UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

REAL-TIME ADAPTIVE PULSE COMPRESSION ON RECONFIGURABLE, SYSTEM-ON-CHIP (SOC) PLATFORMS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

HERNAN SUAREZ
Norman, Oklahoma
2015

REAL-TIME ADAPTIVE PULSE COMPRESSION ON RECONFIGURABLE, SYSTEM-ON-CHIP (SOC) PLATFORMS

A DISSERTATION APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY

_____
Dr. Yan Zhang, Chair

_____
Dr. Sesh Commuri

_____
Dr. Caleb Fulton

_____
Dr. Boon-Leng Cheong

_____
Dr. John Dyer

_____
Dr. John Albert

Acknowledgements

I would like to gratefully thank my advisor, Dr. Yan Zhang, for his guidance, support, and constant encouragement during my graduate study at University of Oklahoma.

My gratitude to my doctoral committee: Dr. Sesh Commuri, Dr. Caleb Fulton, Dr. Boon-Leng Cheong, Dr. John Dyer, and Dr. John Albert for their valuable comments, suggestions and corrections.

In addition, I would like to extend my thanks to the Advanced Radar Research Center (ARRC), its director Dr. Robert Palmer, faculty, staff and students, for their generous assistance during my research at OU.

Last but not least, I would like to thank my parents, Ramon Suarez Castro and Graciela Montalvo de Suarez; and my brothers Antonio, Anibal and Nicolas, for their endless support and encouragement.

Table of Contents

List of Tables

List of Figures

Abstract

New radar applications need to perform complex algorithms and process a large quantity of data to generate useful information for the users. This situation has motivated the search for better processing solutions that include low-power high-performance processors, efficient algorithms, and high-speed interfaces. In this work, hardware implementation of adaptive pulse compression algorithms for real-time transceiver optimization is presented, and is based on a System-on-Chip architecture for reconfigurable hardware devices. This study also evaluates the performance of dedicated coprocessors as hardware accelerator units to speed up and improve the computation of computing-intensive tasks such matrix multiplication and matrix inversion, which are essential units to solve the covariance matrix. The tradeoffs between latency and hardware utilization are also presented. Moreover, the system architecture takes advantage of the embedded processor, which is interconnected with the logic resources through high-performance buses, to perform floating-point operations, control the processing blocks, and communicate with an external PC through a customized software interface. The overall system functionality is demonstrated and tested for real-time operations using a Ku-band testbed together with a low-cost channel emulator for different types of waveforms.

Chapter 1

Introduction

The general operation of a pulsed radar system consists of transmitting electromagnetic (EM) signals to an area of interest. The reflected EM signals from the environment are captured by the antenna and transformed into electrical signals. The radar receiver filters, amplifies and also transforms the radio frequency (RF) signal to an intermediate frequency (IF) signal by mixing the RF signal with local oscillators. A range profile can be generated based on the processed receive signal and its respective round-trip delay time.

Modern radars demand bigger computing power as well as reconfigurable flexibility, which is important for multiple functions. A good example is pulse compression (PC), which is the main focus of this dissertation. Theoretically, in order to increase the radar ability to distinguish nearby targets and maximize the detectable range, it would be necessary to transmit a narrower pulse width with a higher peak energy, which is infeasible due to power limitations of the transmitter, especially for solid-state transmitters. To overcome this problem, the pulse compression technique [1] has been used for decades.

It is known that a long pulse with frequency or phase modulation is able to achieve equivalent spectral bandwidth to that of a narrow pulse. When applying pulse

compression technique, the receiver can compress the modulated long pulse of bandwidth *B* to a pulse width equal to *1/B*, which improves not only the range resolution but also the signal to noise ratio (SNR). Traditional pulse compression commonly uses linear frequency modulation (LFM) due to its simplicity for generation and processing; however, the resultant compressed pulse presents range side lobes in the range gates adjacent to a strong target, which could potentially mask any weak targets [2]. Space and Airborne radars are some example applications, for which range side lobe mitigation is highly desired because the strong scatters from the earth's surface can distort the observations.

Different processing techniques have been investigated to suppress the range side lobes [2-8]. These techniques consider the usage of weighting windows, special waveforms, inverse filters, and adaptive filters. Other techniques are based on adaptive pulse compression (APC) such as the least square (LS) method, as well as optimized and recursive approaches. The reiterative minimum mean-square error (RMMSE) algorithm, derived from the LS method, is a localized optimization-type estimation, which can achieve good performance in terms of retrieving the ground truth [2].

APC algorithms require intensive computation of mathematical operations, for instance, Fourier transforms, matrix multiplications, and matrix inversions. A real-time, high-performance implementation of adaptive pulse compression is a huge challenge to traditional processors due to their fixed architecture and sequential nature of operation. Moreover, in airborne and spaceborne radar applications where size, weight and power consumption (SWaP) are critical constraints, not only the

implementation of efficient algorithms, but also the design of optimal hardware architectures and the use of the appropriate technology are important.

Currently, thanks to the advancement of silicon technology, it is possible to establish a variety of potential real-time and embedded processing solutions with integrated computing resources. These solutions range from general purpose processors (GPP) to application-specific integrated circuits (ASICs). As one of the promising technologies, Field Programmable Gate Arrays (FPGAs) has evolved during the past decades, and incorporated more logic resources, multipliers, memory, high-speed transceivers, processors in a single chip device, and also allowed the interaction between processing units through high-performance buses. The design tools for FPGA have become more matured. In addition, the integration of hardware and software solutions in a single device allowed the design and implementation of customized architectures in a single device to achieve better SWaP, greater reliability and reduced manufacturing cost.

In this dissertation, processor architectures of radar waveform processing, including pulse compression and APC on reconfigurable platforms for radar transceiver optimization are investigated. The objective for this work is to design high-performance System-on-Chip (SoC) processors, which can provide improved target sensing, reduced artifacts, accelerated result generations and reconfigurable capability compared to traditional radar signal processors.

1.1 Expectations of High-Performance Embedded Computing (HPEC) in Radar

The main components of a radar system are the transmitter, antenna, and receiver. Raw radar signals are then passed to the signal processors, which extract useful information regarding targets or scene, and the data processor generates useful information for the users [9]. A radar system can be configured to operate with pulsed or continuous waveforms, with coherent or non-coherent modes [10].

Within a radar receiver, the received signal passes through different processing components, which consist of analog transceivers, digital transceivers with real-time and high computational capabilities, signal processors with efficient data transportation protocol and interface control, and software programs running on CPUs to perform system functions and missions. These characteristics are illustrated in Figure 1-1.

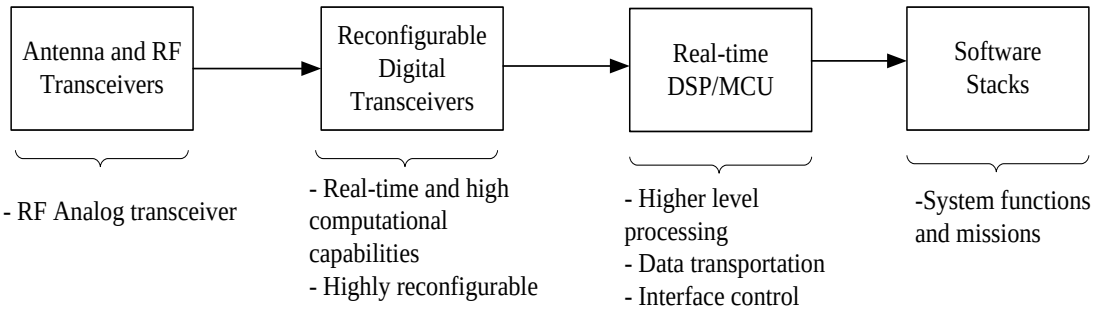


Figure 1-1: Typical functions of a radar receiver.

Initially, radar systems were limited to target detection and range determination functions. With the necessity of more advanced functions in a radar

system, the complexity of digital radar receivers has been increased significantly, which demands more memory, higher speed communication buses, and the computation of large quantities of data in shorter time. Additionally, mobile radars and airborne radars have more SWaP constraints. Modern radar application requires multiple functions, and the computation capability is on the order of GFLOPs and TeraFLOPs, with limited SWaP such as in unmanned aerial vehicle (UAV) and airborne platforms. Furthermore, the digitization at the element level in future phased array radars will increase the computational load to hundreds of TeraOPS for the front-end processing and several TeraFLOPS for the back-end [11].

Surface moving target indicator (SMTI) radar, used to detect and track moving targets on the earth's surface, is a good example of a radar application where the computational load is significant. The computational load for a 48-channel phased array, sampled at 480MHz and 12 bits per sample was estimated in [12]. The processing flow and the number of GOPS for each stage are illustrated in Figure 1-2. It can be observed that pulse compression is one of the processing stages that demands the greatest number of GOPS. The aggregate computational complexity of this system is about 1TeraOPS.

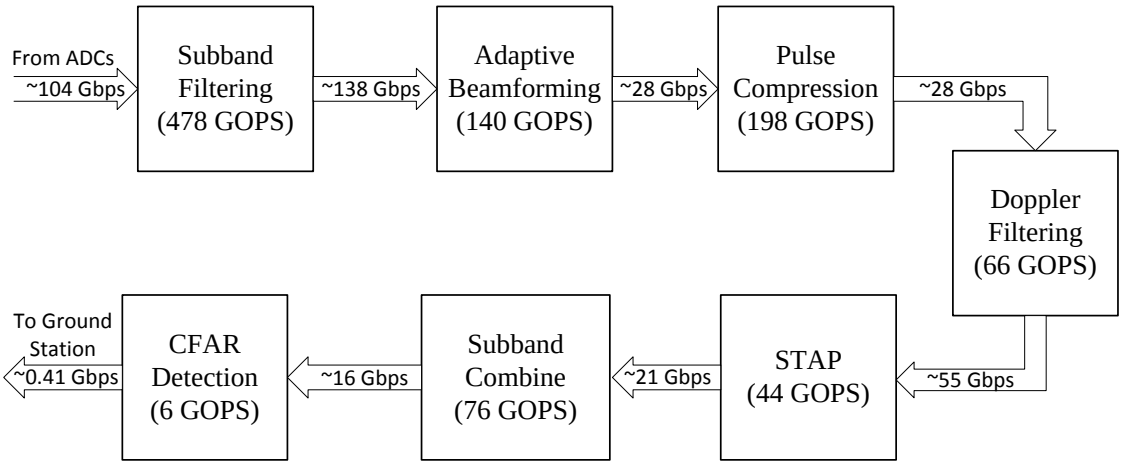


Figure 1-2: Computation load for an example GMTI radar [12]

Historically, to meet these demanding new requirements, manufacturers have been developing more powerful computers or processors by increasing the processor's *clock speed*, but this effort was constrained by physical limitations such as heat dissipation. A new trend is incorporating more processing cores with the intention of executing billions of instructions per second, but the power consumption is increased, and an efficient software application that can take full advantage of all the cores is still absent. This situation has motivated our investigation for hardware-based, reconfigurable parallel computer technologies using FPGAs. The advantages of FPGAs are reconfigurable, low-power, and the software re-programmability.

1.2 Overview of Real-Time Signal Processing Technologies

Gordon Moore estimated in 1965 that the number of transistors on integrated circuits doubles every year [13]. But then in 1975, he updated his estimation to doubling every two years. Nowadays, it is possible to find processing devices with

billions of transistors in a single chip. The processing technologies can be classified in two groups as ‘hardware-based’ and ‘software-based’ solutions.

The hardware-based solutions deliver higher performance with lower power consumption. They can be grouped in application-specific integrated circuits (ASICs) and field-programmable logic devices (FPLDs). On the other hand, the software-based solutions are constituted by programmable processors which by nature execute the instruction in a serial sequence from memory, and possess programmable flexibilities. This group can be divided into two subgroups: the general-purpose processors and the application-specific devices (such as digital signal processors (DSPs), general graphics processing units (GPUs)).

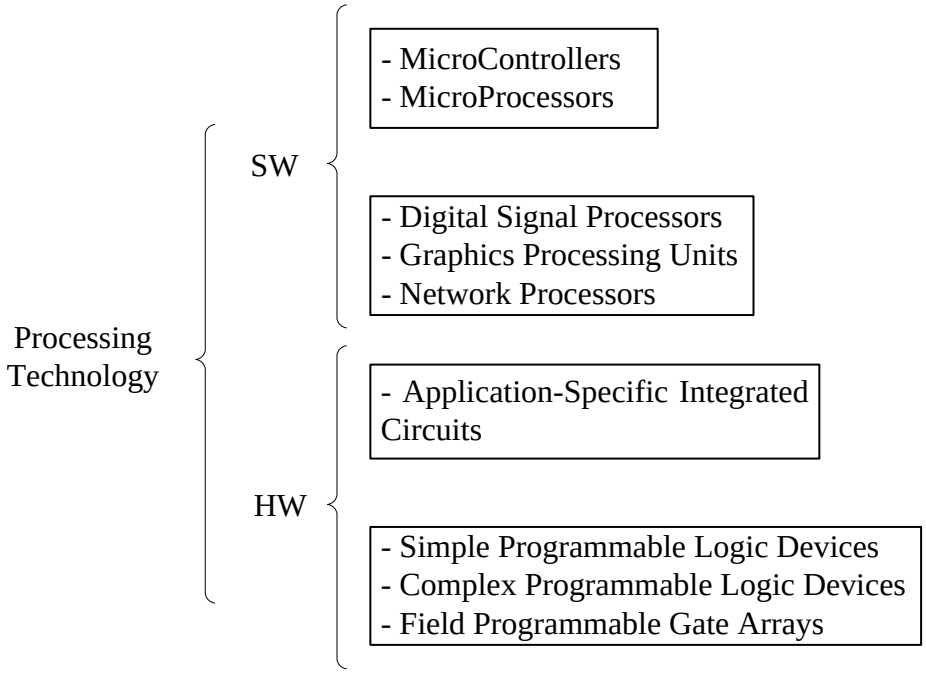


Figure 1-3: Processing technologies classification.

The selection of the appropriate device for a particular radar backend processor is a challenging process because it requires the consideration of several factors such as development cost, performance, hardware resources, power consumption, production cost, time to market, and flexibility. In Figure 1-4, the principal characteristics of some processing technologies are compared. An ideal processing system may incorporate a combination of different technologies and take advantage of the strength of each one.

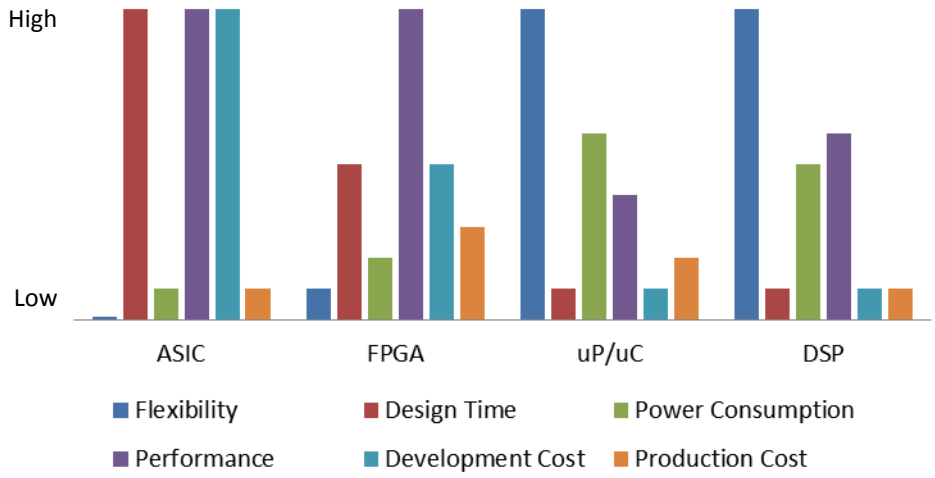


Figure 1-4: Comparison of different technologies for DSP implementation.

A traditional microprocessor (µP) is one of the most common processing solutions in many applications, because of its flexibility to be reprogrammed and relatively low development cost. Most µPs are based on Von Neumann architecture and have inferior computing performance compared to DSPs. Microcontrollers are usually targeted for applications with limited processing requirements. However, some microcontrollers include signal processing engine (SPE) units, such as the Freescale MPC5500 family, which are designed for automotive applications [14].

Because multiply and accumulate (MAC) operations are common to signal processing, DSP devices include dedicated MAC units with particular instructions to accelerate computation. Modern DSPs use Very Long Instruction Word (VLIW) architecture and single instruction multiple data (SIMD) extensions to enhance the levels of data and instruction parallelism.

ASICs and FPGAs are used commonly in applications that require high throughput, especially as front-end signal processors in digital receivers because of their capabilities to handle a large amount of data samples from ADCs. ASIC designs are optimized for specific operations, which can achieve relative small latency and low power consumption, but the fabrication process demands longer time and higher costs, and once the design is fabricated, it cannot be modified. In contrast, FPGAs have the advantage of re-configurability and low power consumption, which are important characteristics for a technology to be considered as a radar front-end signal processor. Moreover, FPGA's computing capabilities are increased if the hard-processor is tightly coupled to create a SoC solution.

1.3 Current State of FPGA Technology

1.3.1 Overview of Device Technologies

The Field Programmable Gate Array (FPGA) was originally developed in the 1980s, and since then it has evolved significantly. The technology has migrated from a simple programmable-logic prototype device to a complex system that includes many hardware resources, such as a large quantity of programmable logic units, dedicated

DSP blocks, embedded processors, block random access memory (BRAM), phase-locked loop (PLL), high-speed gigabit transceivers, and other components. FPGAs are widely used in different areas; the range of applications can be from small digital circuits to larger advanced systems. One important characteristic of FPGAs is re-configurability, which allows the modification of the processing blocks and rerouting of the interconnections to perform a different function without the necessity of powering off. Some FPGAs also offer partial re-configuration capabilities, in which specific areas of the FPGA are modified at the run-time while keeping the other areas operating, which increases silicon reusability significantly.

For a long period, the programmable-logic market was dominated by two vendors, Xilinx and Altera [15]. The XC2000 family was the first FPGA developed by Xilinx and it was comprised of programmable logic units attached to programmable interconnects, and programmable I/O. Altera's first FPGA was based on a PLD structure. The manufacturers have evolved their initial architectures and their programming technologies. Xilinx and Altera FPGAs are based on static random-access memory (SRAM) technology, while Microsemi (previously known as Actel) uses flash and antifuse technology.

The fundamental structure of an FPGA is called 'logic block', which is distributed across the FPGA fabric and interconnected via programmable switches. Xilinx uses the name configurable logic block (CLB), and Altera uses logic element (LE). The content of a logic block also differs from manufacturer to manufacturer. Xilinx's CLB is constituted by two slices; each slice contains a number of look-up

tables (LUTs), storage elements, and multiplexers. For instance, in the Xilinx 7 series FPGAs each slice contains four 6-input LUTs and eight flip-flops [16].

The majority of Xilinx FPGAs are based on 6-input LUTs. Kintex Ultra and Virtex Ultra are the latest families when this dissertation is written, which are based on 16 nm and 20 nm technologies. The on-chip memory size of Spartan and Virtex-4 family is 18 Kbits, while it is 36 Kbits for the 7 Series and UltraScale family. The principal characteristics and the year of introduction of Xilinx and Altera FPGAs are listed in Table 1-1 and Table 1-2, respectively.

Table 1-1: List of Xilinx FPGA families and principal characteristics.

Family	LUT Input	Multiplier	BRAM (Kbits)	µP	Year	Tech (nm)
Spartan 3	4	18x18	18		2003	90
Virtex 4	4	18x18	18	PPC	2004	90
Virtex 5	6	25x18	36	PPC	2006	65
Spartan 6	6	18x18	18		2009	45
Virtex 6	6	25x18	36		2009	40
Artix 7	6	25x18	36		2010	28
Kintex 7	6	25x18	36		2010	28
Virtex 7	6	25x18	36		2010	28
Zynq-7K	6	25x18	36	ARM	2011	28
Kintex Ultra	6	27x18	36		2014	20
Virtex Ultra	6	27x18	36		2014	20

Currently, Altera offers four FPGA families, which are called Cyclone series, Stratix series, Arria series, and Max10 series. As shown in Table 1-2, these FPGAs incorporate memory blocks of different sizes such as M512 (512-bit), M4K (4Kb),

M9K (9Kb), M144K (144Kb), MRAM (512Kb), MLAB (640b ROM/320b RAM), M20K [17].

Table 1-2: Principal Specifications of Altera FPGA Families.

Family	LUT Input	Multiplier	BRAM (Kbits)	µP	Year	Tech(nm)
Cyclone II	4	18x18	4		2004	90
Stratix II	8	18X18	0.5,4,512		2004	90
Stratix III	8	18X18	0.624, 9, 144		2006	65
Cyclone III	4	18x18	9		2007	65
Arria	8	18X18	576		2007	90
Stratix IV	8	18X18	0.624, 9, 144		2008	40
Arria II	8	18X18	9		2009	40
Cyclone IV	4	18x18	9		2009	60
Stratix V	8	27X27	0.640, 20		2010	28
Cyclone V	8	27X27	10		2011	28
Arria V	8	27X27	10	ARM	2011	28
Arria 10	8	27x27	0.640, 20	ARM	2013	20
Stratix 10	8	27x27		ARM	2013	14

Because multipliers and accumulators are essential operations for the implementation of signal processing algorithms, FPGA vendors have included small DSP blocks into the fabric of the FPGA to improve the performance of arithmetic operations and release the logic resources (flip-flops, look-up tables) for other functions. Each Xilinx's DSP block contains two DSP slices. A DSP slice can perform logic and arithmetic functions such as multiply-accumulator, multiply-adder, and counter. DSP slices can also be cascaded to implement more sophisticated functions such as complex multipliers or n–tap FIR filters, thus achieving higher clock rates.

The DSP blocks included in Xilinx FPGAs are called DSP48s, each DSP is mainly composed of a pre-add/subtract unit, a multiplier, and an add/subtract/accumulate engine. Each family of Xilinx's FPGA has a different version of the DSP with some variations in its architecture. DSP48As are included in Xilinx Spartan-3A devices, which consist of an 18-bit pre-adder, followed by an 18x18-bit signed multiplier and a 48-bit add/subtract/accumulate engine. In the Spartan-6 family, it is called DSP48A1, or DSP48E in Virtex-5, and also called DSP48E1 in the Virtex-6 and the 7 series families. The architecture of a DSP48E1 is shown in Figure 1-5, where the main components are a 25-bit pre-adder, 25x18 multiplier, and pattern detector. In the Xilinx UltraScale family, it is called DSP48E2s, and the multiplier and pre-adder width is increased to 27x18 bits and 27 bits, respectively.

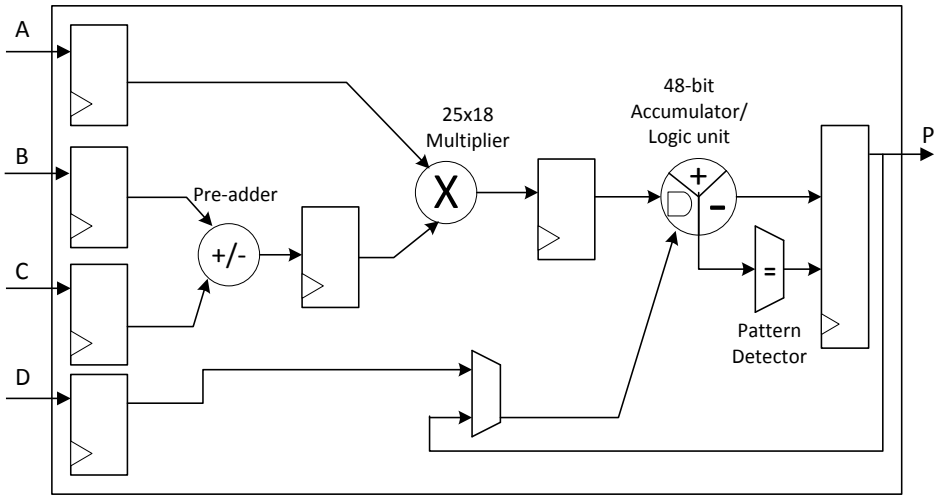


Figure 1-5: Xilinx's DSP48E1 architecture [16]

Another important resource included in an FPGA is fast carry circuitry to perform faster arithmetic operations.

For instance, Xilinx includes dedicated carry logic blocks, called CARRY4 [18]. The logic elements of each block are shown in Figure 1-6.

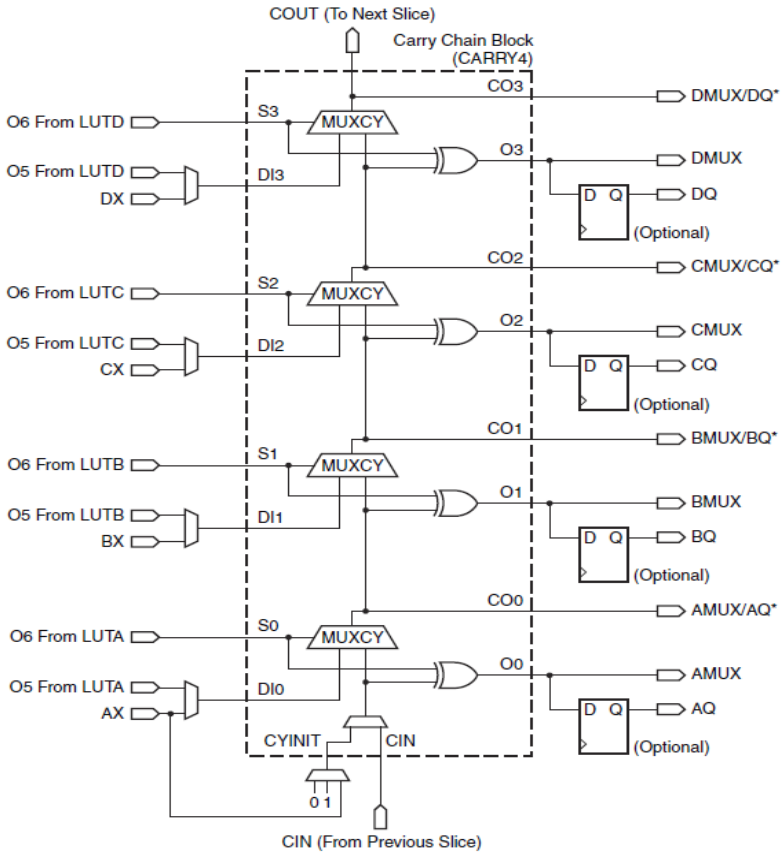


Figure 1-6: Xilinx's carry logic slice architecture [18]

1.3.2 Design Flows

The traditional FPGA design flow is shown in Figure 1-7. The design starts with the description of the system architecture using a hardware description language (HDL), which may include prebuilt functions or intellectual property cores (IPs). HDL design files are synthesized to register-transfer level (RTL), then an implementation process is performed, which consists of three steps: translate, map, place and route.

