided by a graphical interface, which is essential during debugging and development phases.

3.6 Evaluating Partial Reconfiguration on SDRs

Having developed the architectures, we proceeded to complete all FPGA tutorials available on the Red Pitaya website, acquiring a comprehensive understanding of the FPGA's interaction with peripherals such as ADCs, DACs, and GPIOs [65, 66]. This groundwork enabled us to begin creating partial reconfigurable programs for the Red Pitayas. Initial experiments were conducted on the Red Pitaya SDRlab 122-16, which features a more capable Z-7020 SoC, a broader frequency range, and enhanced ADCs and DACs. However, the scarcity of support and detailed guidance for crafting traditional SDR FPGA programs presented challenges in developing complex digital signal processing (DSP) programs. Consequently, we transitioned to using the STEMlab 125-14 model for subsequent PR evaluation tests. Several PR programs were executed on the Red Pitayas, with three particularly noteworthy examples detailed in the following sections.

3.6.1 Simple Partial Reconfigurable Calculator Program

The first partial reconfigurable program executed on the Red Pitayas was a basic calculator application, mirroring the structure of the program introduced in Section 3.3.1 utilizing the Blackboard. This initial endeavor served as a practical assessment of our grasp on the interconnection of GPIOs with the FPGA and the feasibility of implementing PR on Red Pitayas.

Figure 23 illustrates a block diagram of the program, which processes two 8-bit inputs from the onboard GPIO connector and outputs the result via the onboard LEDs. The Partial Reconfigurable Region (PRR) in pink was adaptable to program with an adder, subtractor, multiplier, or divider module. For the calculator program, a memory module is incorporated to store the values of the two operands, a and b, thereby minimizing the number of GPIO pins

utilized. In the absence of this module, 16 GPIO pins would be necessary to specify both 8-bit operands. However, with the memory module in place, only 10 GPIO pins are needed—eight for defining an 8-bit value and two for operand selection. Modules highlighted in lightly colored boxes are not essential for comprehending the program's behavior. These modules were either necessary for the program's operation or instantiated but not utilized in this specific instance. The first evaluation was conducted on the Red Pitaya SDRlab 122-16, employing both the UI-based and command line-based architectures for testing.

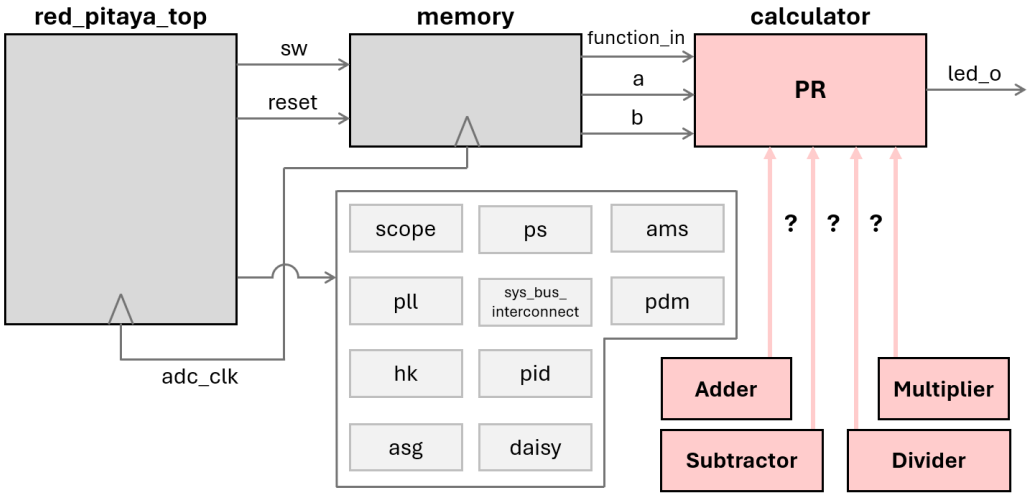


Figure 23. Block diagram of the partial reconfigurable calculator program on Red Pitaya.

Firstly, the UI-based architecture was deployed, enabling the Red Pitaya to be programmed with both full and partial bitstreams successfully. However, challenges emerged when attempting PR on the SDRlab 122-16 model using commands, as partial bitstreams led to program freezes. While full bitstreams configured the Red Pitaya without issues and resulted in the expected behavior, uploading partial bitstreams halted the program execution, and the LEDs failed to respond to changes in inputs. This issue could be rectified by reprogramming the FPGA with full bitstreams, but partial bitstreams remained ineffective. Upon consulting with Red Pitaya

engineers, they suggested outdated firmware might be the culprit, a theory that subsequent tests would prove inaccurate.

The problem continued even with the use of the most up-to-date firmware from the manufacturer, as well as custom firmware designed to support GNU Radio. Consequently, we opted to experiment with the STEMlab 125-14 model, which is equipped with a Z-7010 SoC. This model is more widely used due to its greater affordability and better support compared to the SDRlab 122-16. Switching to the STEMlab 125-14 resolved the issues encountered with command-line programming. Subsequent evaluations of PR on the Red Pitaya were exclusively conducted using the STEMlab 125-14 model.

3.6.2 1-PRR Partial Reconfigurable Scaling Program

The second notable partial reconfigurable program, similar to the calculator program, features one Partial Reconfigurable Region (PRR) but focuses on a signal processing application more relevant to SDR functionalities. This program incorporates a PRR that can be programmed with scaling modules, offering the option to cap signal peaks. Without peak capping, signal overflow might occur when multiplied by a substantial scaling factor, causing any signal portion exceeding the threshold to be "clipped" towards the horizontal axis. Conversely, with peak capping, those overflowed portions are maintained at the limit values.

Figure 24 presents a simplified block diagram of the program. The central element, the process module, represents the PRR and can be programmed with one of the scaling PR modules. The analog module serves as an intermediary between the process module and the DACs/ADCs, converting ADC signals from unsigned format to two's complement. Meanwhile, the asg (arbitrary signal generator) module draws data from a buffer, supplying it to the DACs and

generating initial data for the process module prior to full system configuration. As with the calculator programs, some modules are highlighted in lighter colors without further explanation.

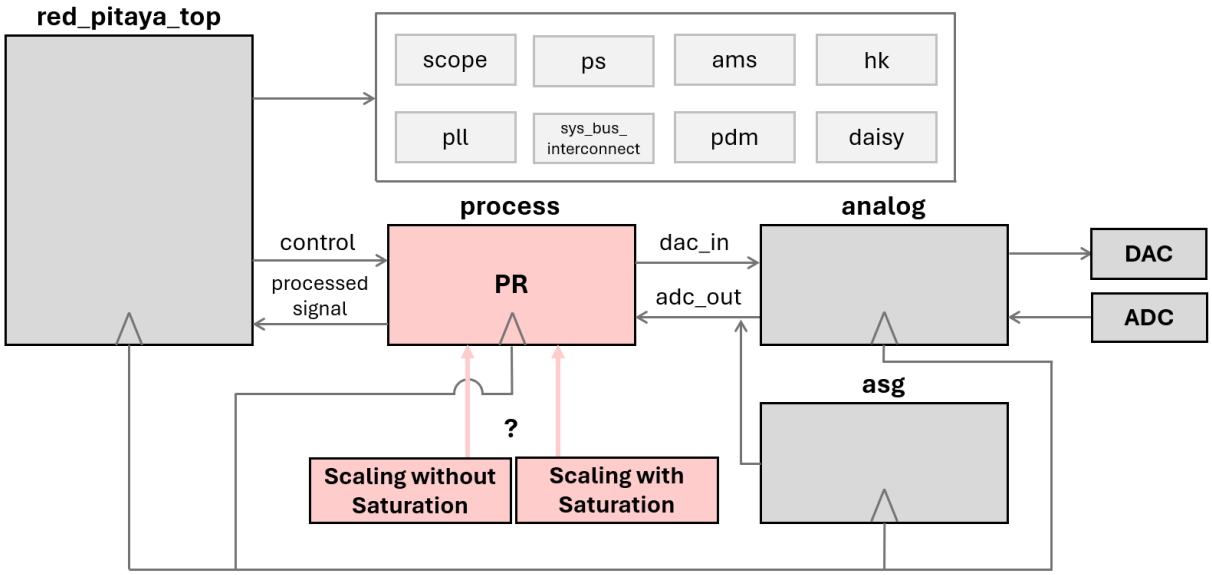


Figure 24. Block diagram of the partial reconfigurable 1-PRR scaling program on Red Pitaya.