Finally, a bitstream file is generated, which is downloaded and used to configure the FPGA device.

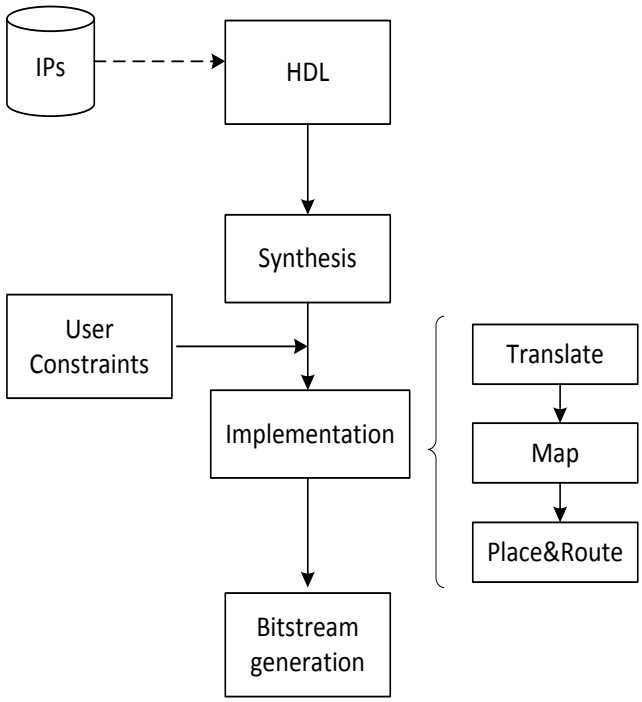


Figure 1-7: Traditional FPGA design flow.

1.3.3 IP Cores

Intellectual property (IP) cores are prebuilt functions that provide solutions to enhance system implementation productivity. IP cores are available for a variety of general functions from simple arithmetic operations to microprocessors, as well as for specific applications such as signal processing, video, networking, storage, and other areas. FPGA vendors offer both own and third-party IP libraries, including IP programs such the Xilinx Alliance program and the Altera Megafunctions Partner Program (AMPP). IP cores can be classified as soft IP, firm IP, or hard IP. A soft IP

core is distributed as synthesizable files describing the register transfer logic of a design. The advantages of a soft IP include flexibility, scalability and portability. A firm IP is delivered in the form of synthesized netlists, which has a certain type of technology dependence. A hard IP core is presented as a mask layout with characteristics such as high performance and predictable functionality, but limited flexibility for system optimization.

1.4 System on a Chip (SoC)

1.4.1 Introduction

According to [19], the evolution of digital design styles occurred in three stages. The first stage, *system-on-backplanes*, was based on several printed boards with specific functions interconnected through the backplane to form a system. This architecture evolved to *system-on-board*, in which processing components were mounted on a single board. And the third stage, *system-on-chip (SoC)*, integrates the board-level functionalities into a single device, resulting in a design with more efficient data exchange between processing units, better computing performance, and improved SWaP compared to a system of discrete components [20].

A SoC design includes at least one microprocessor to run the software component of the system, memory attached to the processor for temporary storage of data and instructions, and peripherals, which can be a coprocessor, a soft-core/hard-core IP, additional memory, or general input and output ports. Processor and peripherals are interconnected via standard buses. A general representation of a SoC solution is shown in Figure 1-8.

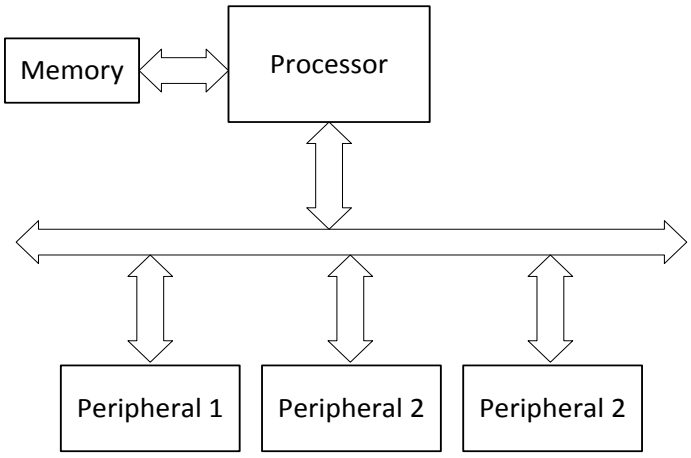


Figure 1-8: Basic concept of a generic SoC architecture.

Early studies proposed different reconfigurable architectures which combined reconfigurable fabric and a programmable processor. For instance, the hybrid architecture Garp was proposed in [21]; the system included a reconfigurable co-processor which was connected to a MIPS-II processor in the same die. The co-processor was also able to access the processor's data memory and the external shared memory through dedicated interconnections. In [22], the authors proposed a reconfigurable architecture called PipeRench; this reconfigurable fabric allowed pipelined reconfiguration of the processing blocks through a hardware virtualization process. A drawback of this architecture was the bandwidth limitations between the main memory, the PipeRench fabric, and the host processor, since PipeRench was connected as a coprocessor. Chimaera architecture was proposed in [23]; this architecture consisted of a small reconfigurable functional unit integrated into a microprocessor itself, reducing the communication bottleneck and taking advantage of the reconfigurable fabric to general-purpose computing. Other reconfigurable systems

were proposed in: PRISM [24], PRISM-II [25], OneChip [26], REMARC [27], MOLEN [28], XiRISC [29], etc.

Xilinx and Altera offer processors in the form of soft-core IP or hard-core IP. Soft-processors are built from logic resources of the FPGA. Xilinx's soft-processor is called MicroBlaze, which is a 32-bit processor with reduced instruction set computing (RISC) architecture. Similar architecture is adopted for Altera's soft-processor, Nios II, which has three different versions: Nios II/f (performance), Nios II/s (performance and low cost), and Nios II/e (low cost).

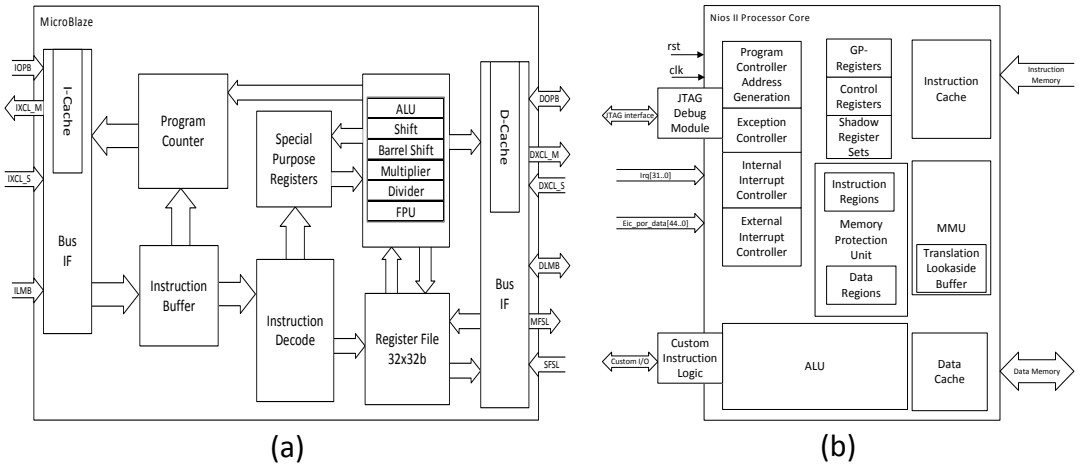


Figure 1-9: Block diagrams of a Xilinx MicroBlaze Processor [30] and an Altera Nios II Processor [31].

Some FPGA families incorporate hard microprocessors into their logic fabrics. For this type, there are commercial FPGA families available in the market such as Xilinx Zynq, Microsemi SmartFusion, and Altera Arria. Earlier Xilinx Virtex families include PowerPC processors, which are based on Harvard architecture and can run up to 550 MHz. The Xilinx Zynq architecture comprises two main units: the Processing

System (PS) and the Programmable Logic (PL), which are interconnected through dedicated Advanced eXtensible Interface (AXI) buses. The PS unit is basically a dual-core ARM Cortex-A9 processor operating at clock speeds up to 1 GHz. Each core is connected to optimized computational units, such as a media processing engine (MPE) or a floating-point processing unit (FPU). Different cache levels are also part of the system, which are controlled by a memory management unit (MMU). A snoop control unit (SCU) interfaces the L1 and L2 caches to ensure consistency of data between them. The processor includes separate L1 caches for data and instructions with a size of 32 KB. The two cores also share a larger L2 cache of 512 KB for instruction and data. In addition, there is 256KB of on-chip memory within the application processing unit (APU).

A SoC design involves hardware logic and programmable processors. Figure 1-10 shows a general overview of the process to implement a SoC solution. This process starts with the specifications of the system, followed by modelling the algorithm. Partition of the design between hardware and software is then performed. Hardware and software partitions follow independent paths first, then they are integrated to obtain the final product and ensure all the parts are tightly coupled.

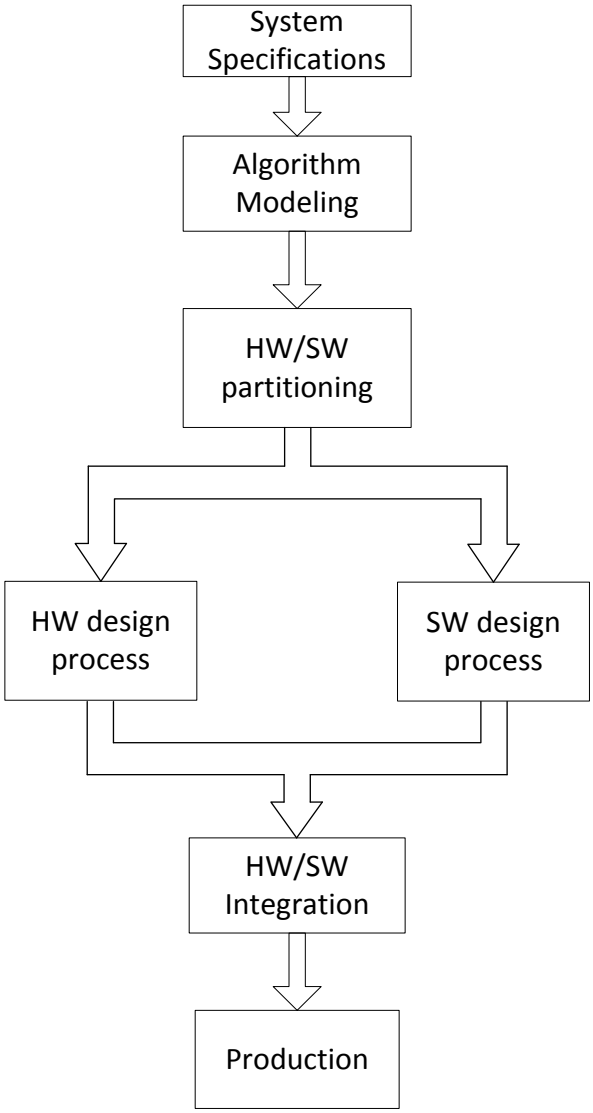


Figure 1-10: General SoC system implementation model.

1.4.2 Hardware/Software Partitioning

The hardware/software (HW/SW) partitioning, or hardware/software co-design, involves the identification of tasks that are more suitable for hardware or software implementation. HW/SW partitioning is a key process in the design of an embedded system because it can have a considerable impact on the performance of the

overall system. There is no tool that can do this process automatically. Due to its parallel nature, programmable hardware is preferred for tasks that are repetitive, and can be efficiently split into multiple and concurrent tasks. Dynamic and unpredictable tasks are better suited for a software-based implementation [20].

Dynamic range is another important factor when deciding the appropriate partition implementation. Traditionally, a general-purpose processor (GPP) has been used for floating-point tasks, due to their special math engines and dedicated floating point units. On the other hand, FPGAs used to be mostly for fixed-point implementations, since floating-point implementations demand much more logic resources. However, in modern FPGAs this is compensated with the increased number of logic resources and DSPs. Therefore, tasks that require floating-point format representation can be implemented in either software or hardware.

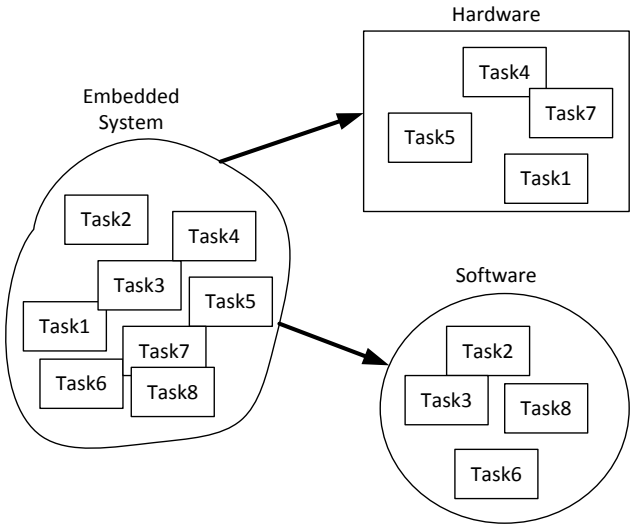


Figure 1-11: Hardware/software partitioning

1.4.3 Advanced eXtensible Interface (AXI) Interconnect Technology

Another essential component in a SoC design is the bus interconnection that coordinates and moves data between the different processing units within the FPGA. Xilinx's interconnect technology prior to the 7 series family is based on the IBM CoreConnect standard, which includes three types of buses: the processor local bus (PLB) for high-speed transactions, the On-chip Peripheral Bus (OPB) for I/O devices, and the device control register (DCR) for configuration and status.

The Xilinx 7 series and UltraScale FPGAs are based on the AXI AMBA 4 standard. AXI was proposed by ARM Holdings public limited company (ARM). This standard defines three types of buses: AXI4, AXI4 Stream, and AXI-Lite. AXI4 is a high-performance bus for memory mapped links, and supports data burst transfer up to 256 data words with data width from 32 to 1024 bits. AXI4-Stream is a non-address based bus with unlimited data bursts, and AXI-lite interconnect is intended to interconnect slow peripherals or control/monitor signals from processing units.

Figure 1-12 shows the top-level architecture of the AXI interconnect core. The core consists of slave interface, master interface, and processing blocks. The crossbar routes the traffic on the AXI channels between the devices connected to the master and slave interface [9].

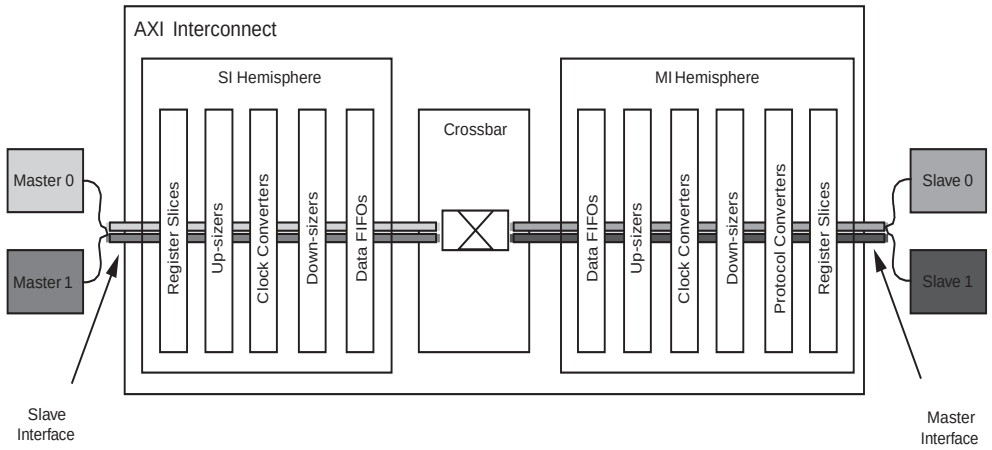
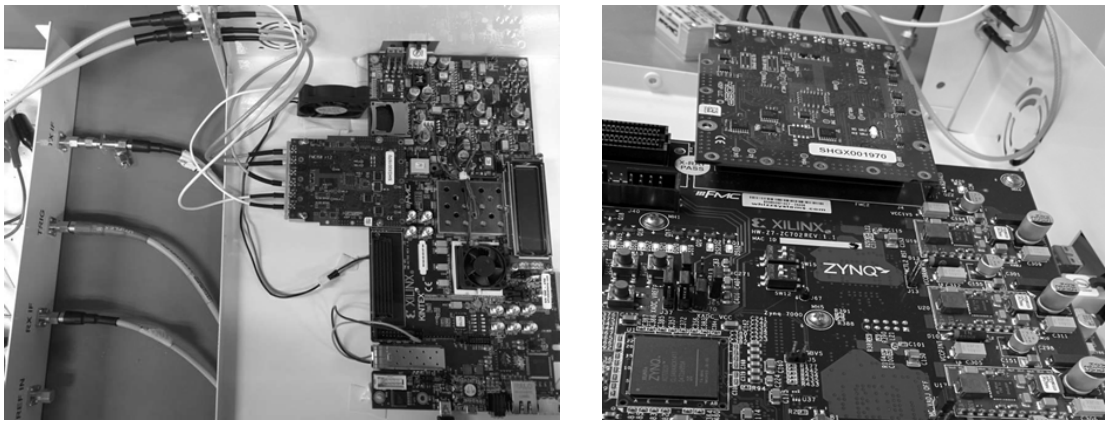


Figure 1-12: Top-level AXI interconnect [32].

1.4.4 Evaluation Platforms

Today's FPGAs are gaining more and more computing power. Xilinx claims 987 GFLOPS peak computing power in a single Virtex-7 980XT FPGA and Altera claims close to 1 TFLOPS peak computing performance for the Stratix V FPGA. The same design and implementation procedure developed in this work can be applied to different and smaller devices. Specifically, we have used two different platforms in most of this dissertation: the KC705 DSP evaluation board and the Avnet ZedBoard 7020 baseboard.

The KC705 board includes an XC7k325t-2-ffg990 FPGA, which has 356K logic cells, BRAMs memory with a total of 16,020KB, and 840 DSP blocks. The ZedBoard's FPGA is the XC7Z020-CLG484-1, which includes a dual-core ARM Cortex-A9 and 85K logic slices, 4,480KB of BRAM, and 220 DSP blocks. Figure 1-13 shows the photos of KC705 and ZedBoard evaluation boards.



(a) (b)

Figure 1-13: Testbed for the implementation of APC. It includes a Ku-band transceiver, (a) Kintex-7 and (b) Avnet ZedBoard evaluation boards.

1.5 Dissertation Outline

The main contribution of this work is developing a general FPGA based SoC framework for radar signal processing, and demonstration of this framework through Xilinx FPGA devices, for specific pulse compression algorithms.

This dissertation is organized as follows: Chapter 2 provides an overview of pulse compression technologies and algorithms. This chapter also introduces the concept of adaptive pulse compression (APC) and its application to modern radar systems. Different algorithms, as well as their computational load requirements are summarized.

Chapter 3 describes the principal processing cores used for the implementation of APC (and other adaptive processing). Hardware implementation of linear algebra operations, such as matrix multiplication and matrix inversion are also discussed.

The details of FPGA implementation of traditional pulse compression is presented in chapter 4. Tradeoffs between different specific design approaches are also discussed.

Chapter 5 focuses on APC processing implementations, and compares various SoC architectures based on basic units of Chapter 4, summarizes design considerations and hardware design results. The achieved performance of the SoC implementations of APC are also discussed.

Finally, Chapter 6 addresses the long-term roadmap of embedded processors and proposed future work for high performance, embedded radar processors.

Chapter 2

Adaptive Pulse Compression (APC) and Implementations

The range (R) between a pulsed Doppler radar and a point target is calculated based on the round-trip travel time of the pulse (T): $R = cT/2$, where (c) is the speed of light. The radar's range resolution is defined as the ability of the radar to resolve objects in range [10]. The maximum detection range can be increased by transmitting a longer pulse width, since more energy is transmitted in the pulse, but a longer pulse can degrade the range resolution (ΔR). In order to improve the radar range resolution and maximize the detectable range, it would be necessary to transmit a narrower pulse width with a higher peak energy, which is generally not feasible due to power limitations of the transmitter, especially for solid-state transmitters. Pulse compression overcomes this problem by using a long pulse modulated in frequency or phase to achieve a similar spectral bandwidth of a short pulse, the long pulse is then "compressed" by the receiver to a width equal to $1/B$, and the range resolution is improved to $\Delta R \approx c/2B$. This improvement can also be represented by a factor called pulse compression ratio (CR), which is approximately the multiplication between the pulse width (τ) and the bandwidth (B), and it is usually much larger than unity. However, traditional pulse compression presents some drawbacks since it uses linear frequency modulated waveforms. In recent years, there are numerous waveform

modulations and processing techniques that have been developed in order to overcome hardware constraints, and improve target detection, interference mitigation, and others.

In general, the radar waveform modulation scheme can be classified as frequency and phase modulation. Frequency modulation waveforms can use either linear or nonlinear modulations. On the other hand, phase modulation can use either biphase or polyphase waveforms.

A filter that maximizes the SNR at the radar receiver is referred to as the matched filter, which is derived from the transmit waveform. Waveform properties such as SNR, range resolution, and Doppler tolerance can be defined in terms of the matched filterer response [10]. The output of the matched filter is equivalent to the cross-correlation between the received signal and the transmit signal. The matched filter is expressed as [1]:

$$MF = \int_{-\infty}^{\infty} s_r(t)\, s^*(t - T_R^{'})\, df \tag{2.1}$$

where, $s_r(t)$ is the received signal, $s(t)$ is the transmit signal, $T_R^{'}$ is the estimate of the time delay, and $(\bullet)^*$ denotes the complex conjugate.

The traditional matched filter can generate significant sidelobes in the range gates adjacent to a strong target, which could potentially mask the presence of smaller targets. For applications such as satellite-borne weather radar the range sidelobes generated from the earth surface can distort the measurements, so that low range sidelobes are highly desired [33], preferably lower than 60dB for light rain measurements [34]. Different types of waveforms, which are designed to achieve low

sidelobe levels, have been studied in [7, 33, 35-37], each type of waveform has advantages and drawbacks.

The characteristics of a radar waveform such as range resolution, range sidelobe level, spacing and range ambiguities, Doppler resolution, Doppler sidelobe level, and spacing of Doppler ambiguities [35], can be described by the *ambiguity function* (AF). AF is derived from equation (2.1) by replacing $s(t) = u(t)\exp(j2\pi f_0 t)$, and obtaining:

$$\left|\chi(\tau, f_d)\right|^2 = \left|\int_{-\infty}^{\infty} u(t)\, u^*(t+\tau)\, e^{j2\pi f_d t} dt\right|^2 \tag{2.2}$$

In which $u(t)$ is the complex envelop of the signal, τ is the time delay, and f_d is the Doppler frequency which is the difference between the received signal and the nominal values expected by the matched filter.

2.1 Pulse Compression Waveforms

2.1.1 Frequency Modulated Waveforms

The most common pulse compression waveform that has been used in radar systems is the linear frequency modulated waveform (LFM) because it is easy to generate and process. The LFM waveform with rectangular shape, bandwidth (B), and pulse width T is represented as:

$$s(t) = \frac{1}{\sqrt{T}} \mathrm{Rect}\left(\frac{t}{T}\right) \exp(j\pi K t^2) \tag{2.3}$$

in which K is the slope frequency which is equal to the ratio between the bandwidth (B) and the pulse width (T). Using equation (2.2) to find the ambiguity function of $s(t)$, the following expression is obtained:

$$\left|\chi(\tau, f_d)\right|^2 = \left|\frac{1}{T}\int_{-\infty}^{\infty} Rect(\frac{t}{T})Rect(\frac{t+\tau}{T})\exp(i\pi Kt^2 - j\pi K(t+T)^2)\right|^2 \qquad (2.4)$$

Therefore, the ambiguity function of an LFM waveform can be written as:

$$\left|\chi(\tau, f_d)\right|^2 = \left|(1-\frac{|\tau|}{T})sinc(\pi(f_d - K\tau)\left(T-|\tau|\right))\right|^2, \quad |\tau| \le T; \quad 0 \text{ elsewhere} \qquad (2.5)$$

Here τ is the shift in time, and f_d is the Doppler shift defined as the difference between the received signal and the nominal values expected by the matched filter.

The autocorrelation of the LFM waveform is essentially a *sinc* function shape with high peak sidelobes of approximately -13.2 dB, and sidelobe levels decreasing at a rate of -4 dB per sidelobe interval. The common method for mitigation of the high sidelobe levels is applying weighting functions to the signal spectrum, but this method may cause SNR loss and degradation of the range resolution. In [7], range sidelobe of -55 dB was achieved by *weighting* the amplitude of the transmit waveform using a half-wave sinusoidal function, with the transmitter operating in the linear rather than saturation region. However, in order to avoid power efficiency degradation in the transmitters, it is preferable to perform the weighting process only on the receiver. The common window functions are Hamming, Kaiser, Hanning, Blackman, etc. Details about their characteristics can be found in [38, 39].

Another method to achieve low range side lobes is through the design of specific non-linear frequency modulated waveform (NLFM) with a suitable matched-filter signal spectrum, where the non-linear rate of the frequency variation plays the same role as amplitude weighting of the spectrum without affecting the radar transmitter efficiency. The complexity of an NLFM might be higher than LFM waveform, but it can provide low sidelobes without the SNR loss caused by a mismatched filter [1]. The literature about NLFM design is vast. For example, [8] described a method for NLFM pulse compression waveform with a truncated Gaussian spectrum, achieving sidelobes of -46 dB for $TB = 1000$. [33] proposed a piecewise NLFM waveform with range sidelobes less than -60 dB. This design was then improved [40] to a continuous NLFM waveform with side lobes of better than -70dB. More information about other implementations can be found in [41-43].

2.1.2 Phased-Coded Waveforms

Phased-coded waveforms are used widely in airborne radars and even in ground-based weather radars recently, e.g. the Next Generation Weather Radar (NEXRAD) system. The waveform is constituted of a sequence of sub-pulses, also known as ‘chips’; the phase modulation has finite states among these chips. The characteristics of phased-coded waveforms are fundamentally dependent on the coding sequence employed [10]. These types of waveforms can be classified into two groups: biphase and polyphase-coded waveforms.