3.6.3 2-PRR Partial Reconfigurable Filtering and Scaling Program

Building on the concepts introduced in the previous program, the third program employs a PR setup incorporating two PRRs. The first PRR mirrors that of the earlier program, offering the flexibility to be programmed with a scaling module, either with or without saturation functionality. The second PRR introduces a new filter block, providing options for programming with either a 5-tap low-pass or high-pass filter module with a predefined cutoff frequency. A simplified block diagram of this 2-PRR system is shown in Figure 25.

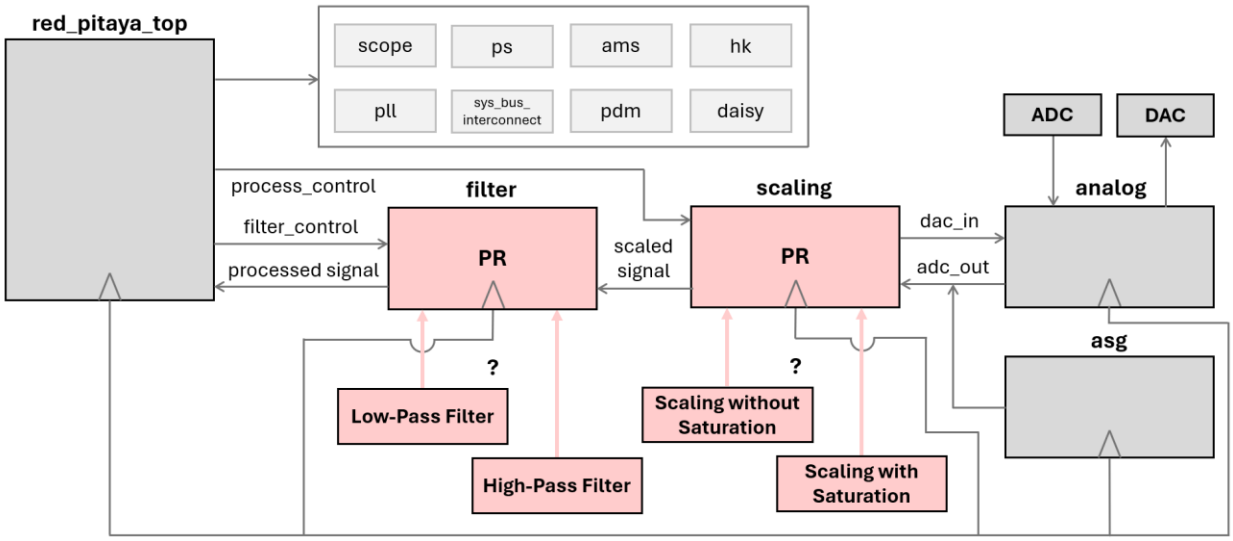


Figure 25. Block diagram of the partial reconfigurable 2-PRR filtering and scaling program on Red Pitaya.

Implementing PR programs with two PRRs offers enhanced flexibility and functionality by allowing for a wider range of PR modules to be used, such as amplification and filtering for radios, and optimized resource usage through task-specific PRR optimization. However, this approach introduces increased complexity in design, requiring more sophisticated control logic for managing multiple PRRs and meticulous floorplanning to balance resource allocation. Figure 26 depicts the outcome of the Floorplanning for this program, with its two PRRs highlighted in magenta boxes towards the bottom of the FPGA layout. Additionally, the debugging and validation processes become more challenging due to the potential for issues arising from the interactions between PRRs and the complex control structures needed. Despite these challenges, the use of two PRRs can significantly augment a system's capabilities, making it a worthwhile consideration for advanced FPGA applications.

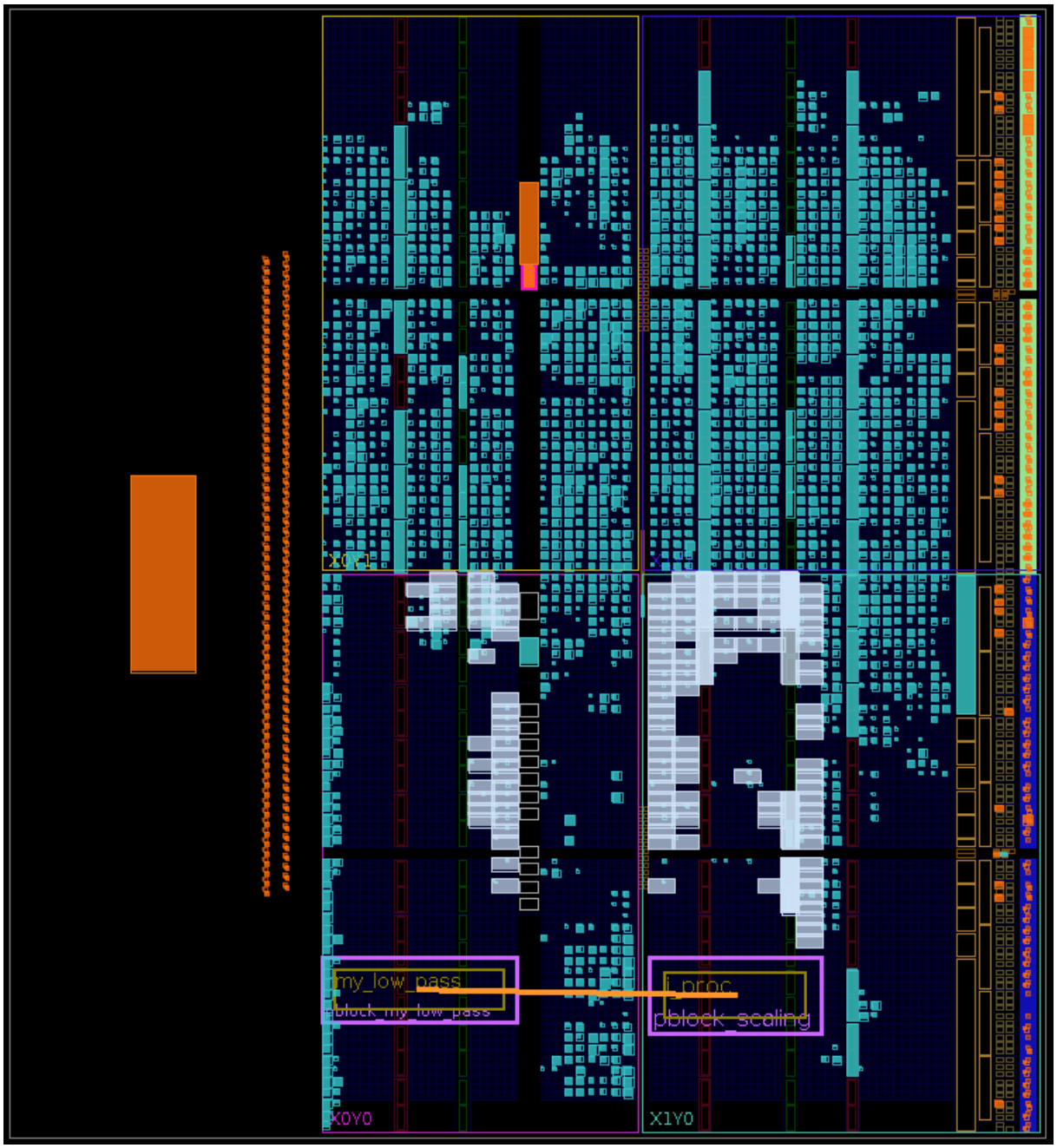


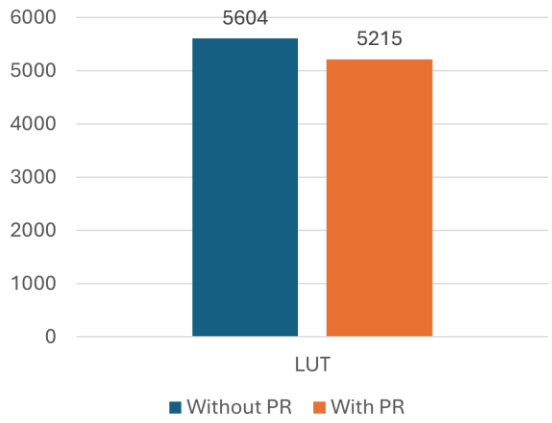
Figure 26. Floorplanning outcome of the 2-PRR filtering and scaling program, highlighting two PRRs in magenta, positioned near the bottom of the FPGA layout.