The phases of a biphase-coded waveform are usually either 0 or π. A well-known binary set of codes is the Barker codes, for which the sidelobe levels of the compressed pulse are equal. The periodic autocorrelation function is given by the following equation:

$$\chi(m,0)=\sum_{k=0}^{N-1}c_k c_{k+m}=\begin{cases}N, & m=0 \pmod N\\ a<N, & m\neq 0 \pmod N\end{cases} \tag{2.6}$$

where N represents the code length. The seven Barker codes and their principal characteristics are listed in Table 2-1, where '1' indicates 0 phase and '0' means π radian phase, and the relationship between the peak side lobe level and the code length is given by: $10\log(\frac{1}{N^2})$ [1].

Table 2-1: Binary Barker Codes

Code Symbol	Code Length	Code	Side Lobe level (dB)	Integrated Sidelobe Levels(dB)
B_2	2	11/10	-6	-3
B_3	3	110	-9.5	-6.5
B_4	4	1110/1101	-12	-6
B_5	5	11101	-14	-8
B_7	7	1110010	-16.9	-9.1
B_{11}	11	11100010010	-20.8	-10.8
B_{13}	13	1111100110101	-22.3	-11.5

Another type of binary code is the Maximal Length Sequence (MLS), which is generated using an n-stage linear feedback register. The length of the binary sequence

is $N = 2^n - 1$, where n is an integer, and the typical sidelobes are approximately $10\log(\frac{1}{N})$ [10].

When the phases of the sub-pulses are not limited to the two phases of 0 and π, the code scheme is called polyphase code, and the sidelobe levels are lower than for the binary codes. The Frank code, which is described in [44], is a popular polyphase code. Variants of the Frank code are the P-codes; P1 and P2 are described by Lewis and Kretschmer in [45]. However, due to the very low Doppler tolerance of P1 and P2, two new codes were then developed: P3 and P4 codes [46]. The k-phase value of P3 and P4 codes are defined in the following equations:

$$\Phi_k^{(3)} = \frac{\pi(k-1)^2}{BT} \tag{2.7}$$

$$\Phi_k^{(4)} = \frac{\pi(k-1)^2}{BT} - \pi(k-1) \tag{2.8}$$

In [47], Felhauer proposed the $P(n,k)$ codes, which are derived by step approximation of the phase function of an NLFM waveform, and can improve the peak sidelobe ratio and the tolerance of low Doppler shifts.

Numerous pulse compression waveform designs can also be found in the literature, each design with advantages and limitations. Some examples are: Costas codes [48], Welti Codes [49], complementary codes [50], Hoffman codes [37], and others.

2.2 Adaptive Pulse Compression Algorithms

In the previous section, several types of waveforms designed to mitigate masking problem of the high range sidelobes were introduced. Although those waveforms can achieve low sidelobes, the implementation is constrained to tradeoffs among range sidelobe level, range resolution, energy efficiency, Doppler tolerance, and other hardware-related factors such as the nonlinearity of power amplifiers and calibration errors [51]. In general, optimal waveform design can achieve low-sidelobes but the waveform can become very complicated, and be specific to a particular radar transmitter chain operation state. To further improve the range sidelobe reduction with "waveform independent" ground truth estimation, several adaptive processing techniques have been proposed: mismatched filtering [52], least-squares estimation [53], and inverse filtering [54]. A waveform-independent approached was proposed in [2], which is based on adaptive estimation at each range cell, while reducing the range sidelobes to level of the noise floor.

The basic Least Squares Estimator (LSE) [55] assumes N samples of the time-waveforms are transmitted, denoted as: $s = [s_0\ s_1\ s_2\ ...\ s_{N-1}]^T$, where, $(\cdot)^T$ indicates the transpose operation. The received signal is given by: $y = Sx + \eta$, where the range profile is $x = [x(0)\ x(1) \ldots x(L-1)]^T$, η is the noise vector, and S is the $(L+N-1)xL$ matrix of the transmitted waveform

$$S = \begin{pmatrix} s_0 & 0 & \cdots & & \cdots & 0 \\ \vdots & s_0 & & & & \vdots \\ s_{N-1} & \vdots & \ddots & & & \\ 0 & s_{N-1} & & \ddots & & \vdots \\ \vdots & 0 & \ddots & & \ddots & 0 \\ & \vdots & & \ddots & & s_0 \\ \vdots & \vdots & & & \ddots & \vdots \\ 0 & 0 & \cdots & \cdots & 0 & s_{N-1} \end{pmatrix} \tag{2.9}$$

It is also assumed that the range profile of the ground truth x has length L, and the received signal vector $y = [y(0)\ y(1) \ldots y(L+N-2)]^T$ is the convolution between the transmitted waveform and the ground truth. Therefore, the estimation of ground truth is given by the following equation [56]:

$$\hat{x}_{LS} = (S^H R S)^{-1} S^H R^{-1} y \tag{2.10}$$

R is the covariance matrix of the noise vector η: $R = E[\eta(l)\,\eta^H(l)]$. This estimation requires the complete range profile, which may demand more computing power and larger memory size. An optimized version, truncated LS (TLS), is based on the segmentation of the received signal into *k*-blocks: $\hat{x}_{TLS} = (S_k^T R_k S_k)^{-1} S_k^T R_k^{-1} y_k$, and processing of each individual block.

Another adaptive APC algorithm, which is based on minimal mean-squared error (MMSE) criteria related to the following cost function ($c(l)$), was introduced first in [56].

$$c(l) = E\left[\left\| x(l) - w^H(l) Y(l) \right\|^2\right] \tag{2.11}$$

where, $Y(l) = [y(l)\ y(l+1) \ldots y(l+N-1)]$. Taking the partial derivative of equation (2.11) with respect to w^H, the MMSE filter weights are obtained:

$$w(l) = (E[Y(l)Y^H(l)])^{-1} E[Y(l)x(l)] \tag{2.12}$$

It is also known that $Y(l) = A(l)s + \eta$, where $A(l)$ represents the matrix of the range profile:

$$A(l) = \begin{pmatrix} x(l) & x(l-1) & \cdots & x(l-N+1) \\ x(l-1) & x(l) & \ddots & \vdots \\ \vdots & \vdots & \ddots & x(l-1) \\ x(l-N+1) & x(l-N) & \cdots & x(l) \end{pmatrix} \tag{2.13}$$

The simplified format for the MMSE filter is derived as:

$$w(l) = \left|x(l)\right|^2 \left(\sum_{n=-N+1}^{N-1} \left|x(l+i)\right|^2 s_n s_n^H + R \right)^{-1} s \tag{2.14}$$

Further details can be found in [2]. In addition, [3] proposed a reduced-dimension algorithm for the MMSE adaptive pulse compression. The reduction is based on two forms of dimensionality reduction: decimation and contiguous blocking.

The different adaptive pulse compression algorithms can be grouped into two groups: global deconvolution-type solution and localized optimization-type estimation. Their computational complexity and features are summarized in Table 2-2.

Table 2-2: Comparison of different APC algorithms

Algorithm (1D)	Computational complexity	Features
Matched Filter (MF)[1]	O(N) Per range/angular cell N as the length of waveform	Simplest and standard
Global deconvolution-type solution		
Normal LSE [55]	O(L^2) per gate, L is total number of range gates	Large computation load and sensitive to errors
Segmented LSE[57]	$O(M(\frac{L}{M})^2)$ per gate, M is number of segments	Loss of information due to segmentation (can be improved using random segmentation)
RLS[55]	$O(N^3)$ Per range/angular cell	Reiterative LS
Improved RLS[58]	$O(N^2)$ Per range/angular cell	Reduced computation of RLS
MF-RLS	$O(N^2)$ Per range/angular cell	RLS use MF output as input
conjugate gradient (CG)[59]	$O(L^{1.5})$, L is total number of range gates	Another method to reduce computational load of LS
Localized optimization-type estimation		
RMMSE (APC)[2]	$O(N^3)$ Per range/angular cell	Reiterative MMSE with no prior knowledge of GT
MF-RMMSE[51]	$O(KN+K^3)$ per gate, K is filter length	RMMSE use MF output as input, usually K<<N

Certain radar systems require the completion of these FLOPs operations within a strict time window. For example, Multi-functional Phased Array Radar (MPAR).

2.3 Real-Time Computational Load Requirements of Pulse Compression Algorithms

The *computation complexity* of an algorithm can be measured by estimating the number of floating-point operations (FLOPs). A FLOP is considered any floating-point operation such as add, subtract, multiply, or divide. For instance, the addition of two complex numbers requires two real additions, while complex multiplication requires six operations, including four real multiplications and two real additions. In matrix computation, the number of FLOPs is generally estimated by the amount of arithmetic associated with the most deeply nested statement. In this work, the notation for the number of FLOPs per second is FLOPS.

Traditional digital pulse compression can be implemented in the time-domain by using cross-correlation, in which the number of FLOPS is given by: $K N_{gates} N_{taps} PRF$, where K = 8 and represents the total number of FLOPs for a complex multiplication and addition. N_{gates} and N_{taps} are the number of gates and number of taps, respectively. PRF is pulse repetition frequency. In practice, it is more efficient to implement pulse compression in the frequency domain using Fast Fourier Transforms (FFTs).

Assuming pulse compression is applied to a single receive channel, the computational complexity of the frequency-domain pulse compression can be roughly estimated by this formula [12]:

$$F_{pc} = (2C_{fft-pc} + C_{mult} N_{fft-pc}) PRF \tag{2.15}$$

where, N_{fft-pc} is the number of samples used in FFT/IFFT, C_{fft-pc} is the complexity of the FFT/IFFT for the N_{fft-pc} complex signal samples, and C_{mult} is the complexity of the point-wise complex multiplication.

N_{fft-pc} , C_{fft-pc} , and C_{mult} are factors related to the specific fixed-point implementation architecture and waveforms based on the required number of addition and multiplication. For the basic fixed-point implementations, we have:

$$C_{fft-pc} \approx 5N_{fft-pc}\log_2(N_{fft-pc}) \tag{2.16}$$

The design of waveform diversity that supports pulse compressor must consider the possible reconfigurable capability requirement in terms of N_{fft-pc}. In general, the constraint of N_{fft-pc} is defined by:

$$N_{fft-pc} > 2BW * T_p \tag{2.17}$$

where T_p is the pulse length (duration in μs), and BW is the baseband waveform modulation bandwidth expressed in MHz.

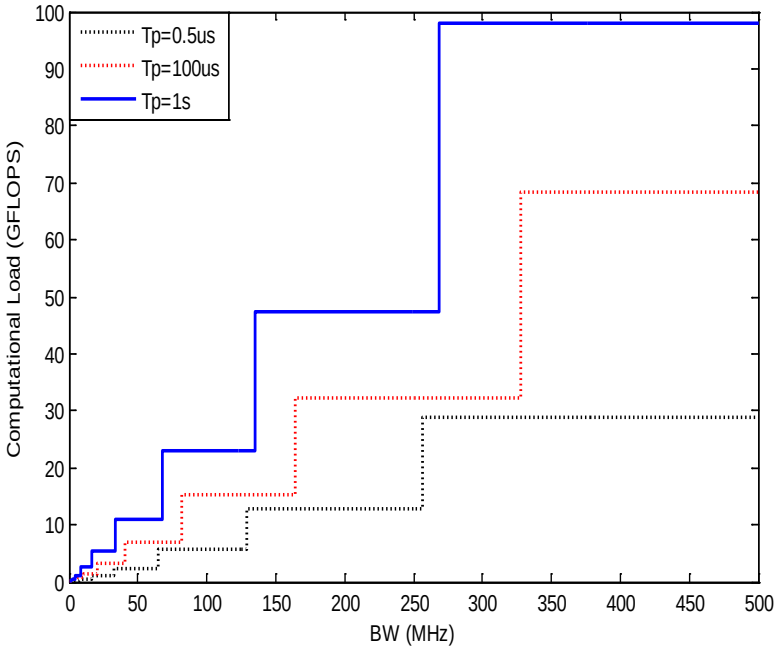


Figure 2-1: Estimation of computational load requirement for real-time matched filter pulse compressor, with different signal bandwidths and pulse length. Assuming 20% transmitter duty cycle for all cases.

Based on equations (2.15)-(2.17), an estimation of real-time computing load requirement for basic matched filter type pulse compression with different waveform parameters can be performed, and some examples are plotted in Figure 2-1. The graph of the estimation presents a stair-stepped shape due to the fixed number of points (power of two) required by the FFT operations. As is seen, for normal weather radar operations, the computational load for a single channel is generally less than 1 GFLOPS, while for wideband noise radar and high-resolution SAR/STAP, the real-time computational load can easily reach more than 40 GFLOPS. A complete digital

processor would also need front-end FIR filtering and implementation of clutter suppression, which can add more computational loads to the processor.

For APC algorithms, much higher computational loads are demanded. The computational cost per stage of the adaptive algorithms is shown in Table 2-3. Where, N is the length of the transmitted waveform, M is the number of subgroups, and K is the length of the MF-RMMSE filter.

Table 2-3: Computational cost of APC algorithms per stage.

Algorithm	Computational Cost (per stage)
Matched Filter (time-domain)	N
RMMSE (APC) [2]	$6N^2+14N$
RMMSE (FAPC) [3]	$\frac{3N^2}{M}(1+\frac{1}{M})+N(1+\frac{13}{M})$
RMMSE (Parallelized)	$N^2(\frac{5}{M^2})+N(4+\frac{13}{M})$
MF-RMMSE [51]	$(2K+1)N+\frac{K(K-1)}{2}$

An example of the estimation of numbers of complex FLOPS for each APC algorithm is shown in Figure 2-2. This example assumes 100 range gates, 30 signal samples, M=5, K=3, and PRF=1 KHz. It can be noticed that the RMMSE algorithm requires a significant number of FLOPS, and the MF_RMMSE approach is able to reduce the complexity to a reasonable number of FLOPs.

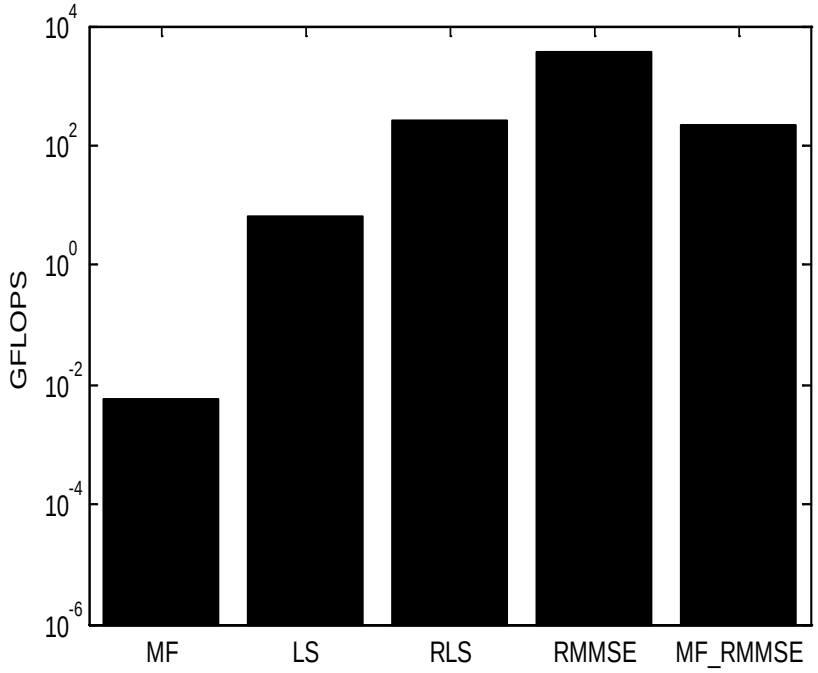


Figure 2-2: Computational analysis of APC algorithms.

Certain radar systems require the FLOPs be computed in a specific and strict time window, which are usually termed as *hard real-time requirement*. For example, the Multi-functional Phased Array Radar (MPAR) potentially requires pulse compression processing to be completed in a time on the order of milliseconds (depending on the coherent processing interval (CPI)). Real-time requirements pose big challenges to the implementation of pulse compression algorithms in radar processors.

2.4 State of the Art of Pulse Compression Implementations

One of the devices that has been used for the implementation of pulse-compression is the surface acoustic wave device (SAW) [60-63]. MESL Microwave

Ltd. [64] offers products based on SAW technology for pulse compression. However, with the advances of silicon technology, digital devices have become an attractive option for the implementation of pulse compression, DSPs, GPPs, GPUs, and FPGAs are some examples. The early implementations usually included more than one chip due to the limited processing capabilities of the processing units; for instance, a dedicated chip for FFT, and others for adders and multiplies. In the example of [65], the radar processor had 64 signal processing (SP) nodes with a maximum sampling rate of 10 MHz. Each node included a TI TMS320C30 DSP, which was connected to a co-processor TMC2310 through a dual-port RAM. The coprocessor was able to execute a 1024-point FFT in 512 μs. [66] proposed a design based on TMS320C25 DSP interconnected to IMS A100 DSP, which performed the LFM pulse compression for small time-bandwidth products, and 8-bit samples with 2.5 MHz sampling frequency. In [67, 68], a prototype of a digital pulse compressor in a single printed-circuit board was presented. The system was clocked at 36 MHz, and power consumption was about 10 W. The PDSP16515 processor was the main component of the system, which was capable of executing 256 and 1024-point radix-4 FFT in 22 μs and 110 μs, respectively. The processor was also connected to an external erasable programmable read-only memory (EPROM), which was used to store the waveforms.

In [69], a processing system for phase coded pulse compression was developed using four INMOS A100 DSPs, a microprocessor and Altera EPLD's. The core structure of this processor was similar to a standard non-recursive filter. The processor's maximum clock speed was 30 MHz. Their design was only tested with simulated data and had some limitations in the functionality such as coherent

integration. But the processor architecture was then integrated into a single device in [70], in which AlteraFLEX10KA100 was the selected device, and configured with a clock frequency of 40 MHz.

DPC using a TMS320C6201 DSP was implemented in [71]. The system included two 12-bit ADCs, clocked at 40 MHz, which were interconnected to the DSP through a synchronous FIFO. The authors indicated that with the DSP clocked at 200 MHz, the digital pulse compressor based on radix-2 FFTs can execute 512 samples within 124 μs.

A multi-processor architecture was studied in [72]. The system included four ADSP-21160 DSPs with 80 MHz clock, 16-bit ADC sampling at 6.6 MHz, and shared external SDRAM where sampled data was stored. Four different approaches to perform the pulse compression in the frequency domain were studied, achieving a processing latency of 1.086 ms for a signal with pulse duration of τ =18 μs and PRF = 833.3 Hz.

As part of the Next-Generation Precipitation Radar (PR-2), [73, 74] designed a radar processor on an Annapolis Wildstar board which contained three Virtex XCV1000-4 FPGAs. Four channels of 12-bit were sampled at 20 MHz with a bandwidth of 4 MHz. The pulse compressor was able to perform 20 GOPS, and the architecture was a 256-tap non-symmetric FIR with a signal template of the 50 μs LFM waveform. An antifuse-based Actel 1280 FPGA was used to mitigate radiation-induced errors in the Xilinx FPGAs, since SRAM-based FPGAs are susceptible to a single event upset (SEU) caused by space radiation.

In [75], a 2-D pulse compressor was implemented on a Xilinx XC2V6000 FPGA, where a MicroBlaze processor was used to control and monitor the system. This architecture was configured to load raw data into a 512 MB DDR SDRAM, and data was then transferred to the pulse compressor's buffers by a DMA controller.

In [76], the implementation of pulse compression in a multi-core platform was presented. The platform included TI C66-core DSPs. The implementation was executed on a TI 6678 evaluation board clocked at 1 GHz, where the compression of 4K samples was performed in 9 μs, with 10 W of power consumption.

2.5 Basic Considerations for Hardware Implementation

The selection of the appropriate number representation format for the implementation of the DPC processor is an important step, since characteristics such as accuracy, dynamic range, and stability can affect the performance of the system, as well as software development costs, hardware system speed, and SWaP.

2.5.1 Number Representation Format

An unsigned fixed-point number is usually expressed in terms of a positive radix (r), the number of digits for the whole part (k) and the fractional part (l). The implicit data set is in the range $\{0,1,\ldots,r-1\}$ [77]. The mathematical representation is given by the following equation:

$$(x_{k-1}x_{k-2}\ldots x_1x_0.x_{-1}x_{-2}\ldots x_{-l})_r = \sum_{i=-l}^{k-1} x_i r^i \tag{2.18}$$

Thus, the binary representation ($r = 2$) of a number can be written as:

$$\sum_{i=-l}^{k-1} x_i 2^i = 2^{-l} \sum_{i=0}^{k+l-1} x_{i-l} 2^i \tag{2.19}$$

However, to implement arithmetic operations in hardware it is more efficient to use the 2's complement representation, since facilities the computation of arithmetic operation that involves signed numbers. The 2's complement representation of a number is obtained as:

$$\sum_{i=-l}^{k-1} x_i 2^i \xrightarrow{2's} 2^{-l}[-2^{k+l-1} x_{k-1} + \sum_{i=0}^{k+l-2} x_i 2^i] \tag{2.20}$$

In (1.20), k and l determine the numerical resolution: 2^{-l} , range $[-2^{k-1}, 2^{k-1} - 2^{-l}]$, and accuracy 2^{-l-1}. For the purpose of this work, a fixed number is denoted as $< k, l >$, where k is the total number of bits for the word and l the number of bits for the integer part.

The main disadvantage of fixed-point format is the limited dynamic range, which is ~6dB per bit. Fixed-point arithmetic operations also require additional operations to prevent or reduce underflow and overflow, which represents a cost in the development and implementation of the design. For instance, in order to guarantee that the sum of *M* numbers of *N*-bits does not overflow, it will be necessary to use $N+log_2(M)$ bits. Moreover, the dot product of *M*-element vectors of *N*-bits will require $2N+log_2(M)$ bits, since the multiplication operation doubles the number of bits. A simple method to prevent overflow is to downscale the operands by shifting some bits prior to the computation, but this approach reduces precision. Other techniques are based on scaling in stages, trying to minimize the precision loss while reducing or avoiding overflow.

As mentioned in the previous chapter, thanks to the advances in silicon technology, modern FPGAs have integrated more logic resources and dedicated cores in a single device, enabling the implementation of floating point arithmetic operations, which has increasingly been practical rather than fixed-point [10]. The advantage of using floating point format is that it can provide high resolution over a large dynamic range. The representation of a floating point number is:

$$X = (-1)^s 1.m2^E \tag{2.21}$$

where s represents the sign, m denotes the mantissa, and E is the exponent A floating-point operation generally demands more hardware resources, since it involves different formatting stages. The Xilinx FPGAs support both single-precession floating point and double-precision floating point. Xilinx's Single-precision floating point format uses a bit for sign, 24-bit fraction and 8-bit exponent [78].

Chapter 3

FPGA Cores for Radar Signal Processing

3.1 Optimized Adder and Multiplier Designs

Addition is a fundamental arithmetic operation. A traditional architecture for adding two numbers is the ripple-carry adder (RCA), in which the carry is propagated from one stage to the next. The RCA architecture is illustrated in Figure 3-1, in which $a = a_{n-1}a_{n-2}...a_1a_0$ and $b = b_{n-1}b_{n-2}...b_1b_0$ are the two n-bit operands, $s = s_{n-1}s_{n-2}...s_1s_0$ is the sum. c_{in} and c_{out} are carry-in and carry-out, respectively.

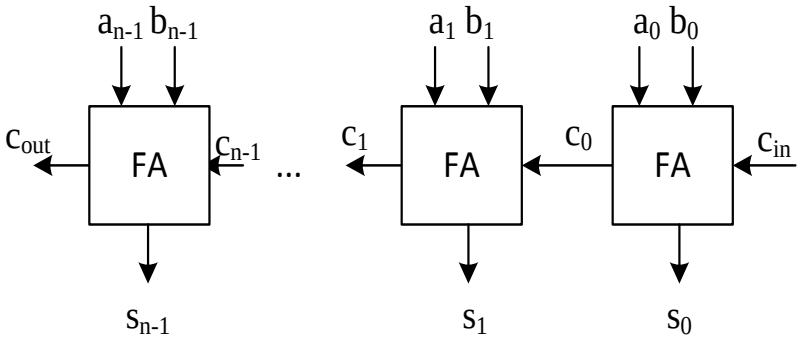


Figure 3-1: Operation of a conventional *n*-bit ripple carry adder.