Chapter 4. RESULTS

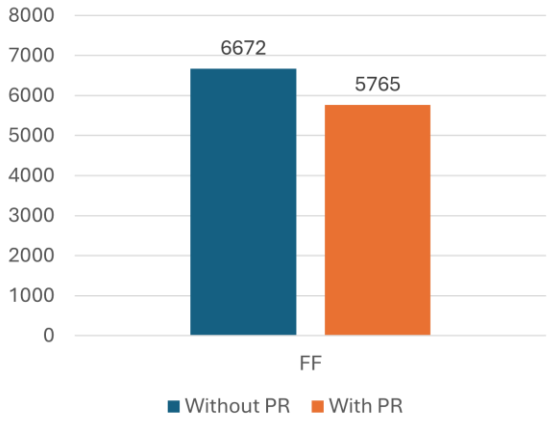
Following the successful synthesis, implementation, and bitstream generation of the three aforementioned Red Pitaya programs, data concerning hardware resource usage and total power were compiled and are summarized in this section.

4.1 Simple Partial Reconfigurable Calculator Program Result

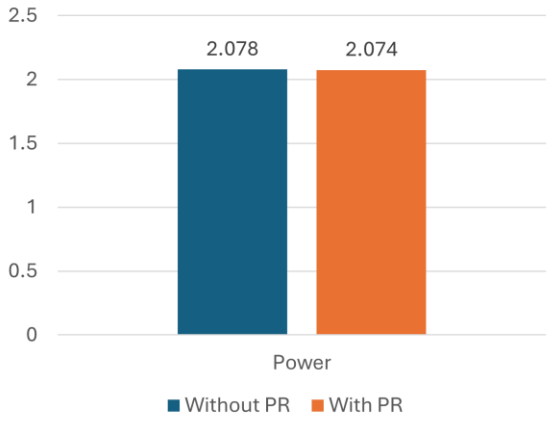
Figure 27 displays the hardware resource utilization for the 8-bit calculator program, comparing versions with and without PR enabled. A decrease of 6.9% in LUT usage and 13.6% in FF usage were noted. However, the reduction in total power consumption was minimal, amounting to less than 0.2%. There was no observed decrease in the usage of BRAM and DSP; both PR and non-PR programs utilized 30.0 BRAM and 24 DSPs.



(a)



(b)



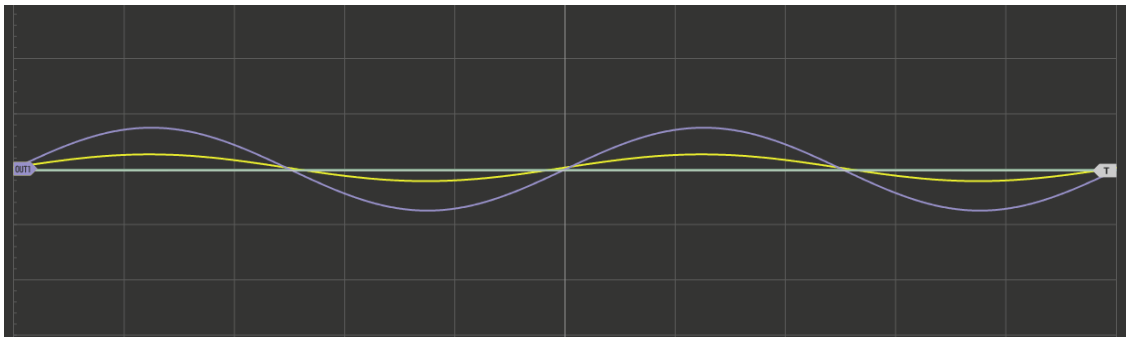
(c)

Figure 27. LUT and FF utilization and power consumption of the 8-bit calculator program on Red Pitaya. (a) LUT utilization. (b) FF utilization. (c) Total power.

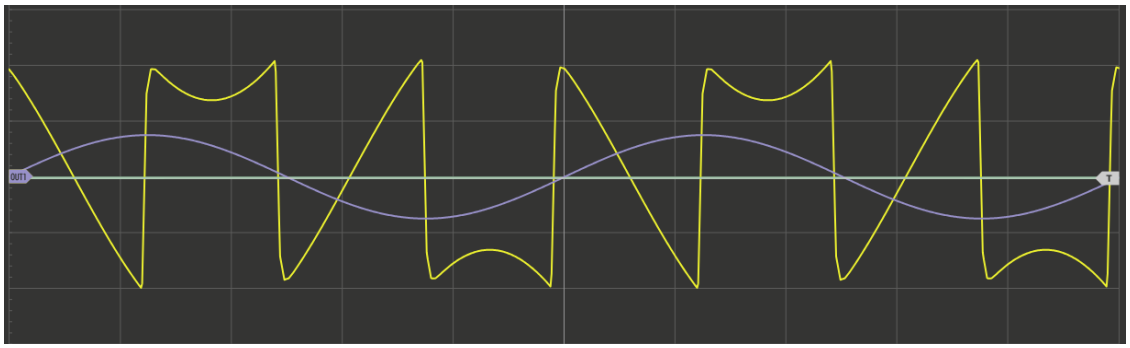
4.2 1-PRR Partial Reconfigurable Scaling Program Result

Figure 25 displays the resulting waveforms of this scaling program. Part a illustrates the original transmitted signal from one of the Red Pitaya's output channels. The purple wave represents the transmitted signal, and the yellow wave represents the received signal after being digitally processed. When the scaling factor is adjusted to a level that induces overflow, one PR module does not have any saturation values, resulting in the sine wave's peak being “clipped” to the x-axis, as depicted in part b. Conversely, the other PR module introduces thresholds—both

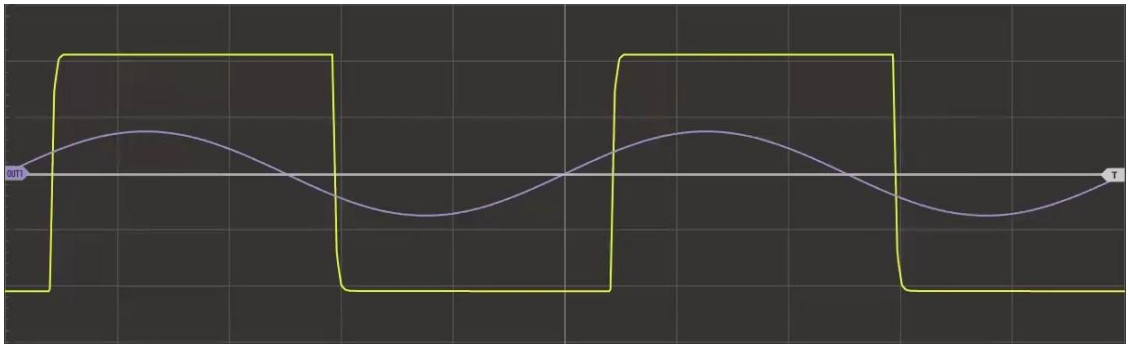
positive and negative—that cap the signal once it's multiplied by the scaling factor. This mechanism ensures that any overflowed signals are capped at these thresholds, as demonstrated in part c. Notably, a temporal discrepancy between the input and output signals is observable in parts b and c. The exact reason for this delay remains uncertain; however, it is suspected to be attributed to either network delays, stemming from the Red Pitaya's connection to the monitoring device via a router, or an internal timing shift caused by the oscillators within the Red Pitayas.



(a)



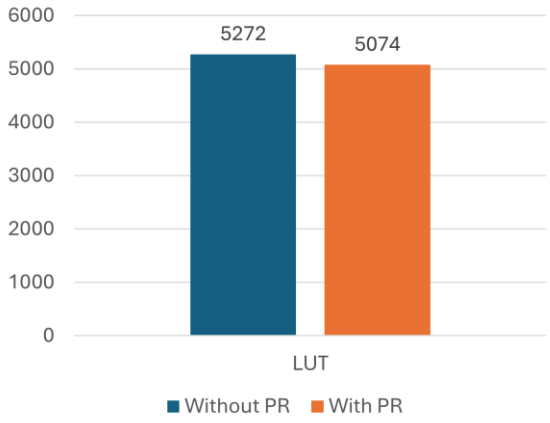
(b)



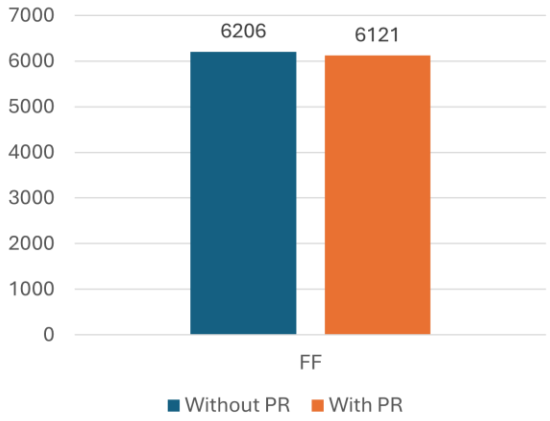
(c)

Figure 28. Waveform analysis of scaled signals in a partial reconfigurable environment. (a) The original transmitted signal is purple, and the received signal is yellow. (b) Waveform after scaling without saturation, showing peak clipping to the x-axis. (c) Waveform with scaling and threshold saturation, illustrating peak capping at predefined thresholds.

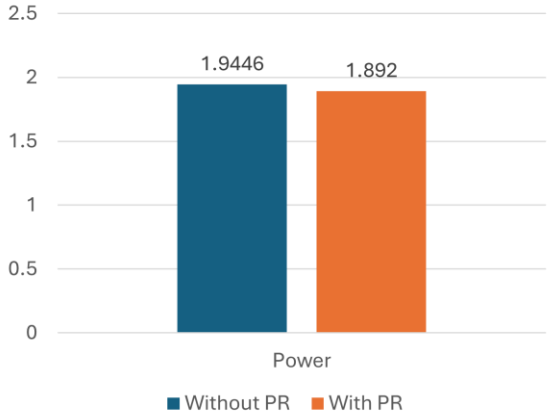
Figure 29 displays a comparison of LUT, FF, DSP usage, and total power consumption between versions of the same program, with and without PR enabled. The observed reductions in LUT and FF usage are relatively minor, at approximately 3.8% and 1.4% respectively. Such minimal reductions could likely be attributed to a suboptimal selection of PRR during the Floorplanning process, suggesting potential for further optimization to minimize resource consumption. Since the presence of more PR modules for a given PRR tends to correlate with greater resource savings, expanding the variety of PR modules could reduce resource use further. Our limited knowledge in floorplanning limited the extent of optimization achievable. Total power consumption was reduced by about 2.7%. The decrease in DSP slice utilization was particularly significant, showing a reduction of over 14.3%, with the total number of DSP slices needed decreasing from 14 to 12. BRAM utilization remains consistent at 30.0 for both program variants.


6000
5000
4000
3000
2000
1000
0
5272
5074
LUT
Without PR
With PR

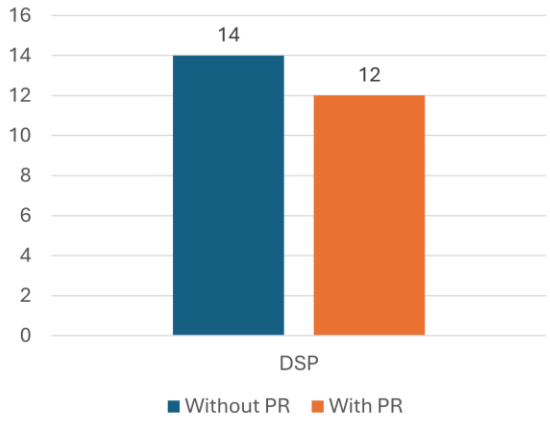
(a)


7000
6000
5000
4000
3000
2000
1000
0
6206
6121
FF
Without PR
With PR

(b)


2.5
2
1.5
1
0.5
0
1.9446
1.892
Power
Without PR
With PR

(c)

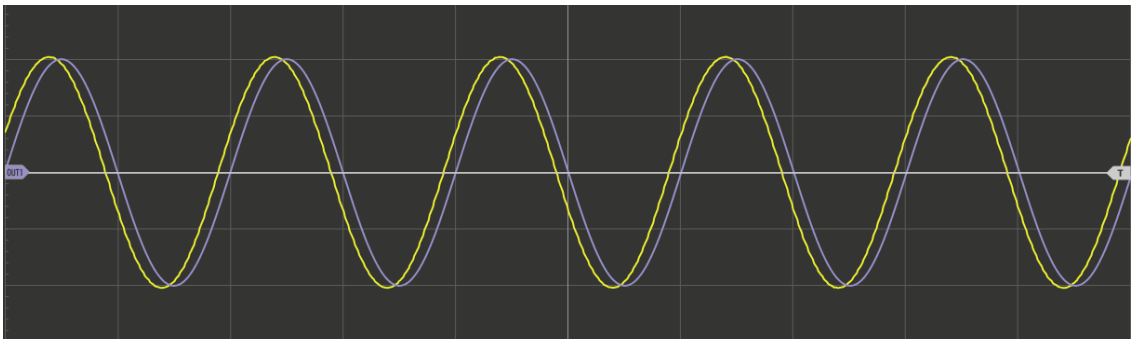


(d)

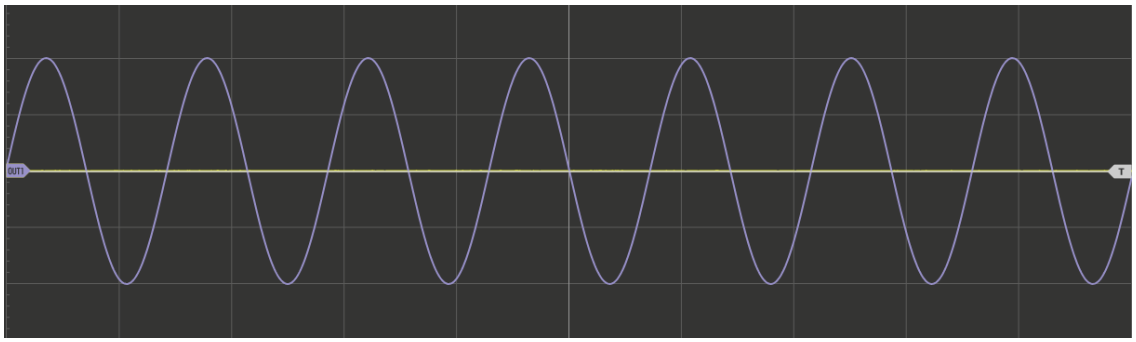
Figure 29. LUT, FF, and DSP slices utilization and power consumption of the 1-PRR scaling program on Red Pitaya. (a) LUT utilization. (b) FF utilization. (c) Total power. (d) DSP slice utilization.

4.3 2-PRR Partial Reconfigurable Filtering and Scaling Program Result

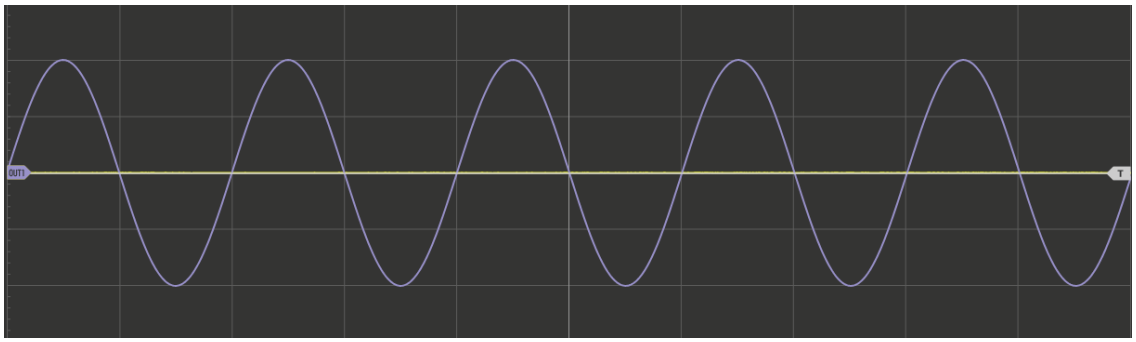
Prior to integrating both PRRs into a single program, the newly developed filter module was first tested on the Red Pitaya. The outcomes of this test are illustrated in Figure 30. With the cutoff frequency established at 600 kHz, programming the PRR with the low-pass filter retains signals at 500 kHz, but largely attenuates signals at 700 kHz. Conversely, employing the high-pass filter module results in the opposite effect: signals at 500 kHz are significantly reduced, whereas those at 700 kHz are maintained.