The RCA architecture has a total computation time of nT_{carry}, where T_{carry} is the delay to generate the carry (e.g., c_{i-1}) in each stage. Therefore, the computation

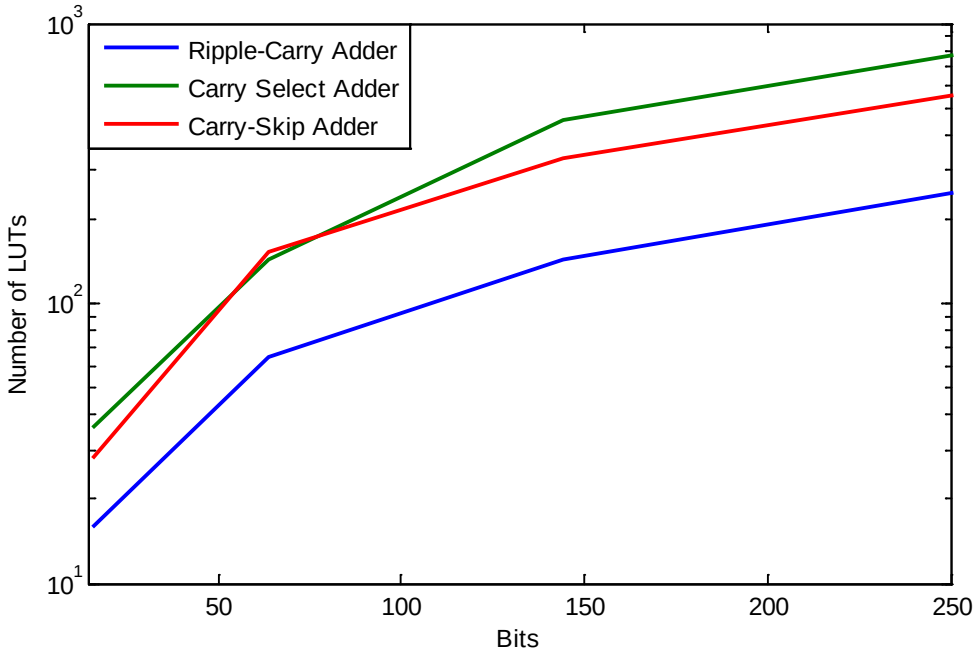
time for this architecture increases linearly with the number of digits, which is a problem when adding larger numbers. Different architectures have been proposed to reduce the path delay between c_{in} and c_{out} in order to achieve shorter computation time. Examples of the fast adder architectures include carry-skip, carry-select, radix-2^k, and conditional-sum [77, 79]. Other architectures such as the *carry-look-ahead* compute the carries at different levels. Assuming:

$$p_i(a_i, b_i) = a_i \oplus b_i \quad , \quad g_i(a_i, b_i) = a_i \odot b_i \tag{3.1}$$

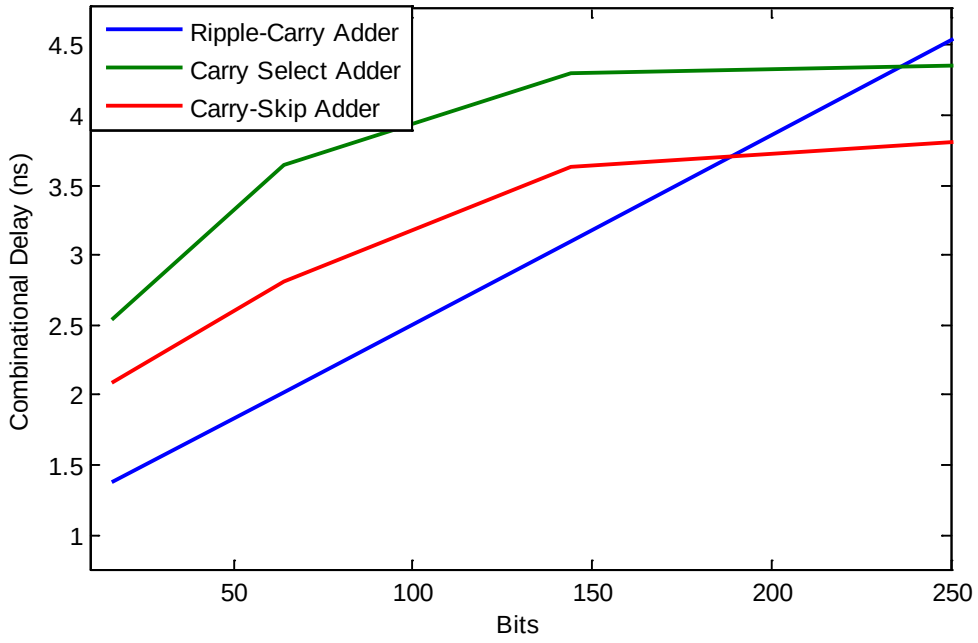
Thus, c_{in} can be expressed as:

$$c_{i+1} = p(a_i, b_i) \cdot c_i + \overline{p(a_i, b_i)} \cdot g(a_i, b_i) \tag{3.2}$$

Three optional architectures, *ripple-carry adder*, *carry select adder and carry-skip adder*, were implemented on a Kintex-7 FPGA (xc7k325t-2-ffg900). The timing and hardware resource estimations are obtained from the synthesis result. Performance in terms of LUT utilizations and combinational delay (delay of the critical path in the circuit) are shown in Figure 3-2.



(a)



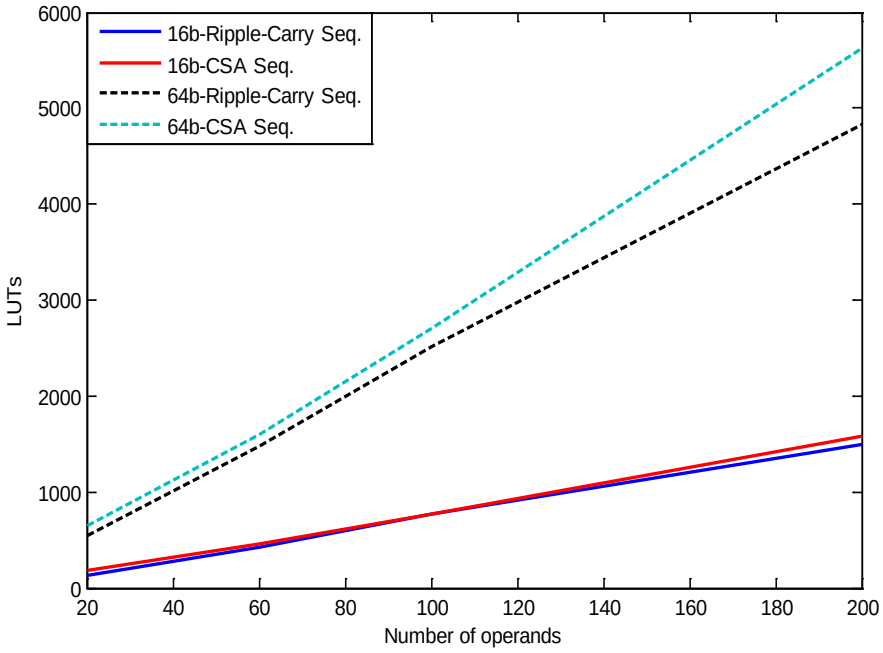
(b)

Figure 3-2: Performance of two-operand adders using different implementations on a Kintex-7 FPGA (xc7k325t-2-ffg900). (a) Number of LUTs, (b) Combinational Delay.

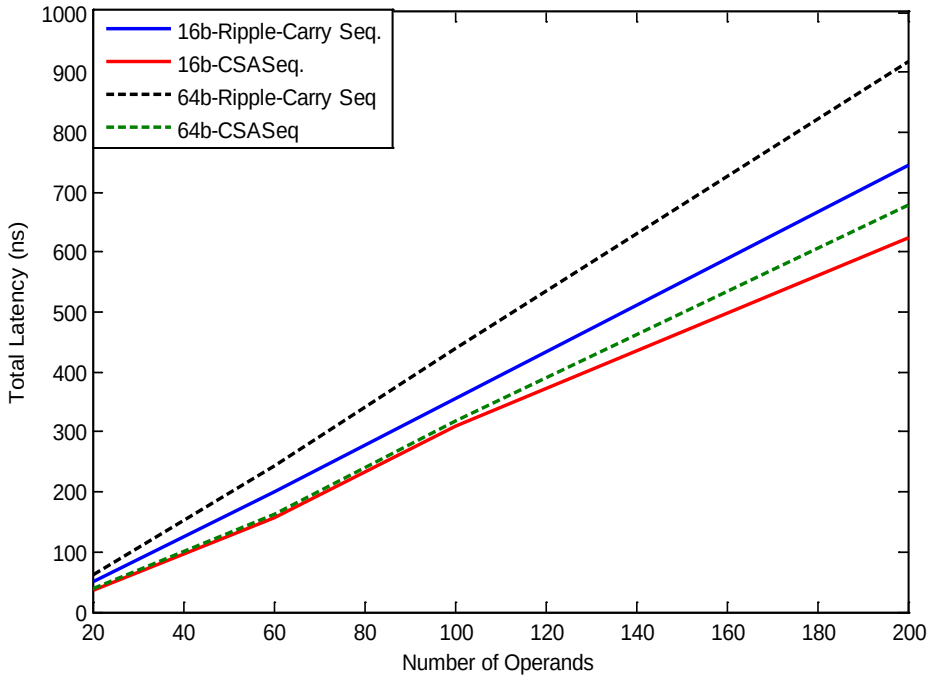
Compared to the other two architectures, the ripple-carry architecture requires fewer LUTs and achieves lower latency for operands with a relatively low number of bits. The ripple carry adder takes advantage of the embedded carry logic circuitry and the regular structure of FPGAs. The advantages of the carry select adder and carry-skip adder are explicit when the number of bits is larger than 200, which is, however, not very common for the radar processors we develop.

Multiple operands are required when computing inner products and other applications. The addition of k n-bit operands, using a tree of two-operand ripple-carry adders requires a computation in the order of $O(n+\log k)$.

A technique to improve the multi-operand computation is called *carry-save adders (CSA)*, in which instead of propagating the carry during each addition, the carry is passed to the next operand, thus reducing the number of inputs from 3 to 2 for computation of each digit. Sequential architectures for CSA and RCA adders were implemented on a Kintex-7 FPGA, and performance for 16-bit and 64-bit word length are presented in Figure 3-3.



(a)



(b)

Figure 3-3: Performance of sequential multi-operand adders for 16 and 64 bits operands on a Kintex-7 FPGA.

The performance results show that CSA achieves lower latency but demands slightly more LUTs than the RCA. The performance difference is more evident for longer word-lengths and larger numbers of operands.

Multiplication is another important arithmetic operation, which can be implemented through multi-operand additions. Consider a multiplicand $a = a_{n-1}a_{n-2}...a_1a_0$, and a multiplier: $b = b_{n-1}b_{n-2}...b_1b_0$. The product of a and b generates a $2n$-bit product $p = p_{2n-1}p_{2n-2}...p_1p_0$.

According to [77], two different approaches can be used to improve multiplication computation: (a) High-radix multipliers to reduce the number of operands, since having a higher radix representation requires fewer number of digits. To further reduce the number of partial products, special encodings such as the booth encoding [80, 81] can be used. (b) Using faster hardware structures to reduce the time required to add the partial products, such as full-tree multipliers (e.g. Wallace's tree [82] and Dadda's tree [83]), partial-tree multipliers, array multipliers, and others [77]. In addition, redundancy representation techniques [84] can be used to have more than one possible representation and provide carry-free additions with a latency independent of the word length, as described in [85-87].

Sequential multipliers can be implemented with the partial-product addition based on optimized architectures such as the shift-add, booth encoding, or carry-save adder. The performance of these three architectures implemented on a Kintex-7 FPGA is compared in Figure 3-4. We can see that the CSA-based sequential multiplier has the lowest latency compared to the other two solutions.

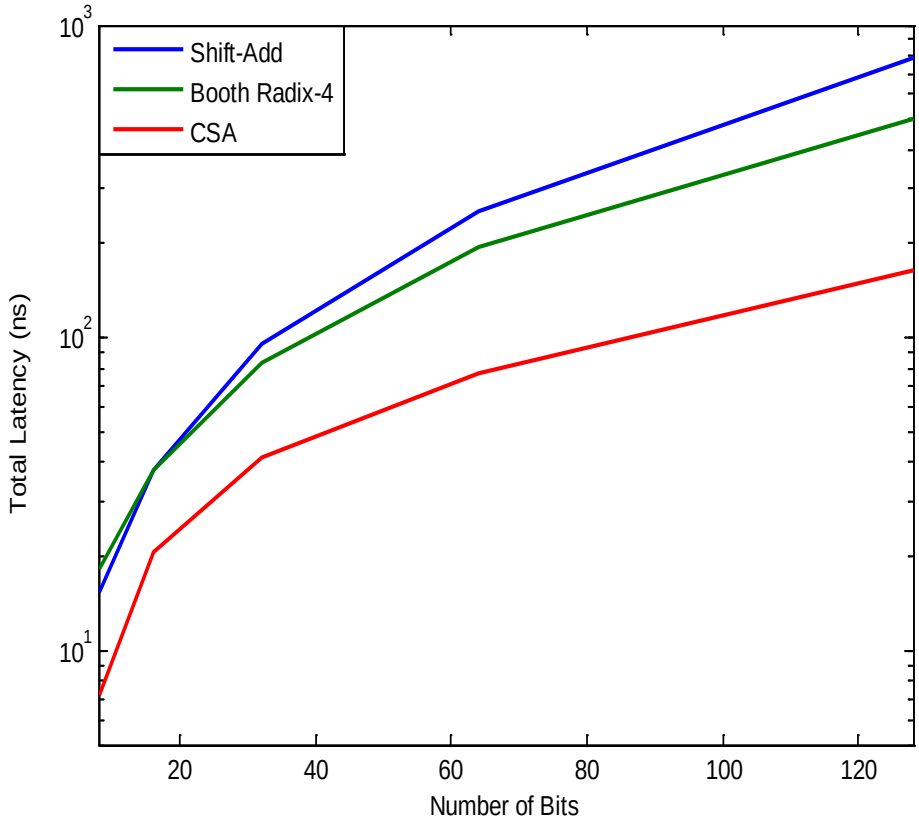


Figure 3-4: Comparison of latency performance of three sequential multipliers through implementation on Kintex-7 FPGA.

In the next steps, the performance of a parallel multiplier using basic designs of CSA and RCA architectures is also compared with that of the commercial designs using Xilinx's dedicated hardware resources/building blocks, including LUT, DSP and CARRY4. The combinational delay for different word lengths is shown in Figure 3-5.

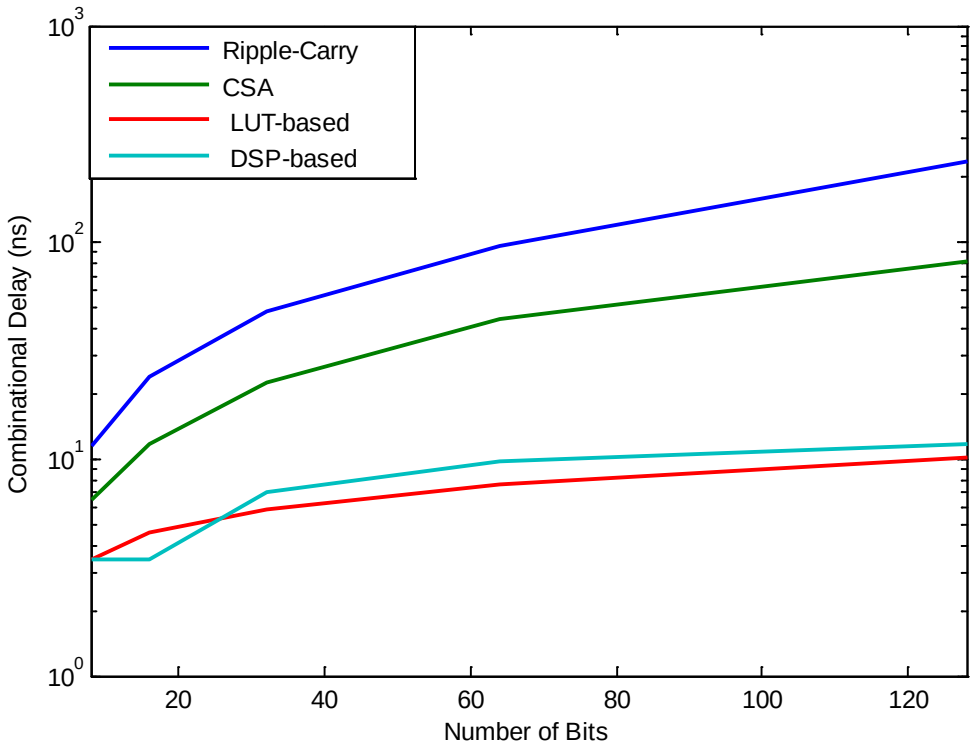


Figure 3-5: Comparison of combinational delay performance of different parallel multipliers including designs using Xilinx commercial building blocks.

The results show that the design based on Xilinx's multiplier, which uses dedicated LUTs and DSPs, achieves shorter combinational time delay compared to the basic CSA and RCA architectures. The reason is that the multipliers based on Xilinx's commercial building blocks are optimized to be more suitable for the structure of the specific FPGA, with faster interconnection and efficient carry chain circuitry. Therefore, CSA is recommended as a good multiplier design for generic hardware while using the Xilinx LUT is more suitable for implementing basic arithmetic operations on Xilinx's specific FPGA devices.

The register-transfer level (RTL) schematic of an 8-bit simple 2's complement adder, using Xilinx Vivado tool, is shown in Figure 3-6. This implementation also includes input/output buffers (IBUF, OBUF), and dedicated carry propagation blocks (CARRY4).

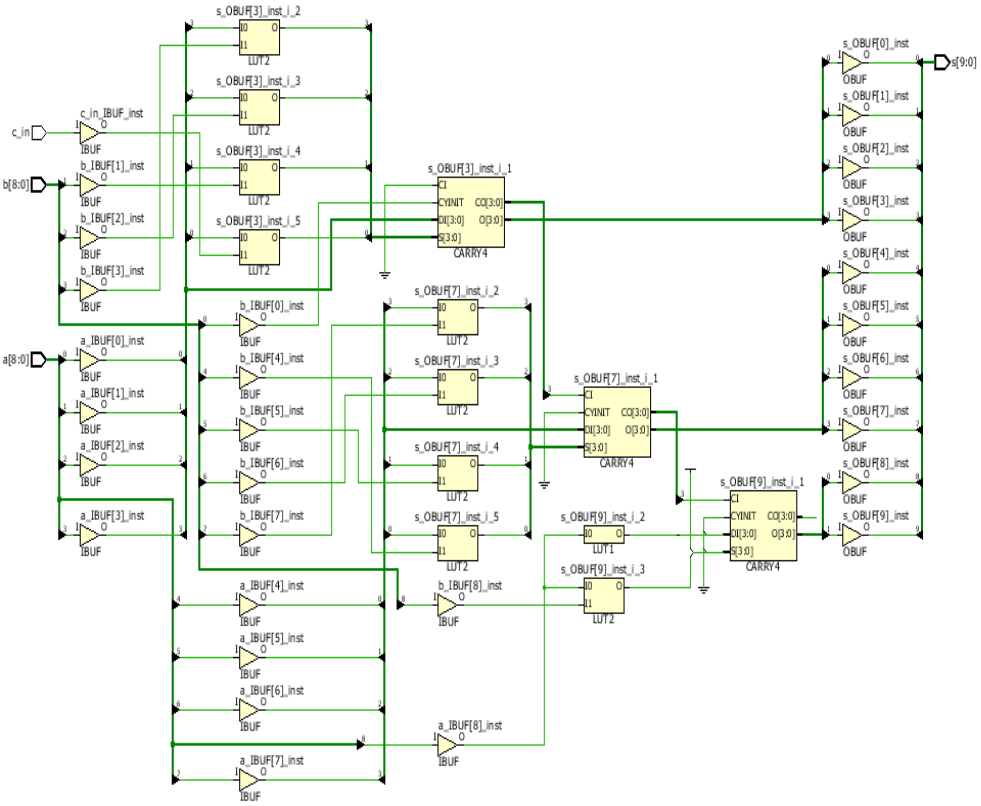


Figure 3-6: Schematic for an 8-bit 2's complement adder on Kintex-7 FPGA.

3.2 Matrix Multiplication

3.2.1 Acceleration Using Coprocessor

Consider matrix $A=(a_{ij})$ and $B=(b_{jk})$ with a dimension of $m\times n$ and $n\times p$, respectively. The matrix multiplication between matrices A and B is expressed in the following equation:

$$c_{ik} = \sum_{j=1}^{m} a_{ij} b_{jk} = a_{i1} b_{1k} + a_{i2} b_{2k} + \ldots + a_{in} b_{nk} \quad , \tag{3.3}$$

in which $i = 1,\ldots,m$, $j = 1,\ldots,n$, $k = 1,\ldots,p$. Each element of matrix C is the dot product of the *ith* row of matrix A with the *jth* column of matrix B. The traditional algorithm for computing matrix multiplication requires the execution of three loops in which the innermost loop performs an addition and multiplication, resulting in a computational complexity of $2mnp$ FLOPs.

Matrix multiplication is an essential step for the implementation of the adaptive pulse compression. As mentioned in Chapter 1 and Chapter 2, the computation complexity ($O(n^3)$) of such algorithms requires a large number of multipliers and adders. For real-time embedded systems, an important model for acceleration is keeping a reasonable load on the main processor, and letting the main processor collaborate with dedicated coprocessors that are designed to perform specific and intensive computing tasks, such as inner products.

A *coprocessor* was implemented on a Kintex-7 FPGA (XC7k325tffg900-2). The architecture of the coprocessor is based on a direct matrix multiplication method fully sequential mode, with buffers as an interface to the buses. More details of this co-processor design will be given shortly. The matrix multiplication coprocessor is attached to a soft-core RISC CPU (MicroBlaze in this case) via the AXI Stream Buses and unidirectional point-to-point links. Two different matrices are stored in the local memory, and matrix multiplication can be computed with and without the coprocessor. A timer module is connected to the slow speed AXI bus, and computation results are

streamed to the PC through the universal asynchronous receiver/transmitter (UART) port.

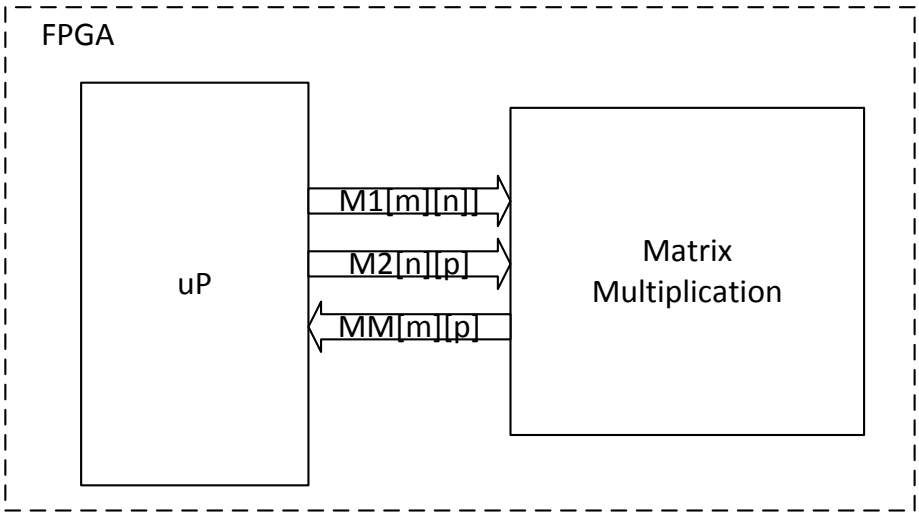


Figure 3-7: High-level configuration of matrix multiplication coprocessor.

The comparison results of matrix multiplication with and without coprocessor are shown in Figure 3-8. The latency includes a number of clocks that the processor takes to write/read data from memory to the AXI bus. To measure the latency, a timer was implemented on the FPGA and attached to the host processor via the AXI Lite bus. For the 4x4 matrix multiplication, the execution time without the coprocessor is 3823 clock cycles, while using the hardware coprocessor it only takes 362 clock cycles, speeding up the computation by about 10 times. Similar performance is achieved for the multiplication of 8x8 matrices, in which the execution time on the processor is 28980 clock cycles, while it is 2557 clock cycles when the hardware coprocessor is used.

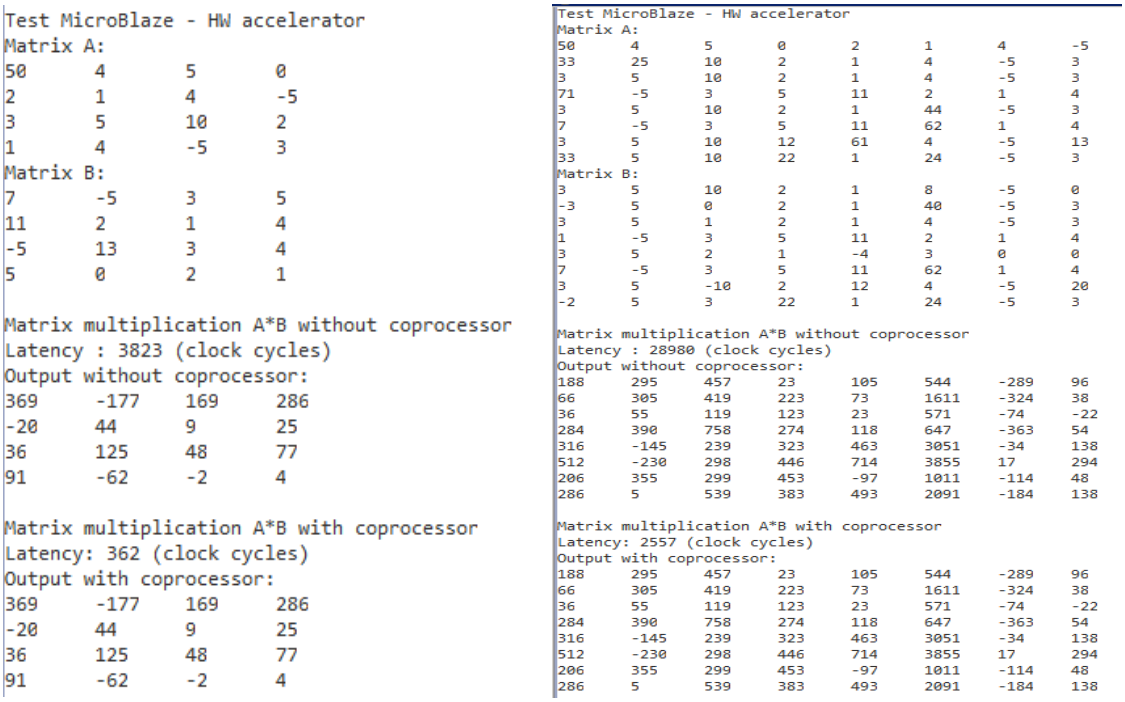


Figure 3-8: Matrix multiplication results from MicroBlaze with and without coprocessor on Kinte-7 FPGA. Latency measured with a timer attached to the AXI Lite bus. (a) 4x4 matrix multiplication. (b) 8x8 matrix multiplication.

3.2.2 Design of Matrix Multiplication Coprocessor

The fully sequential coprocessor was implemented with fixed-point and floating-point format. The fixed-point representation of a m-bit number is <m, n>, where n denotes the number of bits for the whole part. Table 3-1 shows the hardware utilization for different matrix sizes. For both number representations, this architecture requires the same number of BRAMs, but more DSPs, FFs, and LUTs for floating-point, since additional processing for format conversion is performed.

Table 3-1: Hardware utilization for floating-point and fixed-point of matrix multiplication.

	Floating-point				Fixed-point <16,1>			
Matrix Size	BRAM	DSP48E	FF	LUT	BRAM	DSP48E	FF	LUT
8x8	2 (~0%)	5 (~0%)	533 (~0%)	848 (~0%)	2 (~0%)	1 (~0%)	93 (~0%)	131 (~0%)
10x10	2 (~0%)	5 (~0%)	538 (~0%)	858 (~0%)	2 (~0%)	1 (~0%)	98 (~0%)	141 (~0%)
12x12	2 (~0%)	5 (~0%)	538 (~0%)	866 (~0%)	2 (~0%)	1 (~0%)	98 (~0%)	149 (~0%)
14x14	2 (~0%)	5 (~0%)	539 (~0%)	866 (~0%)	2 (~0%)	1 (~0%)	99 (~0%)	149 (~0%)
16x16	2 (~0%)	5 (~0%)	543 (~0%)	875 (~0%)	2 (~0%)	1 (~0%)	103 (~0%)	158 (~0%)
18x18	2 (~0%)	5 (~0%)	549 (~0%)	883 (~0%)	2 (~0%)	1 (~0%)	109 (~0%)	166 (~0%)
20x20	2 (~0%)	5 (~0%)	549 (~0%)	885 (~0%)	2 (~0%)	1 (~0%)	109 (~0%)	168 (~0%)

A hardware implementation that demands a minimum number of hardware resources is valuable because it allows the integration of more functionalities in the FPGA fabric, but it is also important to take into consideration the timing performance in order to meet the real-time requirement. This design was targeted to a clock of 100 MHz, but it can reach a maximum clock frequency of 119 MHz and 128 MHz for floating-point and fixed-point, respectively. Figure 3-9 shows the latency in term of clock cycles for different size of matrices. For this architecture, the latency of the floating point implementation is about 2.5 times higher than the fixed-point implementation.

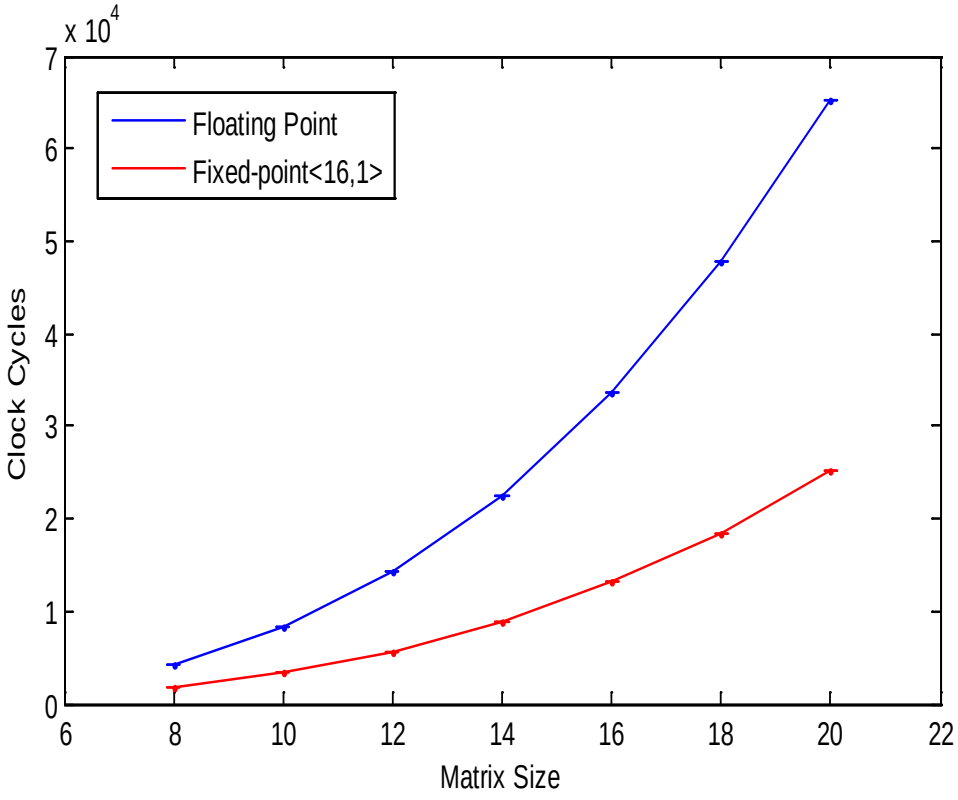


Figure 3-9: Matrix multiplication total latency for floating point and fixed-point implementation

In order to reduce the processing latency, some modifications in the design are necessary but at the same time it is important to consider that the hardware resources in a FPGA are limited. The first approach uses pipelining techniques to increase the concurrency in the execution of equation (3.3) with one clock cycle of initiation interval. The initiation interval (II) is defined as the rate at which the coprocessor can begin process a new set of data [88]. The synthesis result, presented in Table 3-2, shows that 10 DSP48Es are required for the floating-point implementation, while for fixed-point format the number of DSPs increases linearly with the size of the matrix.

And two more BRAMs are required for the floating-point implementation compare to the previous design.

Table 3-2: Hardware resource utilization for a pipelined design.

	Floating-point				Fixed-point<16,1>			
Matrix Size	BRAM	DSP48E	FF	LUT	BRAM	DSP48E	FF	LUT
8x8	4 (~0%)	10 (1%)	1544 (~0%)	2222 (1%)	2 (~0%)	8 (~0%)	155 (~0%)	216 (~0%)
10x10	4 (~0%)	10 (1%)	1779 (~0%)	2339 (1%)	2 (~0%)	10 (1%)	183 (~0%)	268 (~0%)
12x12	4 (~0%)	10 (1%)	1812 (~0%)	2481 (1%)	2 (~0%)	12 (1%)	200 (~0%)	379 (~0%)
14x14	4 (~0%)	10 (1%)	2074 (~0%)	2721 (1%)	2 (~0%)	14 (1%)	222 (~0%)	427 (~0%)
16x14	4 (~0%)	10 (1%)	1956 (~0%)	2756 (1%)	2 (~0%)	16 (1%)	248 (~0%)	430 (~0%)
18x18	4 (~0%)	10 (1%)	2124 (~0%)	2795 (1%)	2 (~0%)	18 (2%)	272 (~0%)	501 (~0%)
20x20	4 (~0%)	10 (1%)	2092 (~0%)	2984 (1%)	2 (~0%)	20 (2%)	288 (~0%)	562 (~0%)

A reduction in the latency can be achieved when the Block RAMs are replaced by distributed memories (FFs, LUTs), and pipelining equation (3.3) with an initiation interval of one clock cycle so that concurrency in the execution is increased by demanding more DSP48Es, as shown in Table 3-3.

Table 3-3: Hardware resource utilization when pipelining and distributed memory are considered in the design.

	Floating-point			Fixed-point<16,1>		
Matrix Size	DSP48E	FF	LUT	DSP48E	FF	LUT
8x8	40 (4%)	8019 (1%)	9864 (4%)	8 (0%)	2285 (~0%)	2267 (1%)
10x10	50 (5%)	11310 (2%)	13322 (6%)	10 (1%)	3497 (~0%)	3138 (1%)
12x12	60 (7%)	40908 (10%)	11920 (5%)	12 (1%)	10621 (2%)	1765 (~0%)
14x14	70 (8%)	55195 (13%)	13926 (6%)	14 (1%)	14404 (3%)	1979 (~0%)
16x16	80 (9%)	63673 (15%)	17931 (8%)	16 (1%)	18162 (4%)	3596 (1%)
18x18	90 (10%)	89407 (21%)	20237 (9%)	18 (2%)	23471 (5%)	3759 (1%)
20x20	100 (11%)	109324 (26%)	23758 (11%)	20 (2%)	28716 (7%)	4912 (2%)

A comparison of latencies for the three approaches is presented in Figure 3-10. It can be noticed that this implementation achieves lower latency compared to the other two approaches. For fixed-point and floating-point representations, the use of pipelining and distributed memories improves the latency in a linearly increasing factor compare to the pure sequential architecture. The speed up factors for floating-point and fixed-point implementations are in the range of [15, 39] and [13, 31], respectively.

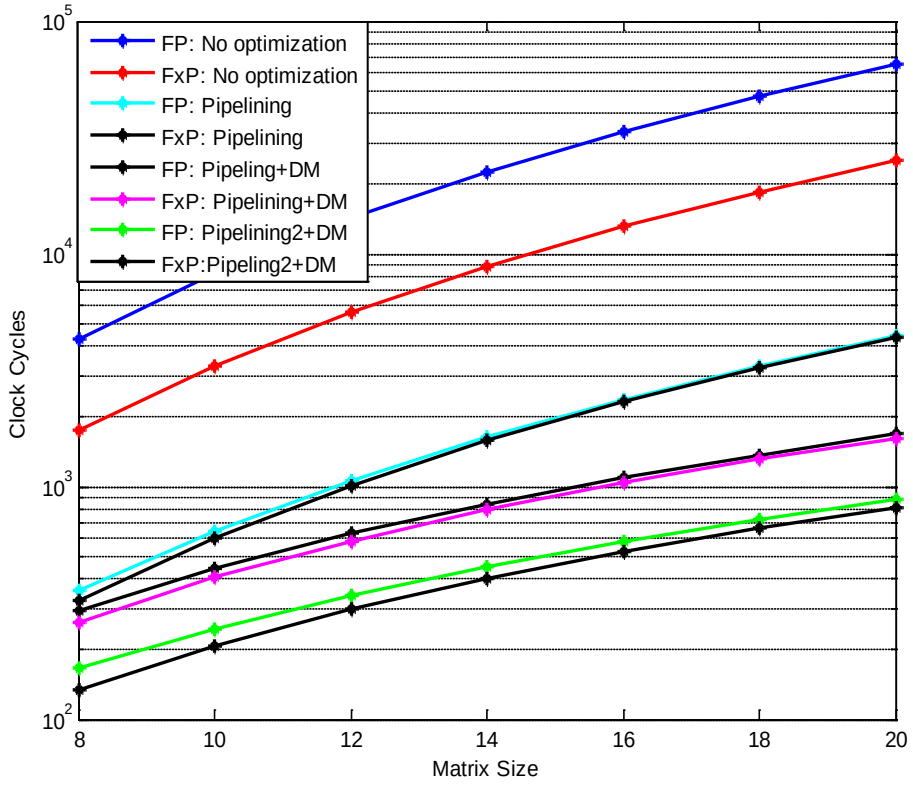


Figure 3-10: Latency in terms of clock cycles for floating point and fixed point implementation using different techniques.

3.3 Matrix Inversion

Matrix inversion is also a critical processing element in the implementation of adaptive signal processing algorithms. There is vast amount of literature discussing hardware implementation of matrix inversions. The most popular methods to calculate the inverse of a matrix are based on techniques such as the QR decomposition (orthogonal matrix (Q) and upper triangular matrix (R)), the Cholesky factorization, and the singular value decomposition (SVD).

Through extensive comparisons, the *Cholesky method* is adopted as the key approach to implement matrix inversion in this work, the computational cost of the decomposition for an n-by-n matrix is approximately $\frac{n^3}{3}$ FLOPs, plus forward and back substitution process $2n^2$ FLOPs. Similar to a matrix multiplier, the hardware implementation takes the form of a coprocessor. The high-level configuration of the coprocessor is shown in Figure 3-11.

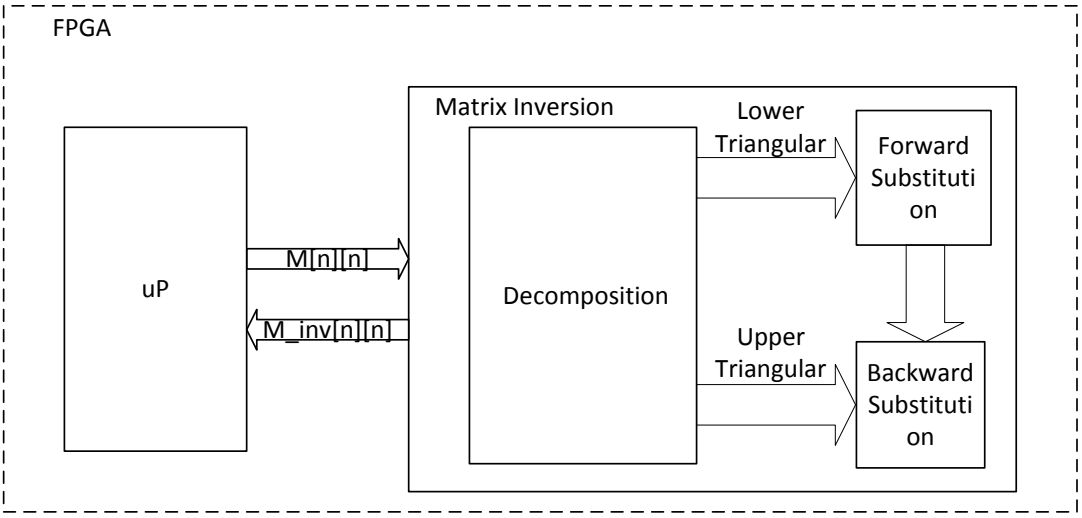


Figure 3-11: High-level matrix inversion coprocessor.

The initial implementation of matrix inversion through *Xilinx Vivado integrated design environment* is based on "non-optimized" architecture, which seeks to minimize area by reusing a small number of hardware resources to compute the matrix inversion. The hardware utilization for both fixed-point and floating-point representations is listed in Table 3-4.

Table 3-4: Hardware utilization for floating-point and fixed-point of matrix inversion.

	Floating-point				Fixed-point<16,1>			
Matrix Size	BRAM	DSP48E	FF	LUT	BRAM	DSP48E	FF	LUT
8x8	7 (~0%)	5 (~0%)	1796 (~0%)	3687 (1%)	5 (~0%)	28 (3%)	1987 (~0%)	3328 (1%)
10x10	7 (~0%)	5 (~0%)	1836 (~0%)	3763 (1%)	5 (~0%)	28 (3%)	2048 (~0%)	3406 (1%)
12x12	7 (~0%)	5 (~0%)	1696 (~0%)	3828 (1%)	5 (~0%)	28 (3%)	2088 (~0%)	3472 (1%)
14x14	7 (~0%)	5 (~0%)	1911 (~0%)	3821 (1%)	5 (~0%)	28 (3%)	2103 (~0%)	3466 (1%)
16x16	7 (~0%)	5 (~0%)	1847 (~0%)	3804 (1%)	5 (~0%)	28 (3%)	2044 (~0%)	3469 (1%)
18x18	7 (~0%)	5 (~0%)	1898 (~0%)	3878 (1%)	5 (~0%)	28 (3%)	2096 (~0%)	3545 (1%)
20x20	7 (~0%)	5 (~0%)	1891 (~0%)	3880 (1%)	5 (~0%)	28 (3%)	2089 (~0%)	3548 (1%)

The estimation of latency for single precision floating point and fixed-point <16, 1> are presented in Figure 3-12. It can be observed that for both representations, the latency increases exponentially with the number of elements in the matrix, while the floating point version sees faster increase.

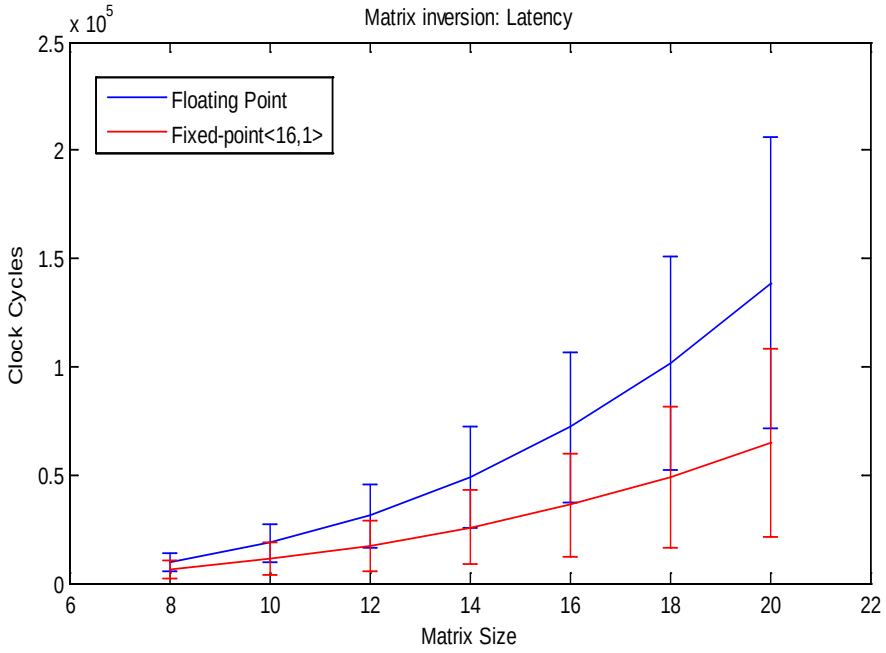


Figure 3-12: Matrix inversion latency for single precision floating-point and fixed-point <16, 1>.

The reduction of latency involves the usage of distributed memory, as well as instantiating multiple hardware resources and pipelining the computation in order to accept new input samples every 600 clock cycles. The result is an overall lower latency at the expense of using more hardware resources, as can be observed in Table 3-5 and Table 3-6.

Table 3-5: Comparison of hardware utilization for floating-point and fixed-point implementation of matrix inversion.

	Floating-point				Fixed-point			
Matrix Size	BRAM	DSP48E	FF	LUT	BRAM	DSP48E	FF	LUT
8x8	0 (~0%)	5 (~0%)	10957 (2%)	11115 (5%)	0 (0%)	736 (87%)	237444 (58%)	261095 (128%)
10x10	0 (~0%)	10 (1%)	15718 (3%)	16412 (8%)	0 (0%)	1270 (151%)	375672 (92%)	414140 (203%)
12x12	0 (~0%)	10 (1%)	22546 (5%)	22634 (11%)	0 (0%)	1992 (237%)	547720 (134%)	607940 (298%)
14x14	0 (~0%)	15 (1%)	29735 (7%)	32064 (15%)	3 (~0%)	2653 (315%)	176881 (43%)	223846 (109%)
16x16	0 (~0%)	25 (2%)	37032 (9%)	45544 (22%)	3 (~0%)	3656 (435%)	212347 (52%)	27767 (136%)
18x18	0 (~0%)	30 (3%)	46442 (11%)	60657 (29%)	3 (~0%)	4863 (578%)	250993 (61%)	33698 (165%)
20x20	3 (~0%)	40 (4%)	56020 (13%)	79859 (39%)	3 (~0%)	6290 (748%)	294530 (72%)	403065 (197%)

Table 3-6 summarizes the latency for the best (Lat_TMin) and worst case (Lat_TMax) for floating-point as well as fixed-point versions. The range of initialization interval (II-T) is also summarized. We can observe that Lat_TMin and Lat_TMax are equal for cases in which the synthesis process achieved the specific initiation interval value (600 clock cycles). In the other cases, the synthesis tool optimized as much as it could to achieve the targeted initiation interval, but from Table 3-6 it can be observed that for a matrix size larger than 12×12, using fixed-point representation is not able to be fully optimized with the Xilinx synthesis tool, resulting in a higher latency compared with the latency of the floating-point implementation.

Table 3-6: Comparison of timing results for floating-point and fixed-point implementation of matrix inversion.

	Floating-point				Fixed-point<16,1>			
Matrix Size	Lat_T Min	Lat_T Max	II_T Min	II_T Max	Lat_T Min	Lat_T Max	II_T Min	II_T Max
8x8	511	511	512	512	822	822	600	600
10x10	557	557	558	360	1105	1105	600	600
12x12	868	868	600	600	1378	1378	600	600
14x14	988	988	600	600	1266	21986	1267	21987
16x16	1037	1037	600	600	1270	30338	1571	30339
18x18	1256	1256	600	600	1906	40426	1907	40427
20x20	1416	1416	600	600	2274	52394	2275	52395

Chapter 4

FPGA implementation of Pulse Compression

4.1 Hardware Implementation of Pulse Compression

In this chapter, a unified digital pulse compression processor is presented as a radar-application-specific-processor (RASP) architecture for the next generation of adaptive radar. Based on traditional pulse compression matched filter and correlation receiver, the processor integrates specific designs to handle waveform diversities, which includes both frequency modulation and randomized waveforms, as well as digital transceiver *self-reconfiguration* for adaptive radars. The prototype of this processor is implemented with Xilinx FPGA devices and tested with an RF spaceborne radar transceiver testbed developed at the University of Oklahoma's Radar Innovations Laboratory (RIL). Validation results show the effectiveness of real-time processing and the engineering concepts.

4.1.1 FPGA in Existing SDR platforms

As mentioned in Chapters 1 and 2, there are strong potentials for the FPGA implementation of real-time pulse compressions. FPGA has been used extensively not only in the traditional HPEC systems but also in newly-emerged commercial software-

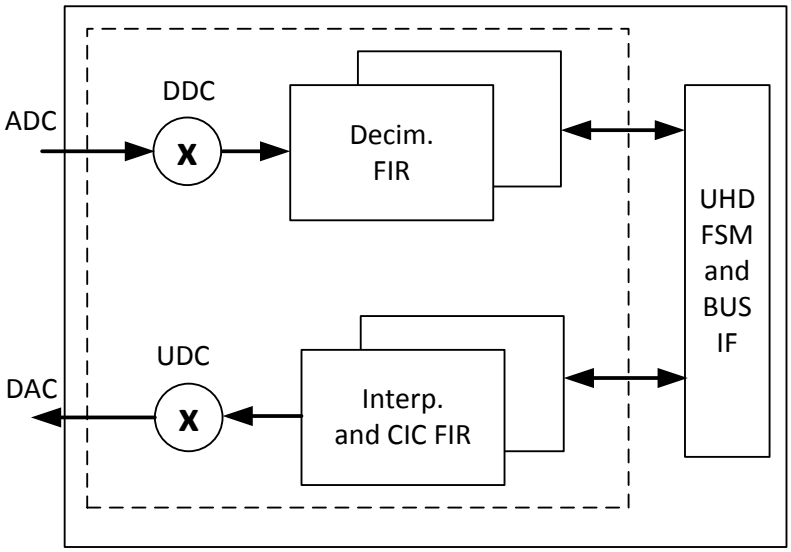
defined radio/radar (SDR) platforms [89, 90]. The reconfigurable capability of FPGA naturally enables the SDR platforms. For example, the Universal Software Radio Peripheral (USRP) platform from Ettus Research [91] has been using both Altera and Xilinx FPGAs in the digital transceiver physical layer. Various FPGA devices have been used in other current SDR platforms [92, 93].

4.1.2 Radar TR Control Layer

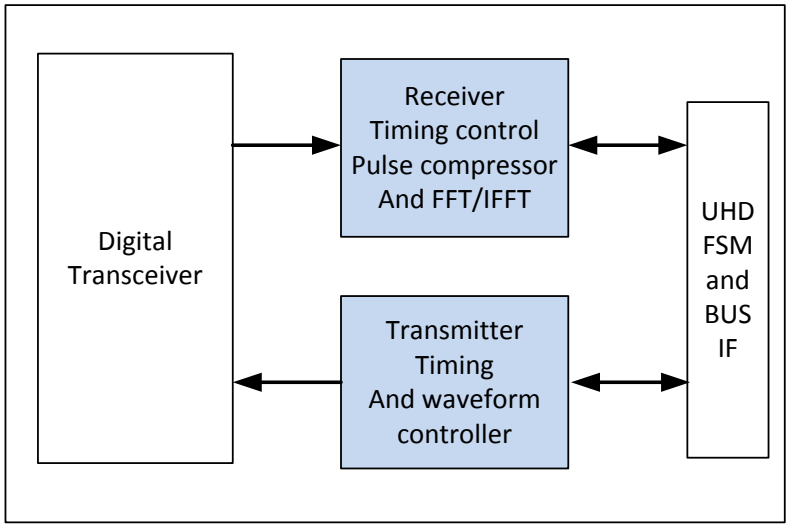
Although there have been some attempts to use commercial SDRs for radar applications [94, 95], the success of these efforts is largely limited by the real-time capability of these platforms. Figure 4-1(a) shows an existing USRP-FPGA implementation, which includes the DDC, UDC, FIR filters, numerical-controlled oscillator (NCO), and PC interface control. These functional blocks just ensure the basic transceiver functions, but the radar processing functions will have to be implemented in software (for this particular example, it is GNU Radio and USRP hardware driver (UHD)). For ground-based radar with low-computational requirements such as weather radars, software-based radar processor implementations have been popular [96, 97], which also use real-time Linux and Graphic Processor Unit (GPU) acceleration when it is possible.

As discussed in previous chapters, for low SWaP (Space, Weight and Power) radar applications and the reconfigurable platforms, PC-based processors do not meet the requirements. A novel aspect of the proposed FPGA implementation scheme is the combination of the hardware-based radar processing functions and SDR architectures, especially for the GNU and USRP type of radio system platforms. The “core” radar

processor hardware includes the TR controller, pulse compressor, and spectrum analyzer. These core blocks are most desired for real-time processors and are used in cognitive radios, etc. The concept of this extension is illustrated in Figure 4-1(b).



(a)



(b)

Figure 4-1: (a) Existing FPGA configuration of N210/E110 from Ettus Research. (b) Proposed FPGA configuration for Radar transceiver (with enhanced radar transceiver Real-time range-Doppler processing blocks).

4.2 Architecture Design and Analysis for Real-Time Pulse Compression Circuitry

Pulse compression can be implemented as a time-domain correlator (as in noise radar's correlation receiver), or a frequency-domain matched filter, which is implemented with FFT/IFFT as shown in Figure 4-2.

The output of the matched filter $y(n)$ using typical time-domain correlation can be expressed as:

$$y(n) = \sum_{m=0}^{M-1} x_r(m)x(n-m) = x_r(m) * x(n) \tag{4.1}$$

In equation (4.1) x_r represents the received signal, and x is the radar waveform template. It is known that the convolution of two signals in the time-domain is equivalent to the multiplication of the signals in the frequency domain. The pulse compression can, therefore, be implemented by converting the received signal and the transmitted signal (matched filter function) to the frequency domain using FFTs. Once they are in the frequency domain, a vector multiplication of them can be performed, followed by an inverse Fourier transform to convert the result back into time domain.

As stated in Chapter 2, frequency-domain matched filter implementation is much more efficient than time-domain correlator for pulse compression. For instance, an N-tap filter in the time-domain requires N complex multipliers for each output sample. And when assuming N-point FFT transform and frequency-domain template in memory, $2\log(N)+1$ complex multipliers per output samples are needed. In

addition, the FFT transform is a power of two point size (N), which needs to satisfy the following conditions:

$$N = 2^k \geq p + q - 1 \quad and \quad q \geq 2B\tau \tag{4.2}$$

In the above equation k is a positive integer, p is the number of samples of the incoming signal, and q is the number of samples of the reference signal. Note that in Figure 4-2, the reference waveform template can also be pre-calculated and stored in the internal memory of the FPGA, which as we will point out, may not be the best option when the waveform bandwidth is large, and it also introduces the possibility of mismatching.