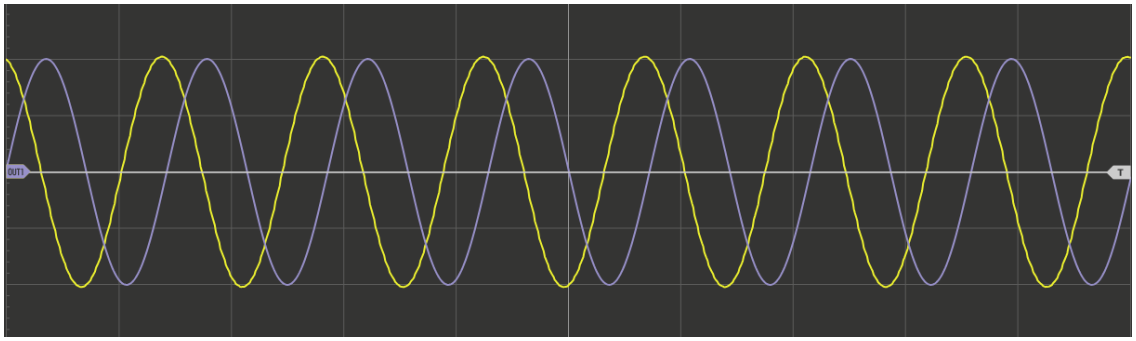
(a)



(b)



(c)



(d)

Figure 30. Waveform outcomes through the filter PRR at a 600 kHz cutoff frequency. (a) 500 kHz signal filtered by the low-pass filter. (b) 700 kHz signal filtered by the low-pass filter. (c) 500 kHz signal filtered by the high-pass filter. (d) 700 kHz signal filtered by the high-pass filter.

Although each PRR functioned effectively on its own, combining them into a single program led to observed glitches and instability. For instance, Figure 31 illustrates the waveform of a 300 kHz signal, scaled by a factor of 1.78 with a saturation value, and subsequently processed by a low-pass filter with a 600 kHz cutoff frequency. In this scenario, sections of the signal that should have been negative were instead converted to positive values. Glitches became

particularly noticeable when the frequency of the transmitted signal was significantly higher than the filter's cutoff frequency. Figure 32 displays the outcome for a 3 MHz signal passing through a high-pass filter with a 600 kHz cutoff frequency and a scaler set to a factor of 2.4.

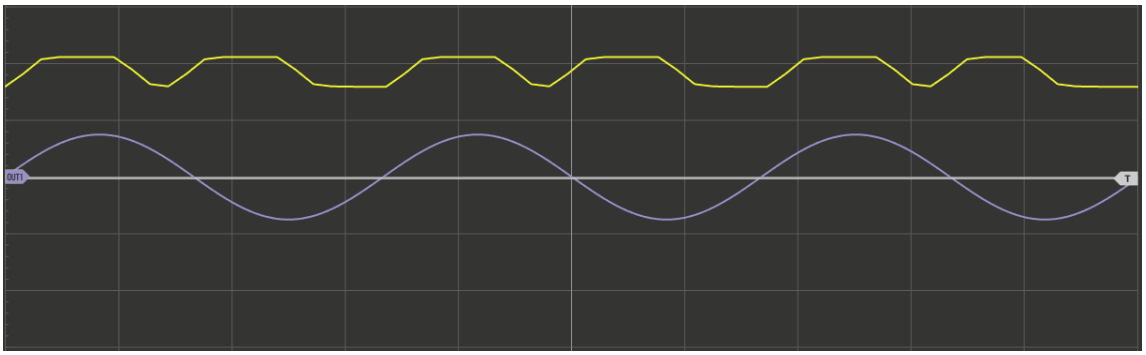


Figure 31. Waveform of a 300 kHz signal scaled by 1.78 with saturation, post low-pass filtering at a 600 kHz cutoff frequency.

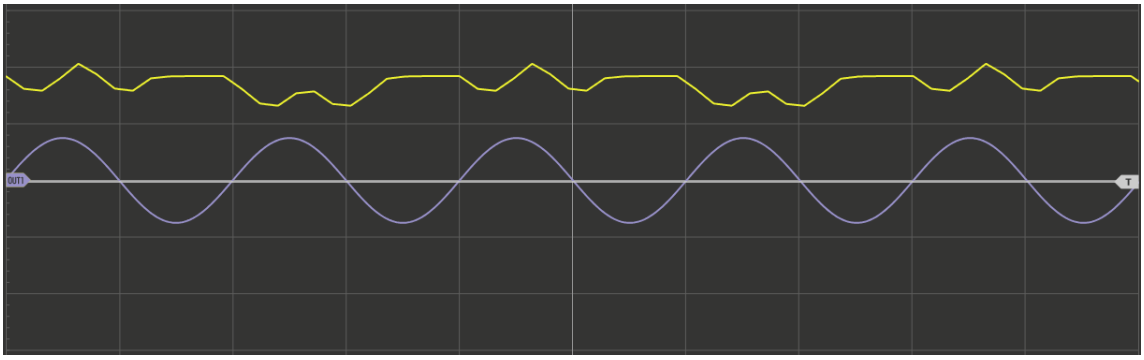


Figure 32. Resulting signal of a 3 MHz frequency passing through a high-pass filter with a 600 kHz cutoff frequency and scaled by a factor of 2.4.

The anomalies observed in the processed signals can likely be attributed to suboptimal timing constraints, as indicated by several critical warnings that emerged post successful synthesis, illustrated in Figure 33. Establishing appropriate timing constraints necessitates a solid understanding of FPGA architecture and programming—a proficiency we currently lack but plan to develop further in future studies.

Design Timing Summary

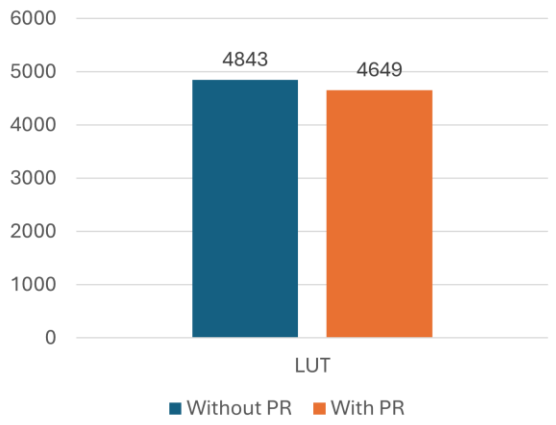
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-3.625 ns	Worst Hold Slack (WHS):	0.031 ns	Worst Pulse Width Slack (WPWS):	1.000 ns
Total Negative Slack (TNS):	-1133.661 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	2110	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	18528	Total Number of Endpoints:	18528	Total Number of Endpoints:	6976

Timing constraints are not met.

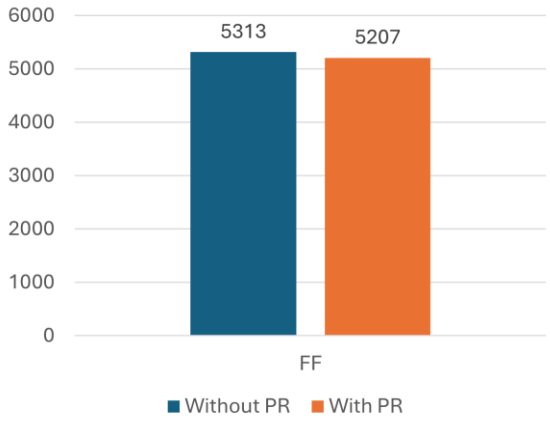
Figure 33. Timing summary in Vivado highlighting that the design did not fully satisfy the timing constraints.

Figure 34 provides a comparison of hardware utilization for the program, highlighting the resource reduction through enabling PR. By implementing PR, the system achieved a reduction of 4.0% in LUTs, with numbers falling from 4843 to 4649. Similarly, FF usage decreased by 2.0%, dropping from 5313 to 5207. The utilization of DSP slices saw a notable decrease of 14.3%, moving from 14 to 12 slices. Despite these reductions, both BRAM utilization and overall power consumption stayed consistent, with BRAM usage holding steady at 30.0 and total power consumption unchanged at 2.105 W.

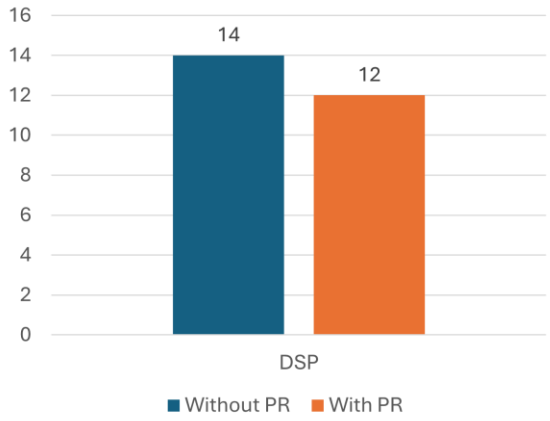
The reductions in hardware utilization have the potential to be even more substantial with optimized floorplanning, specifically by strategically placing and resizing PRRs. This approach offers a promising avenue for future exploration, aiming to determine the extent to which hardware resources can be minimized through PR in programs that incorporate multiple PRRs.



(a)



(b)



(c)

Figure 34. LUT, FF, and DSP slice utilization of the 2-PRR filtering and scaling program on Red Pitaya. (a) LUT utilization. (b) FF utilization. (c) DSP slice utilization.

Chapter 5. Discussion

5.1 Advantages of PR

The implementation of PR in FPGA-based systems, particularly within the realm of SDRs, introduces several substantial advantages that stand to revolutionize how these systems are designed and utilized. One of the most significant benefits of PR is the ability to dynamically modify the functionality of an FPGA without needing to halt its operations or reprogram it entirely. This capability not only enhances the flexibility and adaptability of FPGA applications but also significantly reduces the reconfiguration time, making it possible to switch between different functionalities or algorithms on-the-fly. Such efficiency is invaluable in signal processing applications requiring rapid response to changing conditions or in scenarios where downtime is critical, such as during satellite switching in communication systems. Additionally, this ability to update or add new functionalities through PR without disrupting the existing system offers a path towards future-proofing hardware, allowing for iterative development and continuous improvement.

Furthermore, PR enables more efficient use of the FPGA's resources by allowing for the allocation of hardware resources to different tasks as needed, thereby optimizing performance and potentially reducing power consumption, as the five FPGA example programs illustrate earlier in this thesis.

Beyond these technical benefits, PR also contributes to the practicality and scalability of FPGA projects. By facilitating the development of more complex, multi-functional systems within a single FPGA chip, PR reduces the need for multiple dedicated chips, thereby saving on hardware costs and physical space. This aspect is particularly advantageous in educational settings, such as remote laboratories, where PR can enable a wider range of experiments and

learning opportunities without the need for expansive physical infrastructure. The exploration of PR, especially in multi-PRR setups, underscores its potential to push the boundaries of what is possible with FPGA technology, highlighting its role as a key driver in the evolution of digital design and application.

5.2 Disadvantages of PR

Despite its numerous benefits, PR in FPGA also presents challenges and disadvantages that cannot be ignored. One of the primary hurdles is the increased complexity in the design and management of FPGA PR modules. Implementing PR modules requires a deeper understanding of the FPGA's architecture and the development of sophisticated control logic, such as multiplexers and decouplers, to manage the reconfiguration process, which can steepen the learning curve for designers. Additionally, effective utilization of PR demands meticulous planning and optimization in the floorplanning stage, complicating the design process further. If floorplanning is not carefully executed, activating PR might inadvertently increase hardware resource consumption. For instance, in our initial attempt with a suboptimal PRR selection for the calculator program on the Red Pitaya, there was an approximate increase of 3.1% in LUT and 2.3% in FF usage after activation of PR. Through subsequent iterations aimed at identifying a more suitable PRR location and size, we were eventually able to locate a better PRR that resulted in resource savings.

This complexity extends to debugging and validation efforts, as isolating issues within a dynamically reconfigurable system is more challenging than in a static FPGA design. Moreover, the need for specific tools and support from FPGA vendors for PR can limit accessibility and increase dependency on proprietary solutions. Despite these drawbacks, the advantages of PR often outweigh the challenges, particularly for applications where its flexibility and efficiency

can be fully leveraged. However, the considerations of complexity and resource requirements highlight the importance of careful planning and expertise in the deployment of PR strategies.

Chapter 6. CONCLUSION

This thesis explores the integration of Partial Reconfiguration (PR) within the realm of Software-Defined Radios (SDRs), particularly through the lens of remote laboratory environments. Through the development and evaluation of various PR programs on a development board and the Red Pitaya, a popular SDR platform, this work has demonstrated the benefits that PR offers, including enhanced system flexibility, optimized resource utilization, and decreased total power consumption. This approach provides a path toward developing more complex and resource-intensive applications without necessitating more powerful hardware.

By introducing and testing both UI-based and command line-based architectures, each requiring different hardware and software implementations for PR, this research lays the groundwork for more sophisticated, multi-functional FPGA applications that can be integrated into SDR remote laboratories.

Due to the advanced nature of PR and its rare application on SDRs, there are limited tutorials and resources available to assist when issues arise. Despite these limitations, we successfully developed and evaluated 16 partial reconfigurable programs. While there is no standard on the number of programs required for an extensive evaluation in this area, we believe this number is sufficient for a thorough assessment. The intricacies involved in PR, coupled with the sparse resources available on this advanced subject, underscore the necessity for a deeper understanding of both PR and FPGA architecture. Such knowledge is vital for mastering PR applications and optimizing FPGA resource utilization effectively. The challenges inherent in PR development highlight the need for more extensive exploration and documentation in this field to facilitate better design strategies and to leverage the full potential of FPGA technologies.

In conclusion, this thesis not only contributes to the academic and practical understanding of PR in SDRs but also highlights the importance of continued research and development in this area. By paving the way for future advancements, it underscores the potential for PR to drive significant improvements in the flexibility and efficiency of FPGA-based systems in remote laboratory environments.

Chapter 7. FUTURE WORK

Future directions for this project include the development of additional Partial Reconfiguration (PR) programs, both for further evaluation and to enhance student engagement with PR through our remote laboratory. Additionally, we aim to assess the viability of PR on other popular platforms and to integrate PR functionalities into our remote laboratory.

Two specific areas of PR programming merit further exploration. Firstly, there is a need for deeper investigation into programs that utilize multiple Partial Reconfiguration Regions (PRRs), focusing on optimizing timing and routing. Establishing a systematic approach for implementing such programs is crucial to streamline future explorations. Secondly, programs that leverage nearly all hardware resources available on SDRs present an intriguing opportunity. Without PR, such programs would deplete the hardware capabilities of an SDR, yet with PR, they become feasible. This extreme utilization scenario will provide valuable insights into the potential cost savings PR offers by enabling the use of less powerful devices more efficiently. However, the creation of such complex programs demands significant time and dedication, often spanning several years.

Most evaluations and tests detailed in this study were conducted using Red Pitayas. It's worth noting that many other cost-effective SDRs on the market are also powered by Zynq 7000 Series SoCs, which suggests a similarity in the underlying logic required for implementing PR across these devices. Given this commonality, it stands to reason that these SDRs could potentially benefit from PR in much the same way as the Red Pitayas have. Therefore, extending PR experimentation to include these other devices could prove invaluable. Such explorations could not only validate the applicability and advantages of PR across a broader range of

hardware but also pave the way for a more widespread adoption of PR techniques in the development and optimization of SDRs.

To encourage greater student engagement with this advanced technology, it's essential to develop an extensive library of PR modules and create a user-friendly interface. A feasible approach could involve the development of a graphical application that enables users to connect various PR module blocks into complete designs. Subsequently, these designs would undergo synthesis and implementation on a server, with the resulting bitstreams used to configure FPGAs within remote SDRs. This initiative aims to not only foster curiosity among students about the possibilities offered by PR in FPGA programming but also to inspire engineers to explore and incorporate this distinctive feature in their upcoming projects.

In summary, further research, the development of more PRRs, and the integration of PR into diverse platforms hold the promise of unlocking even greater efficiencies and innovations in FPGA-based systems. Establishing a comprehensive library of PR modules and creating a user-friendly graphical application for module assembly could significantly lower the barriers to entry for students and engineers alike, fostering wider exploration and adoption of PR.

REFERENCES

[1] M. N. O. Sadiku and C. M. Akujuobi, "Software-defined radio: a brief overview," *IEEE Potentials,* vol. 23, no. 4, pp. 14-15, 2004, doi: 10.1109/MP.2004.1343223.