Figure 4-2 shows the high-level architecture of pulse compression implementation, which incorporates different schemes for matched filter implementation. This architecture includes an optional *pre-processing block* which translates the incoming IF signal to baseband, eliminates undesired frequencies, and reduces the sampling rate. The circuit then temporally store the samples in first in, first out (FIFO) buffers, which are activated by an external trigger and controlled by a counter, which controls the writing and reading operations. The N-point FFT is applied to the buffered samples and then multiplied by a reference pulse spectrum, which can be obtained from three different sources: (a) Pre-processing the waveform template using the same input channel, thus the hardware resources can be saved; (b) Real-time samples collected from a dedicated input channel while the transceiver is operating (this scheme demands more hardware resources); (c) Pre-calculated spectrum coefficients stored in the on-chip memory. A weighing function block is required when using scheme (a) or (b). The compressed time-domain pulse is obtained

by applying an N-point inverse Fourier transform (IFFT) to the result of the complex multiplier. The samples of the compressed pulse may then be sent to the DAC for displaying purposes. Depending on the availability of hardware resources and device capabilities, further processing can be performed in the same device. Otherwise, samples may be sent to another processing unit through high-speed links such as serial RapidIO (SRIO).

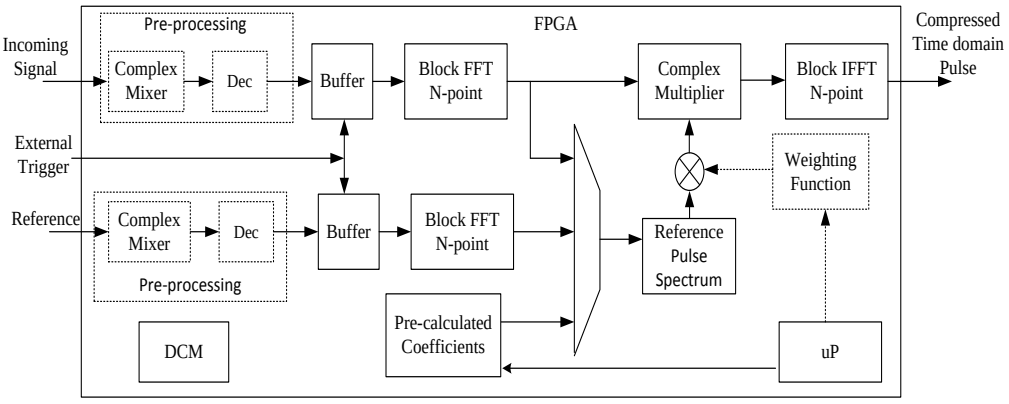
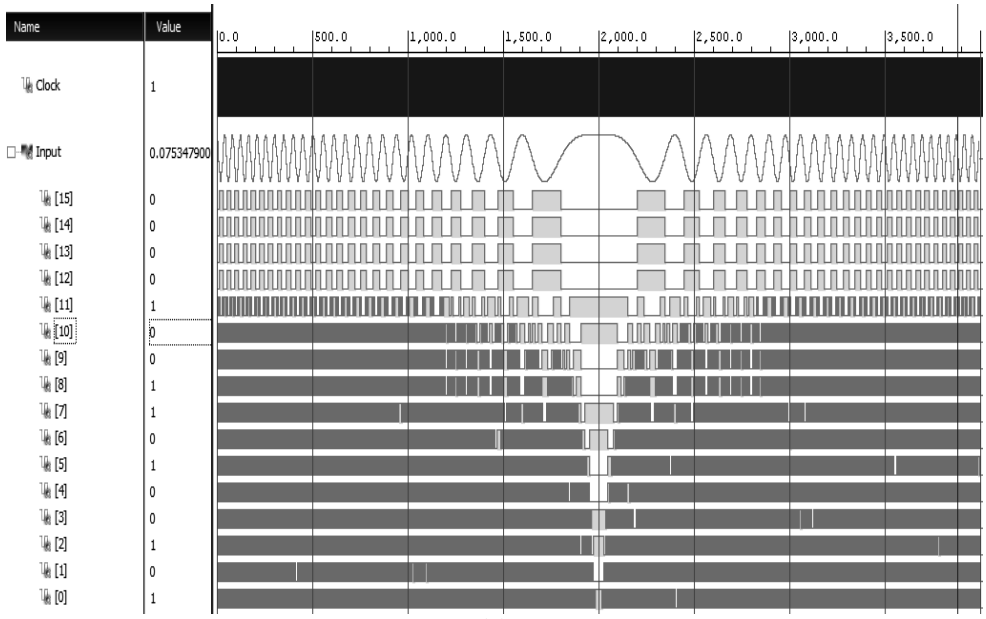


Figure 4-2: High-level block diagram for matched-filter pulse compression implementation.

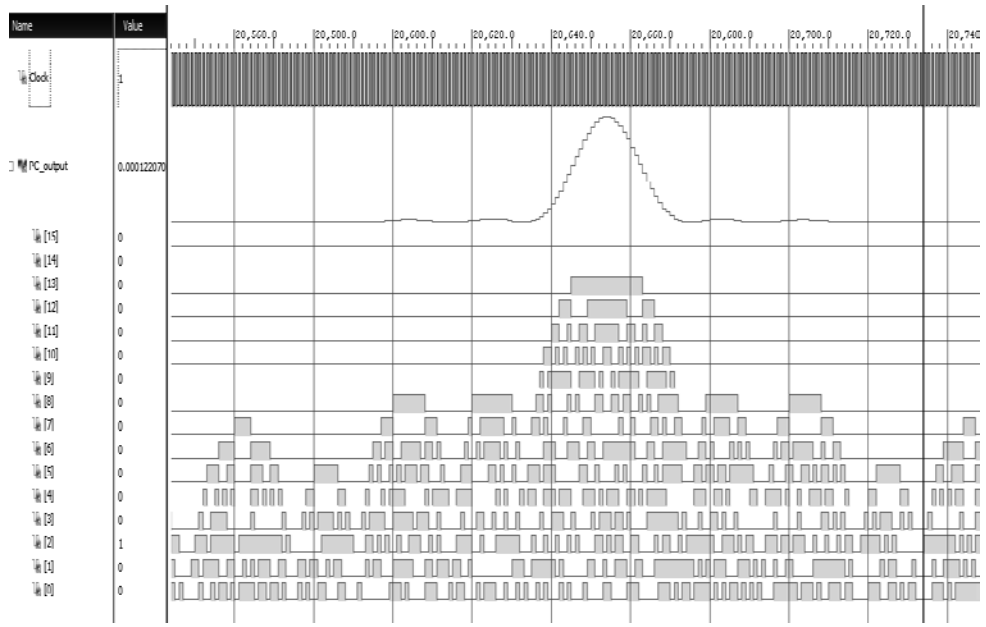
The FFT processing blocks are based on fixed-point operations and configured as radix-2 butterfly stages with distributed memories, in which the stages are pipelined so that data can be continuously streamed. Before a hardware bitstream is generated, fixed-point simulations are performed at different levels.

Using Xilinx's software development tools, the target device for hardware simulations was a Kintex-7 FPGA (XC7k325t-2-ffg990), in which the incoming signal was an LFM waveform with BW = 5 MHz and τ = 20 µs. The results of hardware

simulation are shown in Figure 4-3, in which the uncompressed input signal and the compressed output signal are represented with 16 bits.



(a)



(b)

Figure 4-3: Hardware simulation of pulse compression, using 16-bit digital representation. (a) Uncompressed input signal. (b) Compressed output signal.

Additionally, three different weighting functions: Kaiser ($\beta = 2.23$), Hamming, and Hanning were considered. The comparison between hardware and MATLAB simulations is shown in Figure 4-4. The uncompressed pulse was a chirp signal with a bandwidth of 10 MHz and a pulse duration of 20 μs. The results show that hardware simulations achieve similar sidelobe levels as the sidelobe levels from theoretical predictions. As expected, weighting functions reduce the range sidelobe levels at expenses of range resolution degradation.

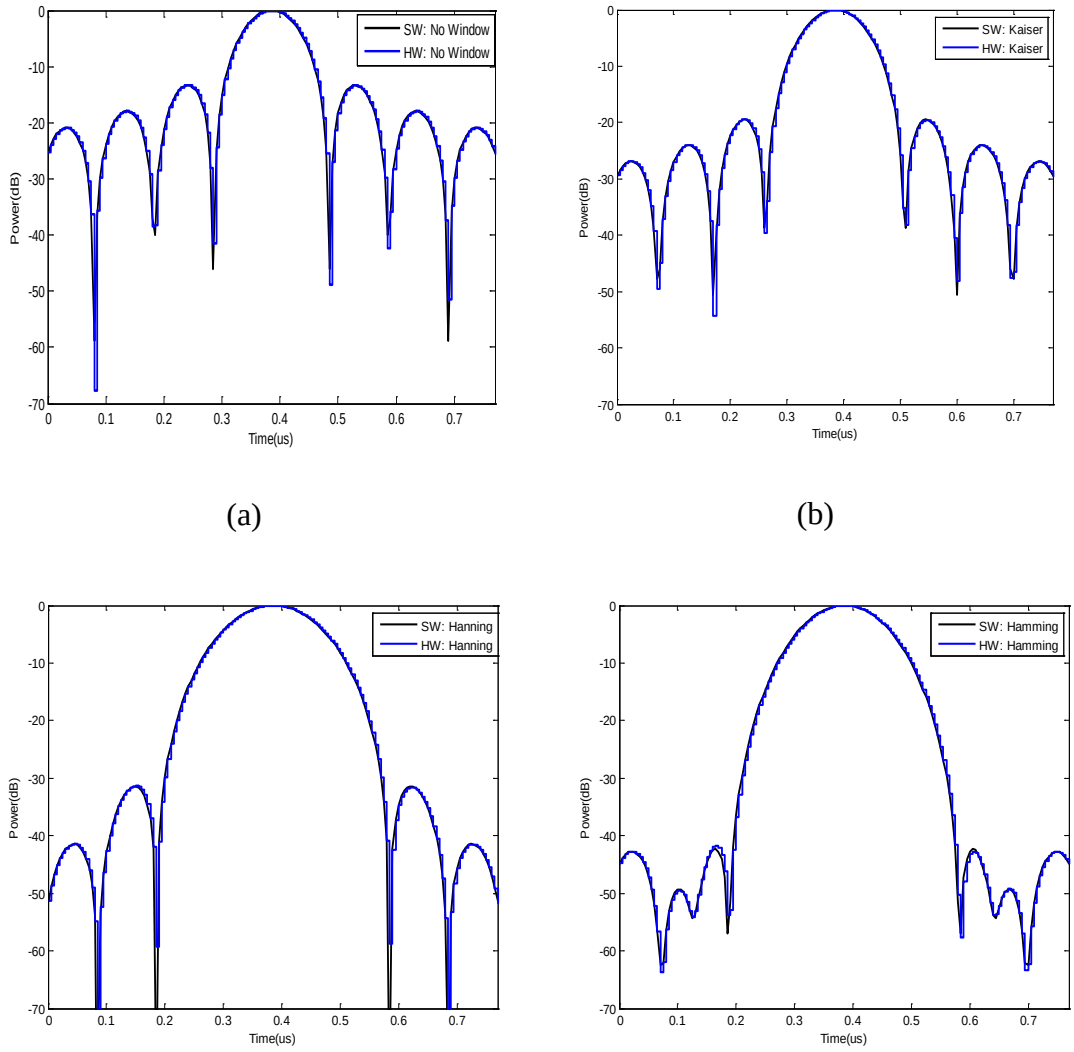


Figure 4-4: Comparison between MATLAB and hardware (Kintex-7 FPGA) simulations of pulse compression for different weighing windows. (a) No window. (b) Kaiser ($\beta = 2.23$). (c) Hanning. (d) Hamming.

The results also show a slight mismatch due to quantization errors, and scale factors applied in the different processing stages, as presented in Figure 4-5. The peak

range sidelobe of the hardware simulations when using no window is -13.28 dB, Kaiser (β = 2.23) is -19.38 dB, Hanning is -31, 32 dB, and Hamming is -41.64 dB.

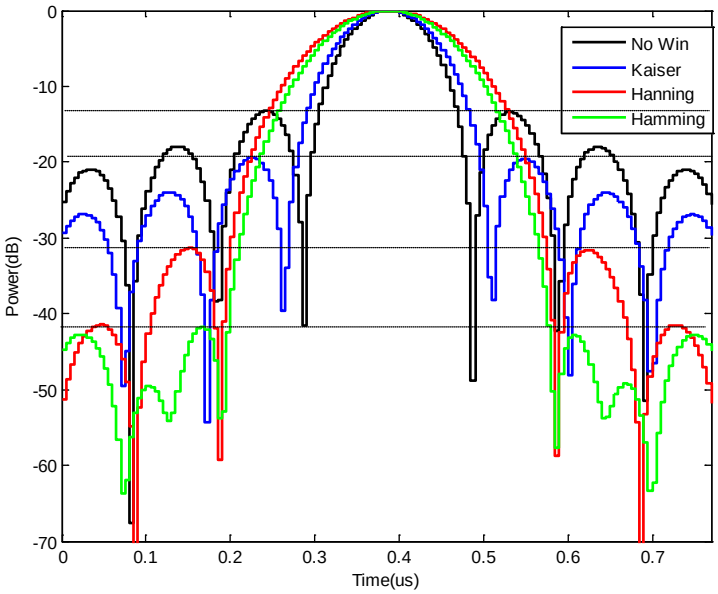


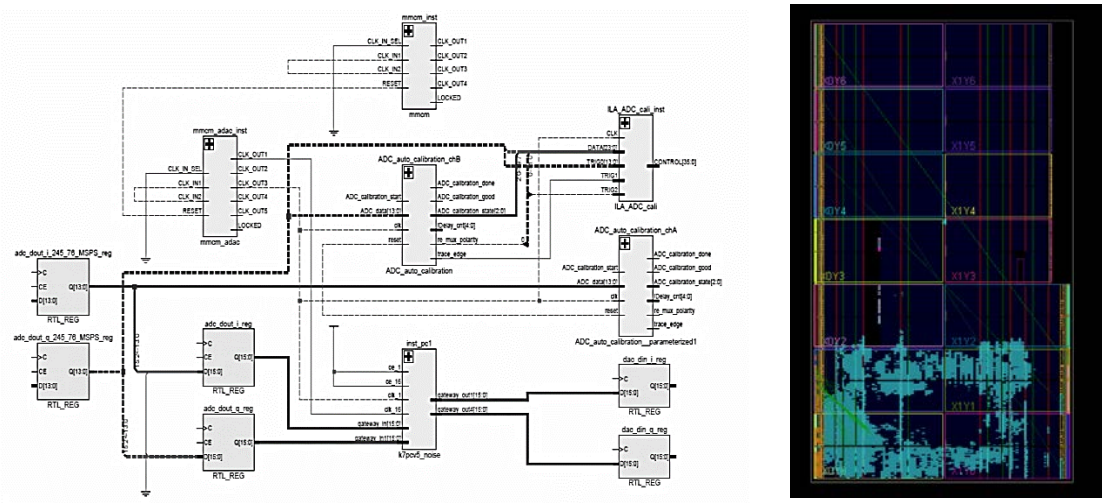
Figure 4-5: Comparison of pulse compression hardware simulation results using different windows: Kaiser (β = 2.23), Hanning, and Hamming. The simulation target is a Kintex-7 FPGA.

4.3 FPGA Device Implementations of Real-Time Pulse Compression

4.3.1 Hardware Resource Utilization

The target device for this implementation was the XC7k325t-2-ffg990 FPGA. The platform was attached to a 14-bit dual-channel ADC (ADS62P49) and a 16-bit dual-channel DAC (DAC3283) FMC daughter board, which includes an external

trigger port. The sampling rate was 246 MSPS, which was configured from the FPGA through the Serial Peripheral Interface (SPI). The input samples were formatted to two's complement 16-bit representation. Figure 4-6 shows typical on-chip implementation result including the simplified RTL diagram and the resulting layout. The pulse compression system clock runs at 246 MHz.



(a) (b)

Figure 4-6: Examples of on-chip implementation results. (a) Simplified Vivado RTL schematic for pulse compression. (b) The resulting layout of pulse compression implementation (light blue area) on the XC7k325t-2-ffg990 FPGA.

The post-implementation resource utilization, in terms of FFs, LUTs, BRAMs and DSPs, is summarized in Table 4-1. This table only considers the pulse compression processing block with 8192-point FFT and IFFT, FIFO buffers, and pre-calculated complex coefficients for the reference pulse spectrum. It can be observed that BRAMs, (17% of the total available for that device is used) are the most

demanding hardware resource for this implementation, and FFT/IFFT operations use the majority of them. Other architectures for FFTs may be considered to reduce the number of hardware resources at expenses of data throughput.

Table 4-1: Device Resource Utilization for two Xilinx FPGAs for the typical matched filter implementation.

Operations\HW Resources	LUTs	FFs	BRAMs	DSPs
FFT	4407	6876	23	25
IFFT	6962	10901	27	42
Complex Multiplier	236	523	0	12
Others: counters, add/subs, relational, registers, etc.	73	231	22	2
Total	11678 (6%)	18531 (5%)	72 (17%)	81 (10%)

The total on-chip power consumption for this implementation was 1.838 W, from which the dynamic power and static power were 1.659 W and 0.179 W, respectively. The pulse compressor block only consumes 0.783 W, which represents ~43% of the total power. The power consumption in each processing block is detailed in Table 4-2.

Table 4-2: Power consumption of pulse compression

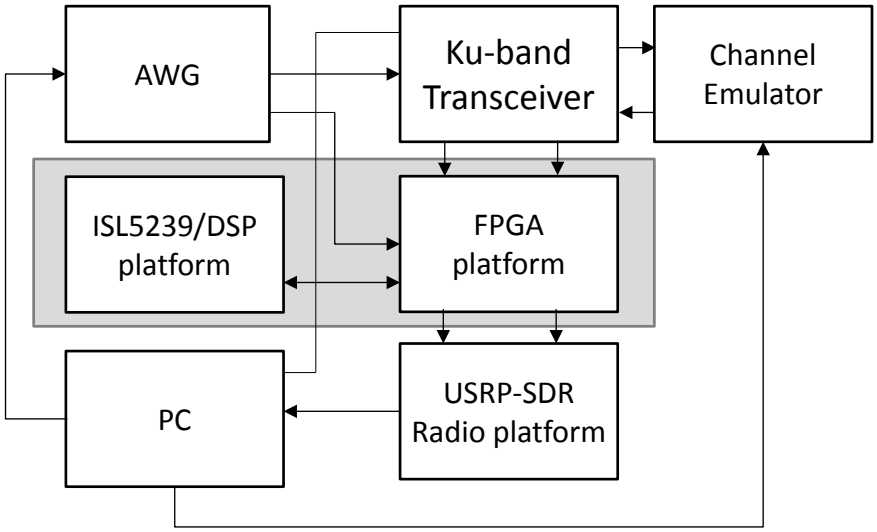
Operations	Power (W)
FFT	0.295 (16%)
IFFT	0.38 (21%)
Complex Multiplier	0.037 (2%)
Others (counters, add/subs, relational, registers, etc.)	0.071 (4%)
Total Power of DPC	0.783 (43%)

4.3.2 Test and Validation Platforms

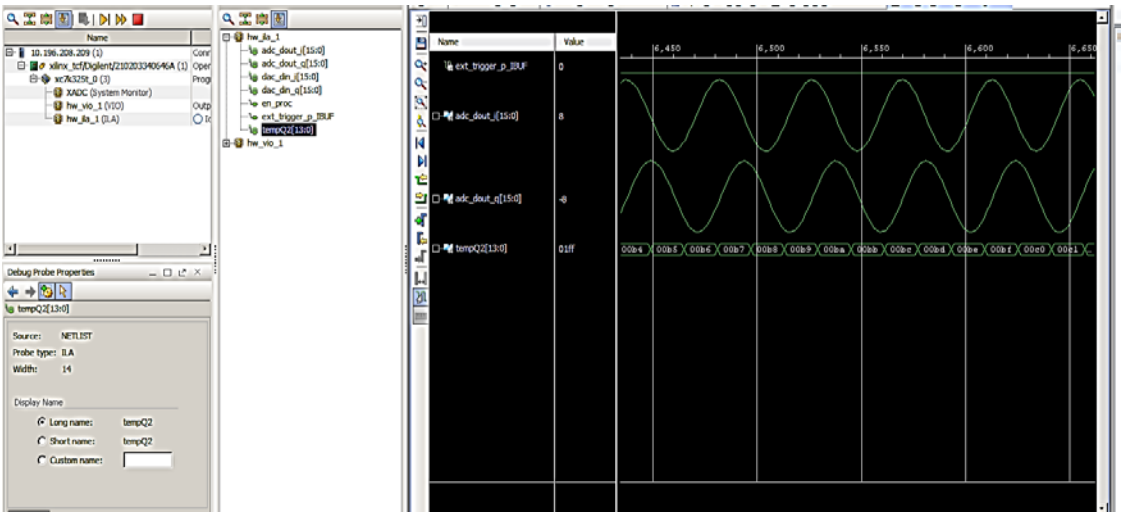
The pulse compression processor implementation can be tested and validated through different methods. For hardware-level tests, two options were used as shown in Figure 4-7. The first option is a complete hardware testbed, in which the FPGA platform, an actual RF transceiver, DSP platform, and PC are all connected as a real-time radar platform. The compressed pulse is sent out of the FPGA platform through the DAC, and is measured as a short analog pulse. This signal is then acquired by the digital storage oscilloscope (DSO) for further verifications.

Another method for hardware verification is using a software-based logic analyzer. For this implementation, *Vivado Logic Analyzer*, which includes virtual I/O (VIO) and integrated logic analyzer (ILA) IP cores, was inserted into the design in order to collect a number of bits from the FPGA through the JTAG interface and displays signals and waveforms, which allows debugging during each step of the

processing with respect to the driving clock. This method is also used for test and verification.



(a)



(b)

Figure 4-7: Methods of hardware verification. (a) Complete hardware testbed, (b) Using Vivado logic analyzer for probing internal signals.

4.4 Experiment Results

4.4.1 System Outputs for Basic PC Waveform

Pulse compression processing results of an up-chirp LFM signal, with a pulse duration of 20 μs and bandwidth of 10 MHz, are shown in Figure 4-8. The target return, in this case, is a simple duplication of the transmit pulse aligned with the waveform template. This figure shows the external trigger with a pulse duration of 500 ns, the in-phase (I) and quadrature-phase (Q) input signals, and the output signal captured by ILA. A slight distortion can be observed in the low-frequency domain of the IQ signals, which is caused by the DC filter in the ADC. For displaying purposes, the compressed pulse is shown in linear scale based on the addition of the power of two of the real and imaginary components: $\mathrm{Im}^2 + \mathrm{Re}^2$.

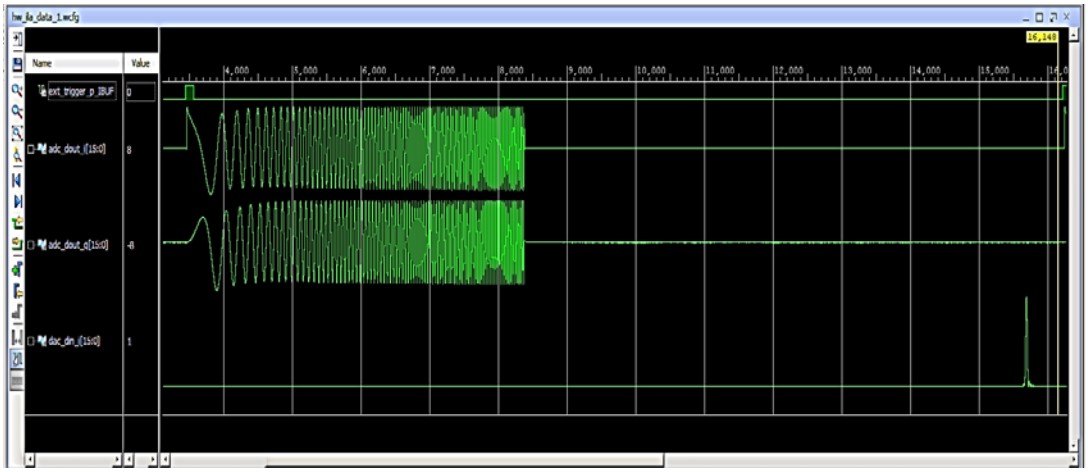


Figure 4-8: Pulse compression results captured using Xilinx's integrated logic analyzer (ILA). External trigger with pulse duration of 500 ns, I and Q with pulse duration of 20 μs and bandwidth of 10 MHz.

Figure 4-9 shows the resulting compressed pulse after converting the samples of the compressed pulse to logarithmic scale. As is observed, the peak sidelobe level is similar to the HW and MATLAB simulations (around -13.3dB).

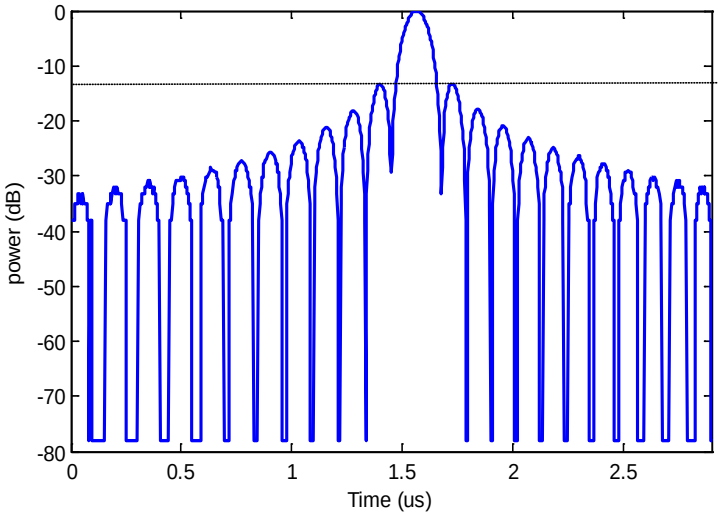
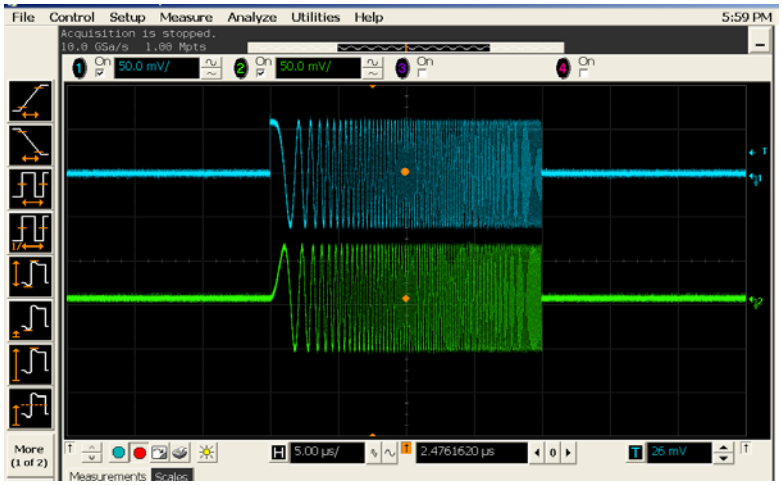
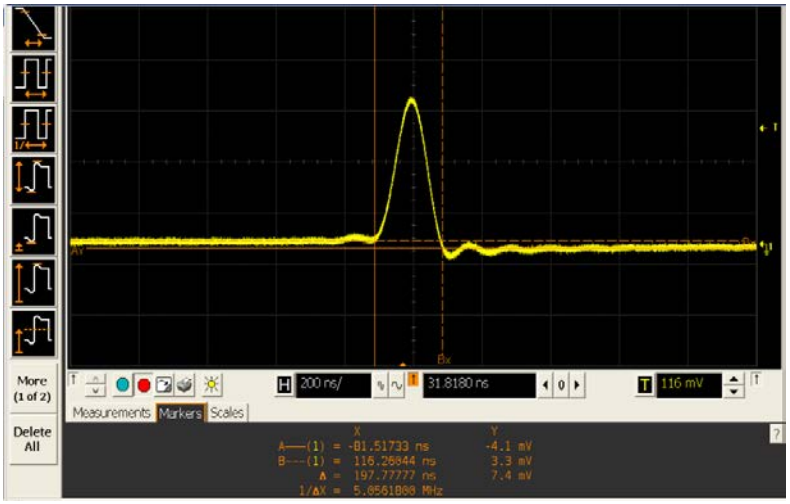


Figure 4-9: ILA samples of pulse compression output converted to logarithmic scale (dB).