[2] A. L. G. Reis, A. F. Barros, K. G. Lenzi, L. G. P. Meloni, and S. E. Barbin, "Introduction to the Software-defined Radio Approach," *IEEE Latin America Transactions,* vol. 10, no. 1, pp. 1156-1161, 2012, doi: 10.1109/TLA.2012.6142453.

[3] A. Pini. "Learn the Fundamentals of Software-Defined Radio." (accessed 03/28, 2024).

[4] "Main Page." https://wiki.gnuradio.org/index.php?title=Main_Page (accessed March 29, 2024).

[5] D. Kafetzis, S. Vassilaras, G. Vardoulias, and I. Koutsopoulos, "Software-Defined Networking Meets Software-Defined Radio in Mobile ad hoc Networks: State of the Art and Future Directions," *IEEE Access,* vol. 10, pp. 9989-10014, 2022, doi: 10.1109/ACCESS.2022.3144072.

[6] G. Kakkavas, K. Tsitseklis, V. Karyotis, and S. Papavassiliou, "A Software Defined Radio Cross-Layer Resource Allocation Approach for Cognitive Radio Networks: From Theory to Practice," *IEEE Transactions on Cognitive Communications and Networking,* vol. 6, no. 2, pp. 740-755, 2020, doi: 10.1109/TCCN.2019.2963869.

[7] "Product Advantages." Xilinx. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html#productAdvantages (accessed April 2, 2024).

[8] "XA Zynq-7000 SoC Data Sheet: Overview," 1.3.2 ed. AMD: Xilinx, 2018.

[9] "Product Table." Xilinx. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html#productTable (accessed April 02, 2024).

[10] "Partial Reconfiguration in the ISE Design Suite." Xilinx. https://www.xilinx.com/products/design-tools/partial-reconfiguration.html (accessed March 27, 20244).

[11] A. R. Bucknall and S. A. Fahmy, "Runtime Abstraction for Autonomous Adaptive Systems on Reconfigurable Hardware," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1-5 Feb. 2021 2021, pp. 1616-1621, doi: 10.23919/DATE51398.2021.9474199.

[12] A. R. Bucknall, S. Shreejith, and S. A. Fahmy, "Network Enabled Partial Reconfiguration for Distributed FPGA Edge Acceleration," in *2019 International Conference on Field-Programmable Technology (ICFPT)*, 9-13 Dec. 2019 2019, pp. 259-262, doi: 10.1109/ICFPT47387.2019.00042.

[13] A. R. Bucknall, S. Shreejith, and S. A. Fahmy, "Build Automation and Runtime Abstraction for Partial Reconfiguration on Xilinx Zynq UltraScale+," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, 9-11 Dec. 2020 2020, pp. 215-220, doi: 10.1109/ICFPT51103.2020.00037.

[14] M. Nguyen, R. Tamburo, S. Narasimhan, and J. C. Hoe, "Quantifying the Benefits of Dynamic Partial Reconfiguration for Embedded Vision Applications," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 8-12 Sept. 2019 2019, pp. 129-135, doi: 10.1109/FPL.2019.00029.

[15] M. Taher and A. Khan, *Impact of Simulation-based and Hands-on Teaching Methodologies on Students' Learning in an Engineering Technology Program*. 2014, pp. 24.701.1-24.701.22.

[16] R. Morales-Menendez, R. A. Ramírez-Mendoza, and A. J. V. Guevara, "Virtual/Remote Labs for Automation Teaching: a Cost Effective Approach∗∗Authors thank Tecnológico

de Monterrey because its support," *IFAC-PapersOnLine*, vol. 52, no. 9, pp. 266-271, 2019/01/01/ 2019, doi: https://doi.org/10.1016/j.ifacol.2019.08.219.

[17] "Developing a remote laboratory for engineering education," *Computers & Education*, pp. 1686-1697, 2011, doi: https://doi.org/10.1016/j.compedu.2011.02.015.

[18] J. Solis-Lastra and B. Albertini, "A Light Systematic Literature Review on Remote Laboratories for Engineering," in *2021 IEEE Sciences and Humanities International Research Conference (SHIRCON)*, 17-19 Nov. 2021 2021, pp. 1-4, doi: 10.1109/SHIRCON53068.2021.9652258.

[19] R. Hussein, R. Maloney, L. Rodrguez-Gil, J. A. Beroz, and P. Orduna, "RHL-BEADLE: Bringing Equitable Access to Digital Logic Design in Engineering Education," *2023 ASEE Annual Conference \& Exposition Proceedings*.

[20] E. Fabregas, G. Farias, S. Dormido-Canto, S. Dormido, and F. Esquembre, "Developing a remote laboratory for engineering education," *Computers & Education*, vol. 57, no. 2, pp. 1686-1697, 2011/09/01/ 2011, doi: https://doi.org/10.1016/j.compedu.2011.02.015.

[21] R. Hussein, M. Guo, P. Amarante, L. Rodriguez-Gil, and P. Orduna, "Digital Twinning and Remote Engineering for Immersive Embedded Systems Education," in *2023 IEEE Frontiers in Education Conference (FIE)*, 18-21 October 2023 2023, pp. 1-8, doi: 10.1109/FIE58773.2023.10343436. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/FIE58773.2023.10343436

[22] R. Hussein *et al.*, "Remote Hub Lab – RHL: Broadly Accessible Technologies for Education and Telehealth," in *Open Science in Engineering*, Cham, M. E. Auer, R. Langmann, and T. Tsiatsos, Eds., 2023// 2023: Springer Nature Switzerland, pp. 73-85.

[23] J. E. Corter, S. K. Esche, C. Chassapis, J. Ma, and J. V. Nickerson, "Process and learning outcomes from remotely-operated, simulated, and hands-on student laboratories," *Computers & Education*, vol. 57, no. 3, pp. 2054-2067, 2011/11/01/ 2011, doi: https://doi.org/10.1016/j.compedu.2011.04.009.

[24] K. A. A. Gamage, D. I. Wijesuriya, S. Y. Ekanayake, A. E. W. Rennie, C. G. Lambert, and N. Gunawardhana, "Online Delivery of Teaching and Laboratory Practices: Continuity of University Programmes during COVID-19 Pandemic," *Education Sciences*, vol. 10, no. 10, doi: 10.3390/educsci10100291.

[25] E. K. Faulconer and A. B. Gruss, "A Review to Weigh the Pros and Cons of Online, Remote, and Distance Science Laboratory Experiences," (in En), *International Review of Research in Open and Distributed Learning*, vol. 19, no. 2, 2018, doi: https://doi.org/10.19173/irrodl.v19i2.3386.

[26] F. Esquembre, "Facilitating the Creation of Virtual and Remote Laboratories for Science and Engineering Education**Research supported by the Spanish Ministry of Economy and Competitiveness (Grant MTM2014-52920-P) and the Fundaci_on S_eneca, Research Agency of the Region of Murcia, Spain, (Grant 19294/PI/14)," *IFAC-PapersOnLine*, vol. 48, no. 29, pp. 49-58, 2015/01/01/ 2015, doi: https://doi.org/10.1016/j.ifacol.2015.11.212.

[27] R. Raman, K. Achuthan, V. K. Nair, and P. Nedungadi, "Virtual Laboratories- A historical review and bibliometric analysis of the past three decades," *Education and Information Technologies*, vol. 27, no. 8, pp. 11055-11087, 2022/09/01 2022, doi: 10.1007/s10639-022-11058-9.

[28] M. Schnieder, S. Williams, and S. Ghosh, "Comparison of In-Person and Virtual Labs/Tutorials for Engineering Students Using Blended Learning Principles," *Education Sciences*, vol. 12, no. 3, doi: 10.3390/educsci12030153.