The pulse compression result in digital form is then converted to an analog pulse output through the DAC with a resolution of 16 bits, which is captured by the Digital Storage Oscilloscope (DSO) and shown in Figure 4-10. The compressed analog pulse is identical to the digital result in Figure 4-9, and the overall processing latency is about 28.8 μs, which justifies the real-time processing. On the other hand, it can be noticed in Figure 4-10 that some distortion was introduced by the DAC during the digital-to-analog conversion process.



(a)



(b)

Figure 4-10: Comparison between uncompressed time domain input ((a), pulse bandwidth = 10 MHz, pulse length = 20 µs), and compressed time domain output pulse ((b), captured by DSO).

Figure 4-11 includes the analog amplitude outputs from the pulse compression processor, when there are two emulated targets adjacent to each other and using a

short pulse of 2 μs. The two targets are approximately separated by 300 meters and are assumed to be identical point targets.

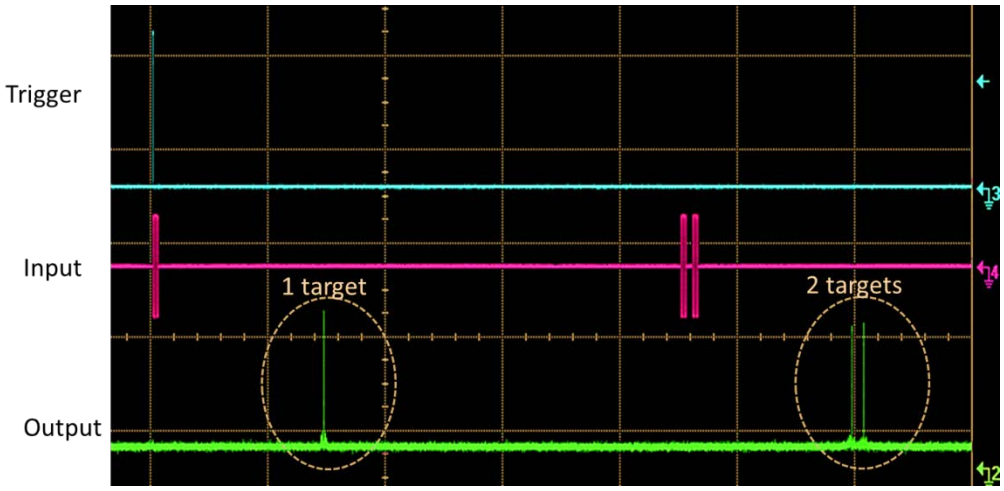


Figure 4-11: Pulse compressor output for multiple emulated targets. Captured by DSO.

4.4.2 Real-Time Pulse Compression for Random Waveform

Compared to the chirp/LFM waveform, random waveforms are usually designed to use wider bandwidth. The FPGA implementation of pulse compression for random waveforms has a limitation on the instantaneous signal bandwidth it can handle, which originates from the limitation of ADC sampling rate and clock frequency. For XC7k325t platform implementation, the largest allowable signal bandwidth is about 100 MHz. The FPGA itself, on the other hand, can process signals with larger bandwidth as long as the computation load fits in the device capacity.

Figure 4-12 (a) shows a sample of a *random-noise* waveform with 40 MHz modulation bandwidth and about 20 μs pulse length. The significant difference between this waveform and the normal chirp waveform is the large *amplitude dynamic*

range, so the ADC/DAC distortions will have impacts on the output. In Figure 4-12 (b), the digital pulse compressor results (captured using Vivado logic analyzer) show good overall sidelobe performance. For the FPGA system's analog output, which is shown in Figure 4-12 (c), the impact of the distortion of the DAC and RF channel can be clearly observed.

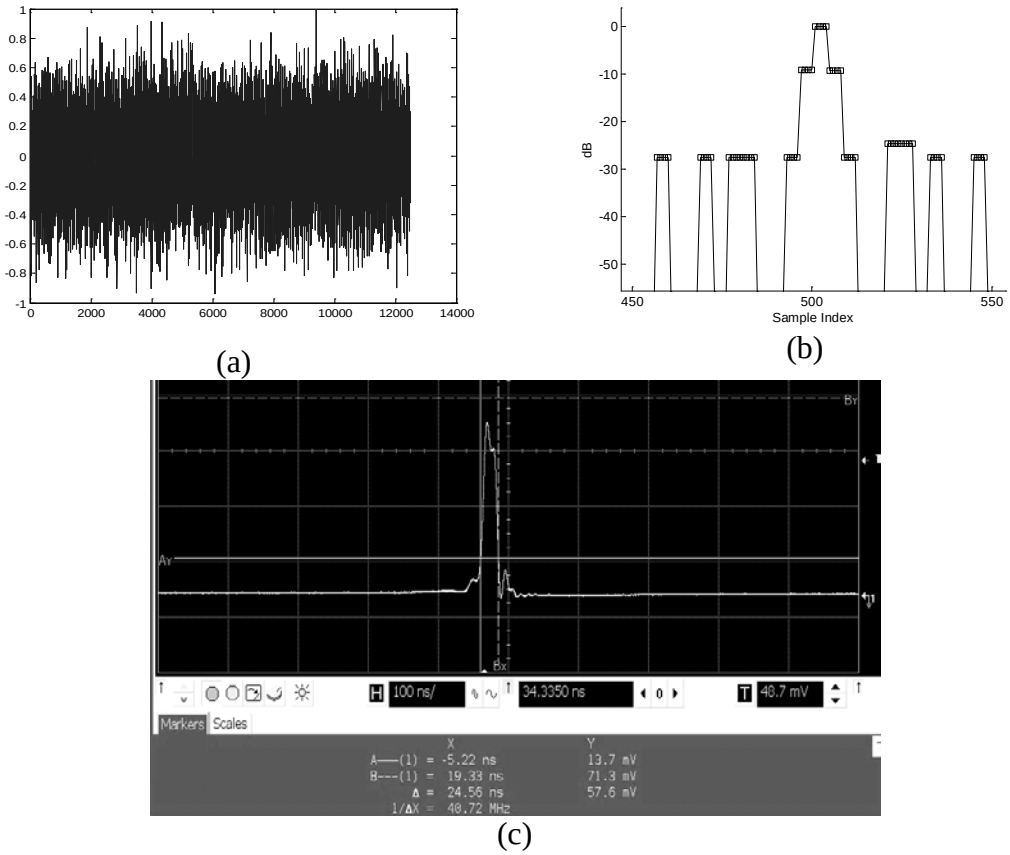
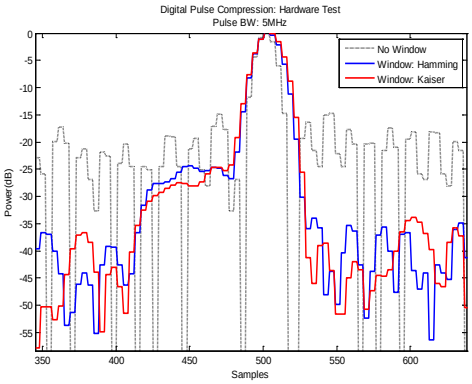


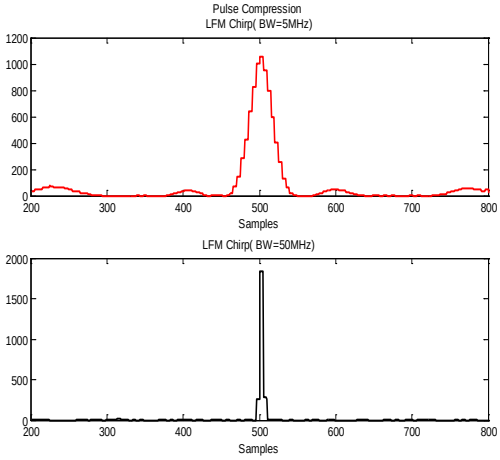
Figure 4-12: Real-time pulse compression of band-limited random noise with the FPGA pulse compression implementation, (a) Input waveform (40 MHz signal bandwidth), (b) Pulse compressor output captured using Vivado logic analyzer (before DAC output), (c) Pulse compressor output converted to analog pulse and captured by DSO.

4.4.3 Impact of Waveform Template Generation Scheme and Timing Misalignment

A waveform template (or local replica) is needed for pulse compression receivers. This template can be either generated locally (within chip) assuming the waveform parameters are all known, or can be sampled from a transmitter coupled/loop back signal. Sometimes the precise transmit waveform is not available and can only be estimated from other ways. This may happen when a pulse compression processor needs to be added into an existing operational radar. Obviously, there is potential mismatching between the received actual signals and the local-generated waveform templates, which can cause degradation in PC results, especially on sidelobes. For hardware implementation, another issue with internally generated templates is the timing misalignment between the two signals. One example is shown in Figure 4-13 (a), where the waveform parameters in terms of bandwidth and pulse length are known, but due to timing (phase) misalignment, the mismatching still causes strong sidelobes (dotted line). Again, adding windowing can mitigate this effect at the cost of resolution.



(a)



(b)

Figure 4-13: Comparison between the PC outputs using internal waveform template generation (without external waveform samples) and PC outputs with external waveform templates and different bandwidths.

When an external source of the waveform template is available, and it is "synchronized" with the received pulse, the FPGA output and the sidelobe are much more stable. Figure 4-13 (b) shows compressed output with 5 MHz BW (same as (a)) chirp waveform and 50 MHz BW chirp waveform when a waveform template is acquired from the external waveform generator (the same as the source to the RF transceiver).

4.5 Conclusions

This chapter presents results about real-time FPGA implementation of a software-defined radar signal processor. The core of this processor is the real-time pulse compression processor, realized as a fixed-point matched filter. As a software-defined IP core, the pulse compression can be easily reconfigured to process different

waveforms on different devices or platforms. The two examples – narrowband chirp waveform for solid-state weather radar, as well as wideband random noise waveform processing, are presented. The capability of processing the noise waveform is largely limited by the ADC speed and DAC dynamic range, while the FPGA devices on the market have enabled the real-time pulse compression and radar controllers for wideband or even ultra-wideband waveforms. It is found that conversions between analog and digital signals can cause distortions in the pulse compression result, and some of these distortions are not necessarily deleterious. Also, generation of waveform templates with adequate timing-alignment is important to the real-time performance.

Chapter 5

SoC Implementation of an Adaptive Radar Processor

The proposed adaptive radar processor is part of the solid-state radar *transceiver optimizer* which is composed of different processing blocks, such as adaptive pulse compressor (APC), pre-distorter, general-purpose processor, coprocessors, and peripherals. The goal of designing, testing and investigation of these building blocks is to integrate them into a single chip and to achieve the lowest C-SWaP, and support airborne and spaceborne platform deployments.

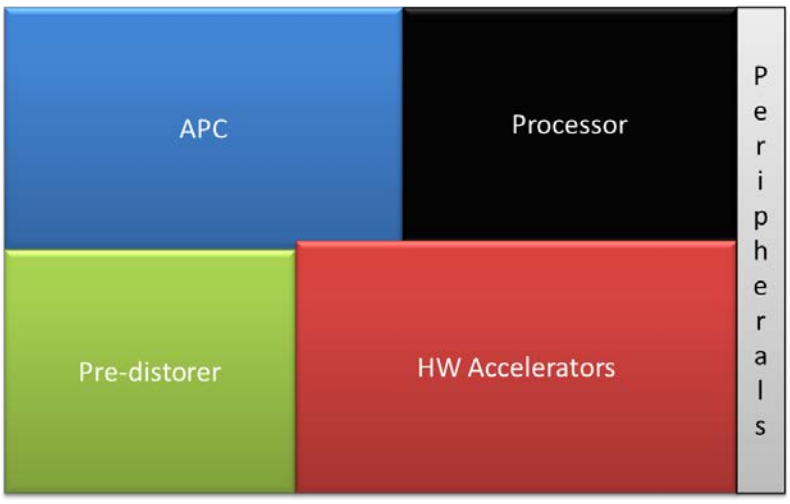


Figure 5-1: System elements of the proposed radar transceiver optimizer.

The overall on-chip system requires one or more processors to execute the software, control program, data transfer, and information processing. The

interconnection architecture of the transceiver optimizer processor, based on Xilinx SoC technology, is shown in Figure 5-2. The modularity of this architecture makes it possible to be applied to different FPGA devices or radar platforms.

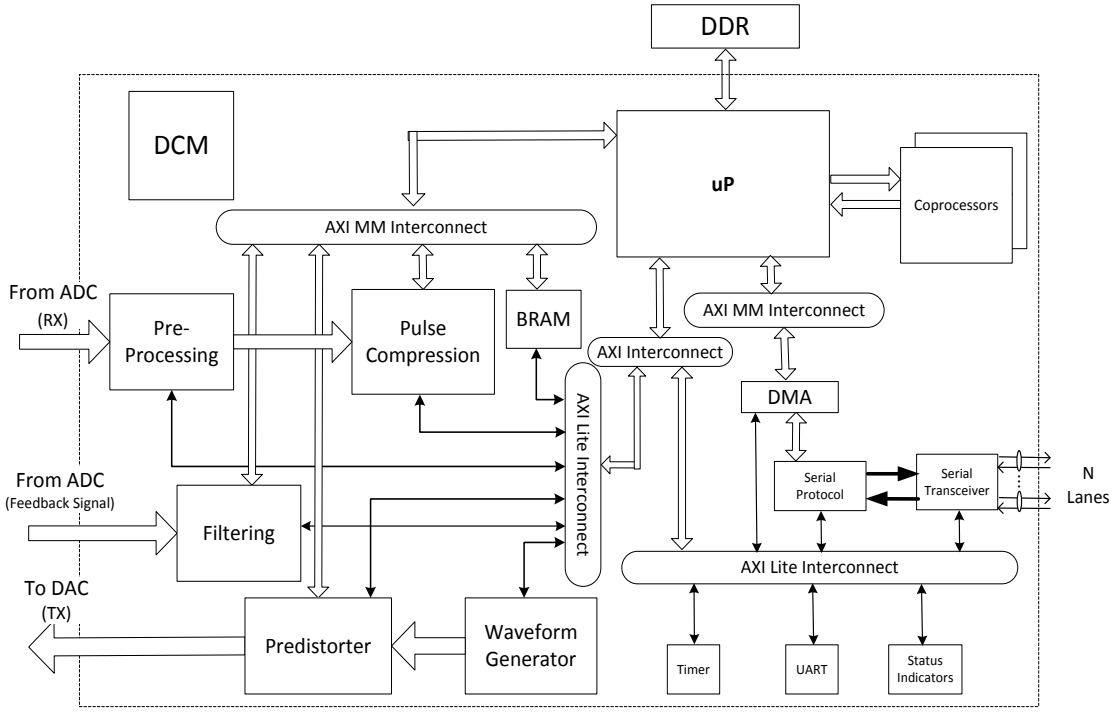


Figure 5-2: Transceiver optimizer System-on-Chip (SoC).

The system collects samples from the ADC, and streams data into the FPGA where filtering, decimation, and direct pulse compression are performed first. Using AXI Interconnection, the compressed pulse samples are then sent to an on-chip microprocessor, which computes the *adaptive pulse compression* and it is supported by custom coprocessors designed to accelerate the computation process. The APC output results can be transferred to another external device through high-speed serial interconnections with multiple lanes.

The transceiver optimizer also performs *digital baseband pre-distortion*, for which the transmitted feedback signal is pre-processed, and then stored in a specific area of the shared memory. These samples along with the pre-distorter output are read by the main processor through the AXI interconnect buses, in order to compute and update the coefficients of the pre-distorter. The primary focus on this chapter is SoC implementation of the APC building block.

5.1 Literature Review: Implementation of Traditional Adaptive Filters

The most common adaptive algorithms used to calculate the filter weights are based on the least-mean-squares (LMS) and the recursive least-squares (RLS). They have been used in different applications [98-109]. It is also known that RLS offers faster convergence over LMS, since it is independent of the spread of the eigenvalues of the correlation matrix. These algorithms and their variants are extensively covered in [110-113].

In general, real-time implementation of adaptive algorithms is limited by different factors such as complexity, accuracy, numerical stability, dynamic range, etc. The low complexity and regular structure of the LMS algorithm make it suitable for being implemented on hardware. For instance, two variants of the Delayed LMS [114] algorithm were implemented in [107]: the DF-DLMS(direct form) and TF-DLMS (transposed form) predictors. The targeted device was a Virtex XCV300-6 FPGA, achieving about 10 times speedup compared to the traditional LMS. The pipelined implementation of TF-DLMS demanded a huge area compare to DF-LMS

and LMS. For 8-bit and 8 taps, the achieved clock frequency was 12 MHz for traditional LMS, 68 MHz for TF-LMS, and 120 MHz for DF-LMS. In [115], a software implementation of D-LMS in a DSP core (SPXK5) is described, achieving a speed of 1 cycle per tap.

On the other hand, the RLS algorithm suffers from the computation of the correlation matrix and its inverse. In [116], RLS with fixed-point format was implemented for a 4-element MMSE adaptive array antenna.

The two principal methods to reduce the computational complexity are based on the application of the matrix inversion lemma or the QR-decomposition recursive least-squares (QRD-RLS) technique. The QRD-RLS approach has been studied for several decades, as well as its performance in different applications [117-127]. In [128], the implementation of matrix inversion using the QR technique, fixed-point, was implemented on a Spartan 3 FPGA XC3S1000, achieving a maximum clock speed of 13.6 MHz, using 25% of the hardware resources, and the maximum matrix size dimension was 23x23 performed in 253 μs. [124] presents a hierarchical architecture of QRD-RLS for digital beamforming, in which techniques such as look-ahead, pipelining and folding were applied to increase the throughput, reduce area, and power consumption.

Numerical format analysis of RLS is described in [125, 129, 130]. For instance, [130] proposed a derived Kalman algorithm and compared with traditional RLS, their performance was analyzed for fixed-point and floating-point representation, in which it also used a modified floating-point format to take advantage of the 18-bit multipliers in the Virtex-5 FPGA family.

A variation of this QR-RLS is called *systolic array QR-RLS* [131, 132], which is also known as *Givens rotation* or *Coordinate Rotate Digital Computer* (CORDIC)-based RLS algorithms. In [133], the authors described an architecture of VLSI systolic array for an adaptive nulling processor based on the CORDIC algorithm in a systolic architecture, in which the function of a CORDIC cell is to update the *Cholesky* factor of the correlation matrix every 22 ms. It was estimated that 96 CORDIC cells can be capable of updating a 64-element weight vector for 300 observations in 6.7 ms. [100] presents 16-bit QR decomposition for 4x4 matrices, achieving 10x and 100x speedup compared to an Intel i7 processor (3.6 GHz) and ARM Cortex A-9 (533 MHz). Another method to improve the RLS algorithm is based on dichotomous coordinate descendent iterations (RLS-DCD), which was implemented on a Xilinx Virtex-II Pro XC2VP30 in [134]. The results shows that the transversal RLS-DCD can update a 64-tap adaptive filter at a rate of 74 KHz, occupying an area of 1306 slices. Other adaptive methods such as the Levenberg-Marquardt algorithm to solve non-linear LS, was implemented on a Virtex-5 FPGA in [102], obtaining an execution time of 60 μs operating with a clock frequency of 200 MHz.

5.2 System-on-Chip (SoC) Implementation of APC

As described in Chapter 2, adaptive pulse compression is a series of radar signal processing algorithms that are independent of waveforms and achieve an optimal estimation of ground-truth. These algorithms have been shown to work for both point-target and distributed-target scenarios. In this chapter, hardware implementation of the basic APC algorithms, LSE and RMMSE, for real-time

transceiver optimization is presented. The same architecture as well as design guidelines can be extended naturally to other APC algorithms. An important parameter to evaluate the design is the latency, which is the number of clock cycles that a processing unit takes to generate the outputs from corresponding inputs. Also, since hardware resources are limited in an FPGA, the estimation of the silicon area demanded by a design is also important. This is expressed in terms of DSP48E units (DSPs), flip-flops (FFs), and look-up tables (LUTs).

5.3 LS-APC Multi-Coprocessor Architecture

The output of the LS algorithm is given by equation (2.10). It can be seen that matrix multiplication and matrix inversion are the two main operations for the LS algorithm. In order to accelerate the computation of these two matrix operations, two independent coprocessors are considered in this first architecture, they communicate with the host processor via dedicated AXI buses.

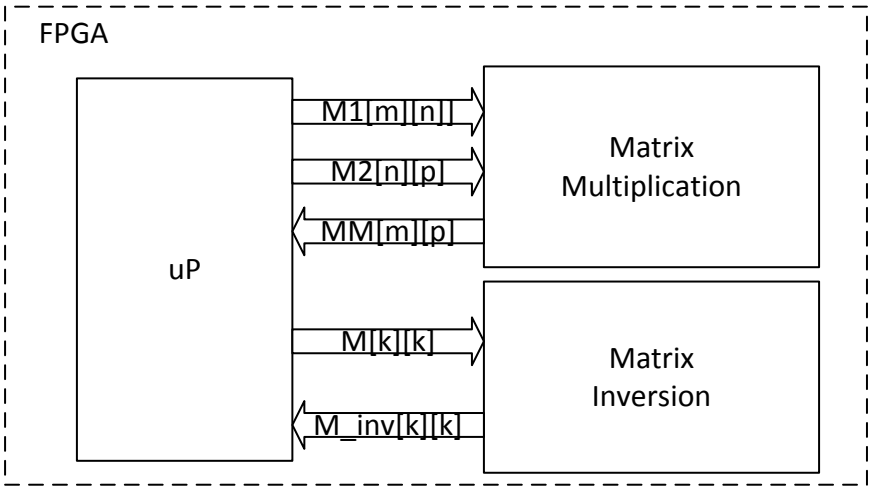


Figure 5-3: Multiple co-processor for LS-APC.

As shown in Figure 5-3, the two coprocessors were independently studied and tested in the previous chapter. The host processor initializes the process, and passes data to the multiplication coprocessor to calculate the $S^H S$ term; results are then sent back to the main processor to execute the matrix inversion in the second coprocessor. Since the matrix coprocessor considers square matrices, the output from the matrix inversion will need to be filled with zeros to form an $L+N-1$ x $L+N-1$ matrix, a similar modification is performed for the S^H matrix. Both reshaped matrices are sent to the first coprocessor for multiplication to obtain the LS filter weights which will be applied to the incoming signal y. As it can be noticed, when the three processors are connected, the total execution time is determined not only by the coprocessors' processing latency, but also by the time needed to execute the processor instructions (fetch, decode, execute), transfer data from memory to coprocessors and vice versa. For example, when the MicroBlaze processor is configured without caches, the average latency for transferring a value from memory to AXI Stream port is about six clock cycles (CPU), and reading from the port and storing in memory takes about five clock cycles (CPU), which can limit the performance of this architecture. A traditional method to reduce data transfer latency is to use a dedicated unit called direct-access memory (DMA). The DMS unit can efficiently perform *burst transfers*.

A summary of the total hardware resources of the matrix multiplication and matrix inversion coprocessors, for a sequential architecture with minimum silicon area occupancy, is shown in Table 5-1.

Table 5-1: Total hardware resources for the matrix multiplication and matrix inversion.

	Floating Point			
Size	BRAM	DSP48E	FF	LUT
8x8	9 (1%)	10 (1%)	2329 (~0%)	4535 (2%)
10x10	9 (1%)	10 (1%)	2374 (~0%)	4621 (2%)
12x12	9 (1%)	10 (1%)	2234 (~0%)	4694 (2%)
14x14	9 (1%)	10 (1%)	2450 (~0%)	4687 (2%)
16x16	9 (1%)	10 (1%)	2390 (~0%)	4679 (2%)
18x18	9 (1%)	10 (1%)	2447 (~0%)	4761 (2%)
20x20	9 (1%)	10 (1%)	2440 (~0%)	4765 (2%)

Replacing the BRAMs for *distributed memory* and pipelining the hardware architecture in each coprocessor, the combined latency is reduced by a factor that varies linearly with the number of matrix elements, as shown in Figure 5-4. The speed up factor for the considered matrix sizes in the figure is in the range of 18 to 66.