[29] M. Inonan, A. Paul, D. May, and R. Hussein, *RHLab: Digital Inequalities and Equitable Access in Remote Laboratories*. 2023.
[30] R. Hussein and D. Wilson, *Remote Versus In-hand Hardware Laboratory in Digital Circuits Courses Remote versus In-Hand Hardware Laboratory in Digital Circuits Courses*. 2021.
[31] M. Yamakura, K. Hironaka, K. Azegami, K. Musha, and H. Amano, "The Evaluation of Partial Reconfiguration for a Multi-board FPGA System FiCSW," *Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, 2019, doi: 10.1145/3337801.3337805.
[32] S. Hosny, E. Elnader, M. Gamal, A. Hussien, and H. Mostafa, "Multi-Partitioned Software Defined Radio Transceiver Based on Dynamic Partial Reconfiguration," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, 12-14 Oct 2020 2020, pp. 1-4, doi: 10.1109/ISCAS45731.2020.9181178.
[33] A. Sadek, A. Nassar, and Y. Ismail, "Towards the Implementation of Multiband Multistandard Software Defined Radio using Dynamic Partial Reconfiguration," *International Journal of Communication Systems*, vol. 30, 06/08 2017, doi: 10.1002/dac.3342.
[34] A. Hassan, R. Ahmed, H. Mostafa, H. A. H. Fahmy, and A. Hussien, "Performance evaluation of dynamic partial reconfiguration techniques for software defined radio implementation on FPGA," in *2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*, 6-9 Dec. 2015 2015, pp. 183-186, doi: 10.1109/ICECS.2015.7440279.
[35] A. Kamaleldin *et al.*, "Design guidelines for the high-speed dynamic partial reconfiguration based software defined radio implementations on Xilinx Zynq FPGA," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 28-31 May 2017 2017, pp. 1-4, doi: 10.1109/ISCAS.2017.8050456.
[36] K. A. A. Kumar, "FPGA implementation of PSK modems using partial re-configuration for SDR and CR applications," in *2012 Annual IEEE India Conference (INDICON)*, 7-9 Dec. 2012 2012, pp. 205-209, doi: 10.1109/INDCON.2012.6420616.
[37] O. Machidon, A. Machidon, P. Cotfas, and D. Cotfas, "Leveraging Web Services and FPGA Dynamic Partial Reconfiguration in a Virtual Hardware Design Lab," *International Journal of Engineering Education*, vol. 33, 01/01 2017.
[38] S. Grassi, A. Convers, and A. Dassatti, "FPGA Partial Reconfiguration in Software Defined Radio Devices," *Proceedings of the GNU Radio Conference; Vol 5 No 1 (2020): Proceedings of the 10th GNU Radio Conference*, 2020.
[39] S. Hosny, E. Elnader, M. Gamal, A. Hussien, and A. Khalil, *A Software Defined Radio Transceiver Based on Dynamic Partial Reconfiguration*. 2018, pp. 158-161.
[40] "Vivado Overview." Xilinx. https://www.xilinx.com/products/design-tools/vivado.html (accessed April 2, 2024).
[41] K. Vipin and S. A. Fahmy, "FPGA Dynamic and Partial Reconfiguration: A Survey of Architectures, Methods, and Applications," *ACM Comput. Surv.*, vol. 51, jul 2018, doi: 10.1145/3193827.
[42] O. B. Tariq *et al.*, "High-Level Annotation of Routing Congestion for Xilinx Vivado HLS Designs," *IEEE Access*, vol. 9, pp. 54286-54297, 2021, doi: 10.1109/ACCESS.2021.3067453.

[43] Y. Xiao, A. Hota, D. Park, and A. DeHon, "HiPR: High-level Partial Reconfiguration for Fast Incremental FPGA Compilation," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, 29 Aug.-2 Sept. 2022 2022, pp. 70-78, doi: 10.1109/FPL57034.2022.00022.
[44] "Blackboard." Real Digital. https://www.realdigital.org/hardware/blackboard (accessed March 27, 2024).
[45] "Remote Hub Lab." University of Washington. https://rhlab.ece.uw.edu/ (accessed March 25, 2024).
[46] "LabsLand - Home." LabsLand. https://labsland.com/en (accessed March 29, 2024).
[47] M. Moran, P. Fernandez, and R. Hussein, "Adaptación de un laboratorio remoto de SDR para analizar desigualdades digitales en educación de comunicaciones inalámbricas en Latinoamérica," *Innovaciones Educativas*, vol. 25, pp. 32-43, 12/01 2023, doi: 10.22458/ie.v25iEspecial.4920.
[48] B. Chap, M. Inonan, Z. Zhang, P. Orduna, P. Arabshahi, and R. Hussein, "Board 382: RHLab RELIA: A Remote Integrated Environment forEmbedded Computing and RF Communication Systems," Baltimore , Maryland, 2023/06/25. [Online]. Available: https://peer.asee.org/43077.
[49] M. Inonan and R. Hussein, "MELODY: A Platform-Agnostic Model for Building and Evaluating Remote Labs of Software-Defined Radio Technology," *IEEE Access*, vol. 11, pp. 127550-127566, 2023, doi: 10.1109/ACCESS.2023.3331399.
[50] "ADALM-PLUTO." Analog Devices. https://www.analog.com/en/resources/evaluation-hardware-and-software/evaluation-boards-kits/adalm-pluto.html (accessed April 1, 2024).
[51] "USRP N210." ETTUS Research. https://www.ettus.com/all-products/un210-kit/ (accessed April 6, 2024).
[52] A. V. Shindin, S. P. Moiseev, F. I. Vybornov, K. K. Grechneva, V. A. Pavlova, and V. R. Khashev, "The Prototype of a Fast Vertical Ionosonde Based on Modern Software-Defined Radio Devices," *Remote Sensing*, vol. 14, no. 3, doi: 10.3390/rs14030547.
[53] Z. Wang, X. Shi, W. Wang, and W. Cai, "High-Performance Digital Lock-In Amplifier Module Based on an Open-Source Red Pitaya Platform: Implementation and Applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 72, pp. 1-14, 2023, doi: 10.1109/TIM.2022.3221746.
[54] " Red Pitaya - Swiss Army Knife For Engineers." Red Pitaya. https://redpitaya.com/ (accessed March 19, 2024).
[55] "STEMlab 125-14." Red Pitaya. https://redpitaya.com/stemlab-125-14/ (accessed March 20, 2024).
[56] "SDRlab 122-16." Red Pitaya. https://redpitaya.com/sdrlab-122-16/ (accessed March 20, 2024).
[57] M. Inonan, Z. Zhang, P. Orduna, R. Hussein, and P. Arabshahi, "RHLab Interoperable Software-Defined Radio (SDR) Remote Laboratory," presented at the 21st International Conference on Smart Technologies & Education (STE), Helsinki, Finland, 2024.
[58] P. Anantha and N. S. S. R. K. P. Nalli, "Design of Software Defined Radio (Sdr) Waveforms For Portabilityand Interoperability," *Journal of Aerospace Sciences and Technologies*, vol. 73, no. 3, pp. 200 - 206, 07/27 2023, doi: 10.61653/joast.v73i3.2021.78.
[59] S. T. Arzo, F. Zambotto, F. Granelli, R. Bassoli, M. Devetsikiotis, and F. H. P. Fitzek, "A Translator as Virtual Network Function for Network Level Interoperability of Different

IoT Technologies," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, 28 June-2 July 2021 2021, pp. 416-422, doi: 10.1109/NetSoft51509.2021.9492677.

[60] Q. A. Shah, I. Shafi, J. Ahmad, S. Alfarhood, M. Safran, and I. Ashraf, "A Meta Modeling-Based Interoperability and Integration Testing Platform for IoT Systems," *Sensors*, vol. 23, no. 21, doi: 10.3390/s23218730.

[61] P. Orduña "This was the generic first version, made with Python." Github. https://github.com/remotehublab/rhl-relia-3d-parts/tree/main/v1 (accessed March 25, 2024).

[62] Z. Zhang and P. Orduña. "The third version of cases for ADALM-PLUTO and Red Pitayas." Github. https://github.com/remotehublab/rhl-relia-3d-parts/tree/main/v3 (accessed March 26, 2024).

[63] "Guided Tutorial GRC." GNU Radio. https://wiki.gnuradio.org/index.php/Guided_Tutorial_GRC (accessed March 20, 2024).

[64] Z. Zhang, M. Inonan, P. Orduna, and R. Hussein, "RHLab: Towards Implementing a Partial Reconfigurable SDR Remote Lab," presented at the 21st International Conference on Smart Technologies & Education (STE), Helsinki, Finland, 2024.

[65] A. Baral, S. Deepthi, M. A. Bathija, R. R. Shetty, and D. Raj, "BPSK Demodulation Technique Using Xilinx Vivado," in *Computer Communication, Networking and IoT*, Singapore, V. Bhateja, S. C. Satapathy, C. M. Travieso-Gonzalez, and W. Flores-Fuentes, Eds., 2021// 2021: Springer Singapore, pp. 1-8.

[66] "FPGA lessons." Red Pitaya. https://redpitaya-knowledge-base.readthedocs.io/en/latest/learn_fpga/4_lessons/top.html (accessed March 12, 2024).