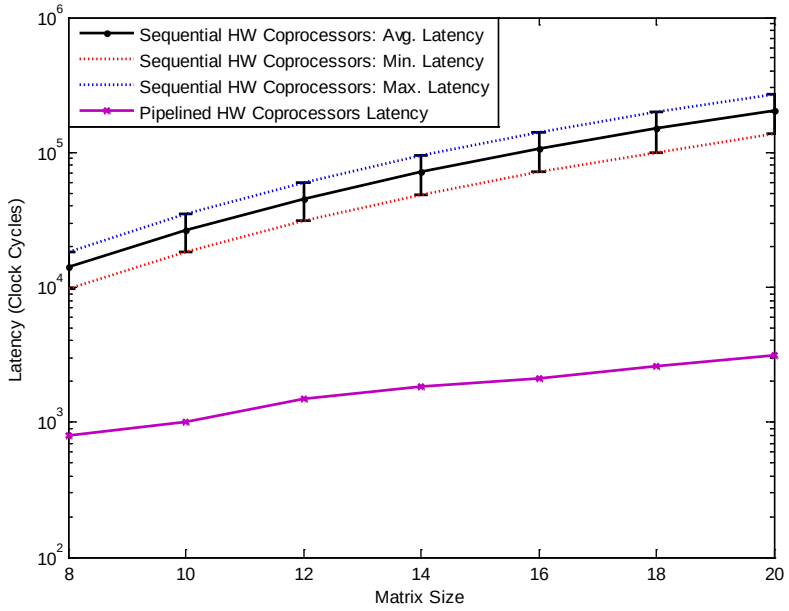


Figure 5-4: Combined latency of matrix inversion and matrix multiplication coprocessors for the sequential and pipelined versions.

However, the number of DSP48Es, FFs and LUTs is much more than the sequential architecture. Table 5-2 summarizes the total hardware resources for the implementation of the two coprocessors with lower latency.

Table 5-2: Total hardware resources for pipelined version of matrix multiplication and matrix inversion.

Size	DSP48E	FF	LUT
8x8	45 (5%)	18976 (4%)	20979 (10%)
10x10	60 (7%)	27028 (6%)	29734 (14%)
12x12	70 (8%)	63454 (15%)	34554 (16%)
14x14	85 (10%)	84930 (20%)	45990 (22%)
16x16	105 (12%)	100705 (24%)	63475 (31%)
18x18	120 (14%)	135849 (33%)	80894 (39%)
20x20	140 (16%)	165344 (40%)	103617 (50%)

5.4 Single LS-APC Processor

The second type of implementation utilizes a single LS processor, which communicates with the main processor through the AXI stream buses. The samples from the transmitted waveform *s* and the received signal *y* are buffered first, and then the LS algorithm is applied to estimate the output vector, x_LS . The architecture performs the processing in a sequential manner, as is shown in Figure 5-5. The performance of both fixed-point and floating-point implementations has been studied. The system clock speed for this design was targeted at 100 MHz.

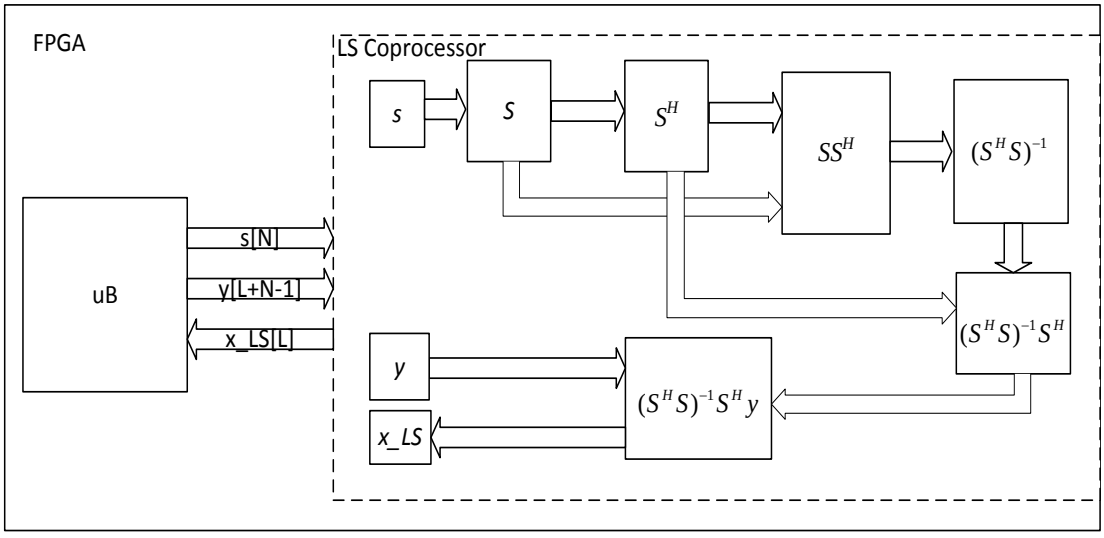


Figure 5-5: Internal architecture of the single LS coprocessor option.

The summary of hardware utilization for 16-bit fixed-point single co-processor implementations is shown in Table 5-3. The implementation requires 13 DSP48Es which represents about the 1% of the total available for XC7k325t device. The estimated dynamic power consumption, considering only the coprocessor, is between 95 mW and 133 mW for the number of range gates listed in the table.

Table 5-3: Hardware utilization of LS fixed-point implementation using 16-bit fixed-point format for Xilinx XC7k325t FPGA.

Gates	BRAMs	DSP48Es	FFs	LUTs
10	8 (~0%)	13 (1%)	1905 (~0%)	3508 (1%)
20	8 (~0%)	13 (1%)	2040 (~0%)	3808 (1%)
30	13 (1%)	13 (1%)	2087 (~0%)	3881 (1%)
40	18 (2%)	13 (1%)	2077 (~0%)	3969 (1%)
50	34 (3%)	13 (1%)	2431 (~0%)	4402 (2%)

When the number of bits is increased to 32, the implementation requires slightly more logic resources, and four times more DSPs than the 16-bit implementation. The dynamic power consumption of the coprocessor is between 213 mW and 261 mW.

Table 5-4: Hardware utilization of LS fixed-point implementation using 32-bit fixed-point format for Xilinx XC7k325t FPGA.

Gates	BRAMs	DSP48Es	FFs	LUTs
10	8 (~0%)	56 (6%)	3709 (~0%)	5935 (2%)
20	10 (1%)	56 (6%)	3706 (~0%)	6085 (2%)
30	21 (2%)	52 (6%)	3513 (~0%)	5780 (2%)
40	35 (3%)	56 (6%)	3753 (~0%)	6184 (3%)
50	67 (7%)	56 (6%)	4107 (1%)	6590 (3%)

The number of clock cycles required by the LS coprocessor to produce an output (latency) versus different number of signal samples is shown in Figure 5-6. For this experiment, the number of range gates was maintained constant as 60. It can be

observed that the latency varies linearly with respect to the number of waveform signal samples.

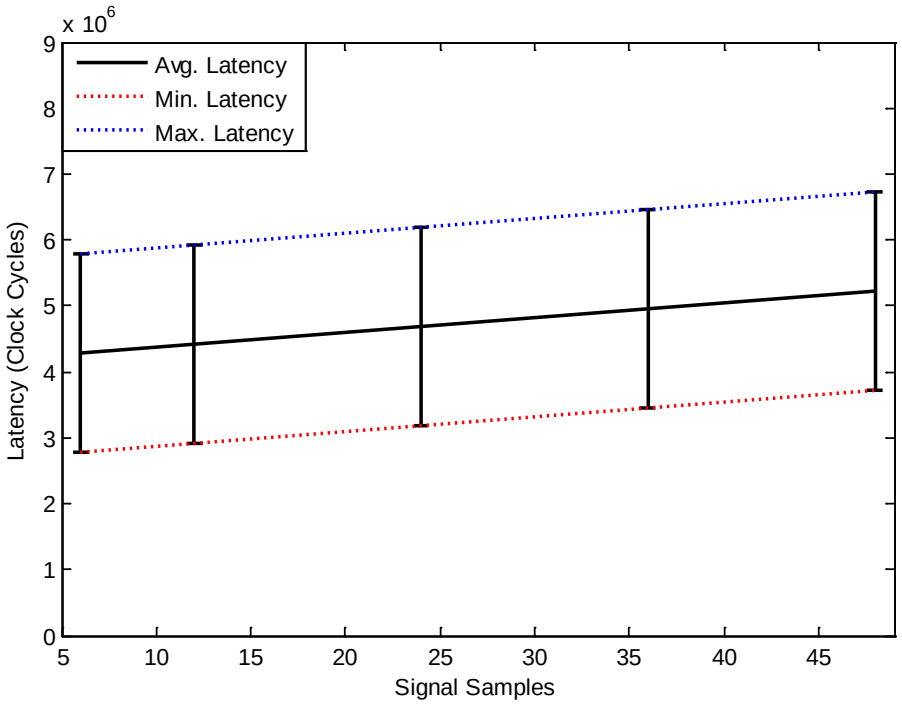


Figure 5-6: Estimated latency of LS coprocessor for different number of signal samples with a constant number of range gates. The bar plot also shows the range of variation (max and min) of latency estimation. Number of range gates = 60.

5.5 LS implementation based on Floating-Point Data Format

The same co-processor architecture in Figure 5-5 is also implemented using floating-point data formatting, wherein we used single precision floating-point format containing 32 bits: one sign bit, eight exponent bits, and 23 fractional bits. In this initial approach, the coprocessor performs in a fully sequential mode the estimation of

the ground truth. Table 5-5 summarizes the hardware resource utilization of the floating point co-processor implementation. The dynamic power is in the range of 127 mW to 141 mW.

Table 5-5: FPGA resource utilization for floating-point implementation

Gates	BRAMs	DSP48Es	FFs	LUTs
8	8 (~0%)	10 (~1%)	2753 (~0%)	5126 (~2%)
10	8 (~0%)	10 (~1%)	2765 (~0%)	5198 (~2%)
12	8 (~0%)	10 (~1%)	2841 (~0%)	5308 (~2%)
14	8 (~0%)	10 (~1%)	2910 (~0%)	5369 (~2%)
16	8 (~0%)	10 (~1%)	2853 (~0%)	5363 (~2%)
18	8 (~0%)	10 (~1%)	2901 (~0%)	5426 (~2%)
20	8 (~0%)	10 (~1%)	2894 (~0%)	5441 (~2%)
22	11 (~0%)	10 (~1%)	3190 (~0%)	5784 (~2%)
24	16 (~0%)	10 (~1%)	2966 (~0%)	5586 (~2%)

The impact of number of range gates on the latencies of floating-point LS implementation is shown in Figure 5-7. The maximum clock frequency for this implementation is 125 MHz.

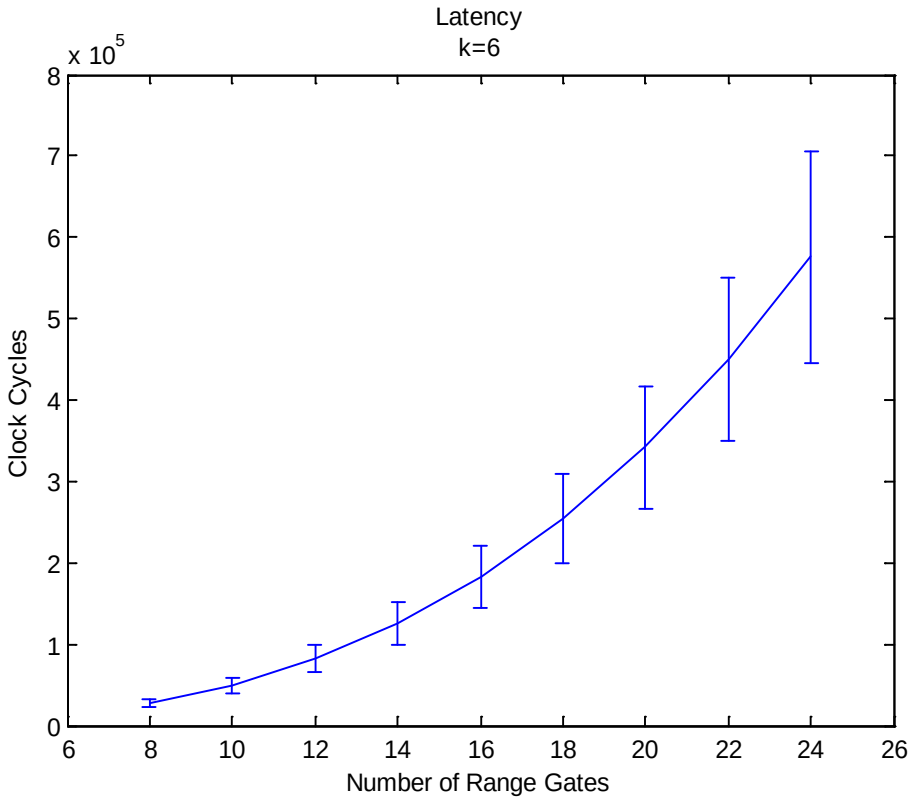


Figure 5-7: Estimated latencies for different number of range gates for floating point implementation, assuming the number of transmitted signal samples is 6 (a short pulse).

The latency for fixed-point and floating-point architectures is compared in Figure 5-8. The fixed-point with 16-bit configuration results in a better performance (smaller latency) compared to the 32-bit fixed-point implementation. As expected, it also shows all fixed point implementations have lower latency than that of the floating-point implementation. It can be also seen again that the latency of the floating-point architecture increases more rapidly compared to the other two architectures.

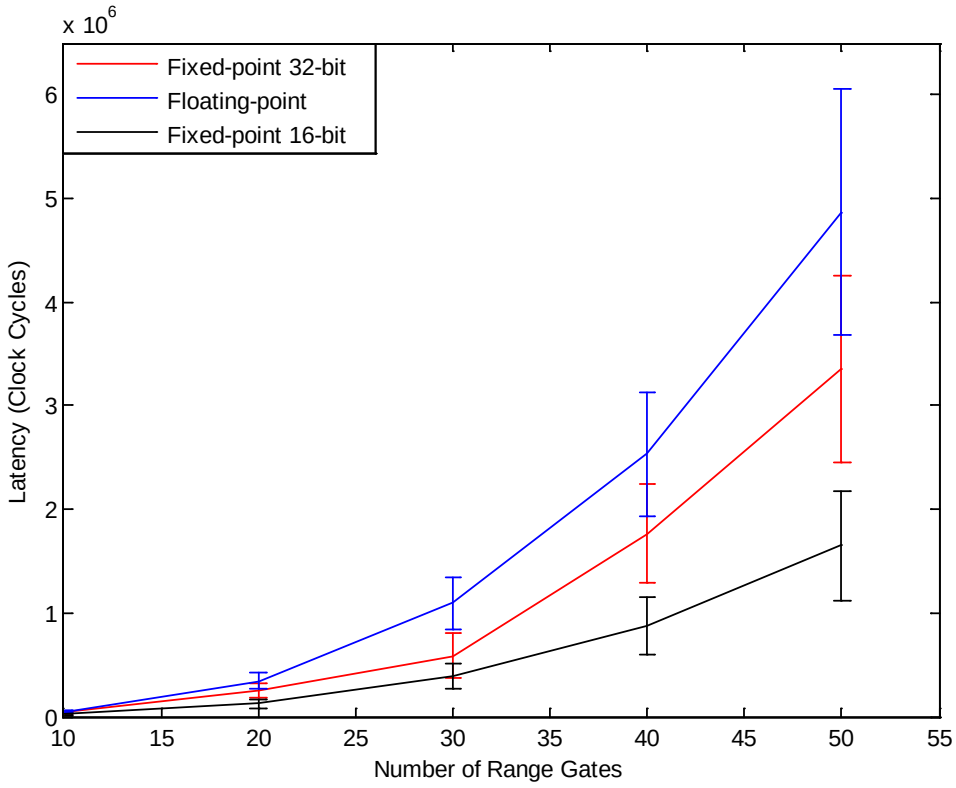


Figure 5-8: Performance comparison between fixed-point and floating-point implementation for different number of range gates. Comparison of Latency Between Fixed-point and Floating Point Implementation

Signal Samples = 6.

As we have seen in the previous results, the latency of LS implementations is usually large and also varies in certain ranges. Implementations of the single co-processor LS gate-level architecture are then studied. The methodology includes *pipelining* and *parallelizing* the design, and replacing BRAMs with distributed memory in order to reduce latency and improve throughput. As a result from these improvements, the first architecture is able to accept a new set of data (initiation interval) every 100 clock cycles. The hardware utilization is shown in Table 5-6.

Table 5-6: FPGA hardware resource utilization for pipelined floating point implementation.

Gates	DSP	FF	LUT
8	15 (1%)	8058 (1%)	6858 (3%)
10	25 (2%)	10654 (2%)	10418 (5%)
12	33 (3%)	17314 (4%)	14312 (7%)
14	40 (4%)	21800 (5%)	18440 (9%)
16	50 (5%)	27983 (6%)	23696 (11%)
18	63 (7%)	36310 (8%)	31360 (15%)
20	73 (8%)	45917 (11%)	39112 (19%)
22	88 (10%)	56486 (13 %)	46986 (23%)

Now, assuming that a new set of data is received every 50 clock cycles, the throughput of the coprocessor then needs to be increased by about two times. This improvement will results in at least 1.5 times more hardware resource utilization, as shown in Table 5-7.

Table 5-7: FPGA hardware resource utilization for initiation interval of 50 clock cycles

Gates	DSP	FF	LUT
8	30 (3%)	10850 (2%)	11116 (5%)
10	40 (4%)	16748 (4%)	15892 (7%)
12	58 (6%)	25108 (6%)	22429 (11%)
14	78 (9%)	33792 (8%)	30522 (14%)
16	118 (14%)	57179 (14%)	48594 (23%)
18	143 (17%)	70028 (17%)	59304 (29%)
20	173 (20%)	82783 (20%)	71352 (35%)
22	191 (22%)	94045 (23%)	82545 (40%)

The performance of these two designs in terms of latency is compared in Figure 5-9, which shows the number of clock cycles required by the coprocessor to compute the x_LS output vs number of range gates. Decreasing the initiation interval by a factor of two reduces the latency for number of range gates larger than 16 (or LS matrices larger than 16×16).

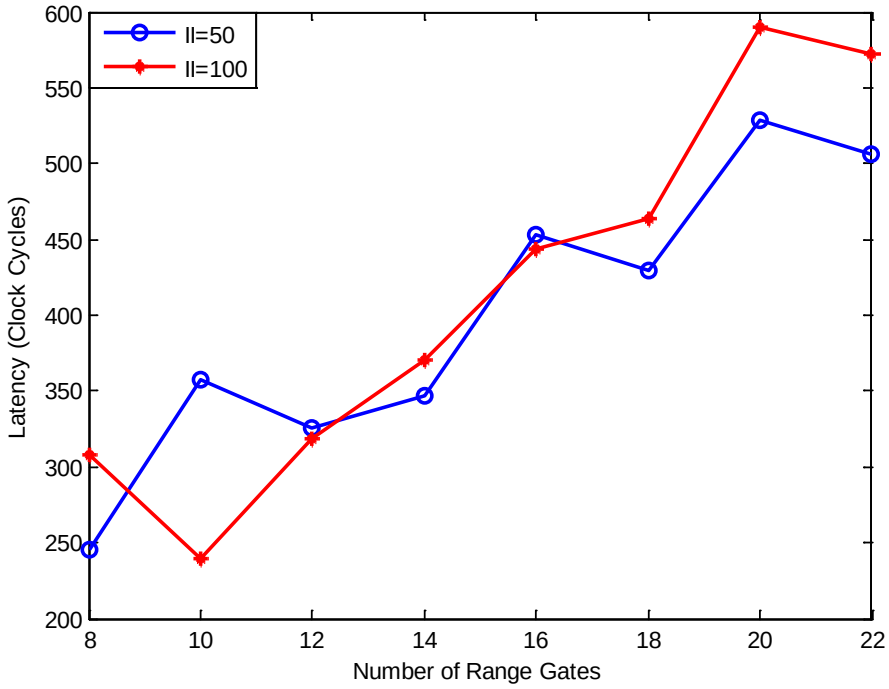


Figure 5-9: Comparison of latency in terms of clock cycles for different initiation intervals when number of samples is 6.

It has been shown that the low latency version of the LS occupies more silicon area, which can lead to more power consumption. An estimation of the dynamic power consumption is presented in Figure 5-10. The estimations are calculated using the Xilinx Power Estimation (XPE) tool. The dynamic power consumption for the

pipelined versions is clearly higher than the non-pipelined version, and also the implementation with II=50 consumes about 1.5 to 2 times more power than the II=100 version, due to the additional hardware resources demanded.

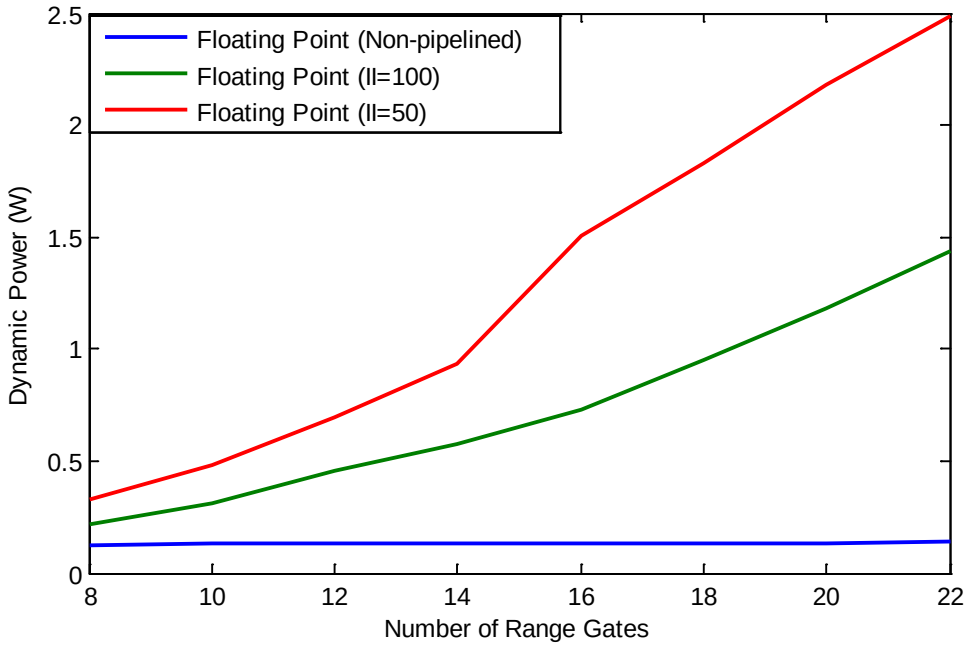


Figure 5-10: Comparison of dynamic power consumption required by the LS coprocessor.

If the waveform is fixed (no adaptive or dynamic waveforms), an improvement to the LS architecture is to pre-calculate the matrix S^H and $(S^H S)^{-1}$, store them in on-chip memory and then apply to the received signal vector y, as shown in Figure 5-11. This approach will not only maximize the throughput but also minimize the latency, which will be mainly dependent on a matrix-vector multiplication with $L^2 + NL - L$ multiplications.

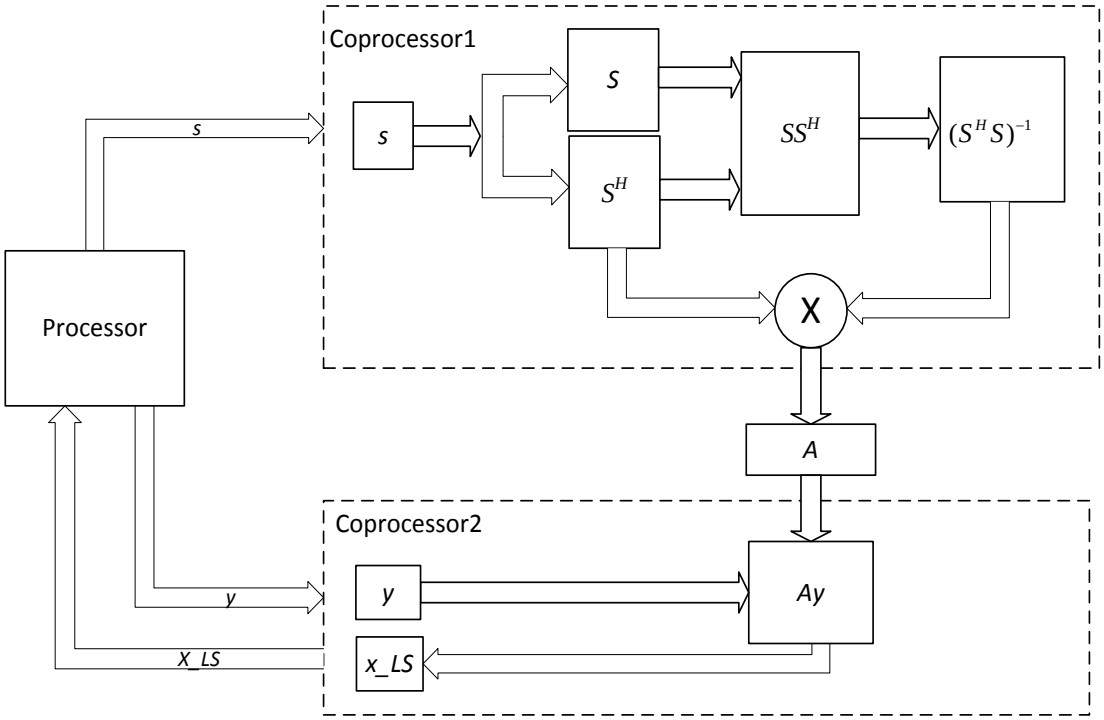


Figure 5-11: Architecture for fixed-waveform architecture, where Coprocessor 1 is only activated for the estimation of the filter coefficients.

5.6 RMMSE-APC Co-Processor Architecture

The RMMSE algorithm is described in [2]. The algorithm requires the calculation of matrix $C(l) = \sum_{n=-N+1}^{n=N+1} \rho(l+n+N-1)\, s_n s_n^H$ and matrix inversion $(C+R)^{-1}$ for each range gate. Here ρ is defined as the target signal power for the range bins, s_n represents the n-samples, shifted version of the waveform s, and R is the noise covariance matrix.

The hardware architecture for the RMMSE co-processor is presented in Figure 5-12. The matrices R and $SS(n)$ are pre-calculated and stored in on-chip memory, and $SS(n) = s_n s_n^H$, where $n \in [-N+1, N-1]$. The vectors y, ρ and s are also put into the on-chip memory. The values of ρ are updated in each iteration, and reduced in size by N-1 elements.

The filter weights for a range gate are estimated by matrix-vector multiplication operation and scaled by the value of ρ, as follows: $w = \rho(l+N-1)\,(C+R)^{-1}s$. These weights are applied to the incoming signal vector to obtain the estimated ground truth. The operation can be represented by an N-vector multiplication: $\hat{x} = w^T\,[y(l+(k-1)(N-1)) \ldots y(l+k(N-1))]$. The target signal power for the range bins is scaled by a constant value $\eta \in [0,2]$ to guarantee convergence stability. This last step is performed in software.

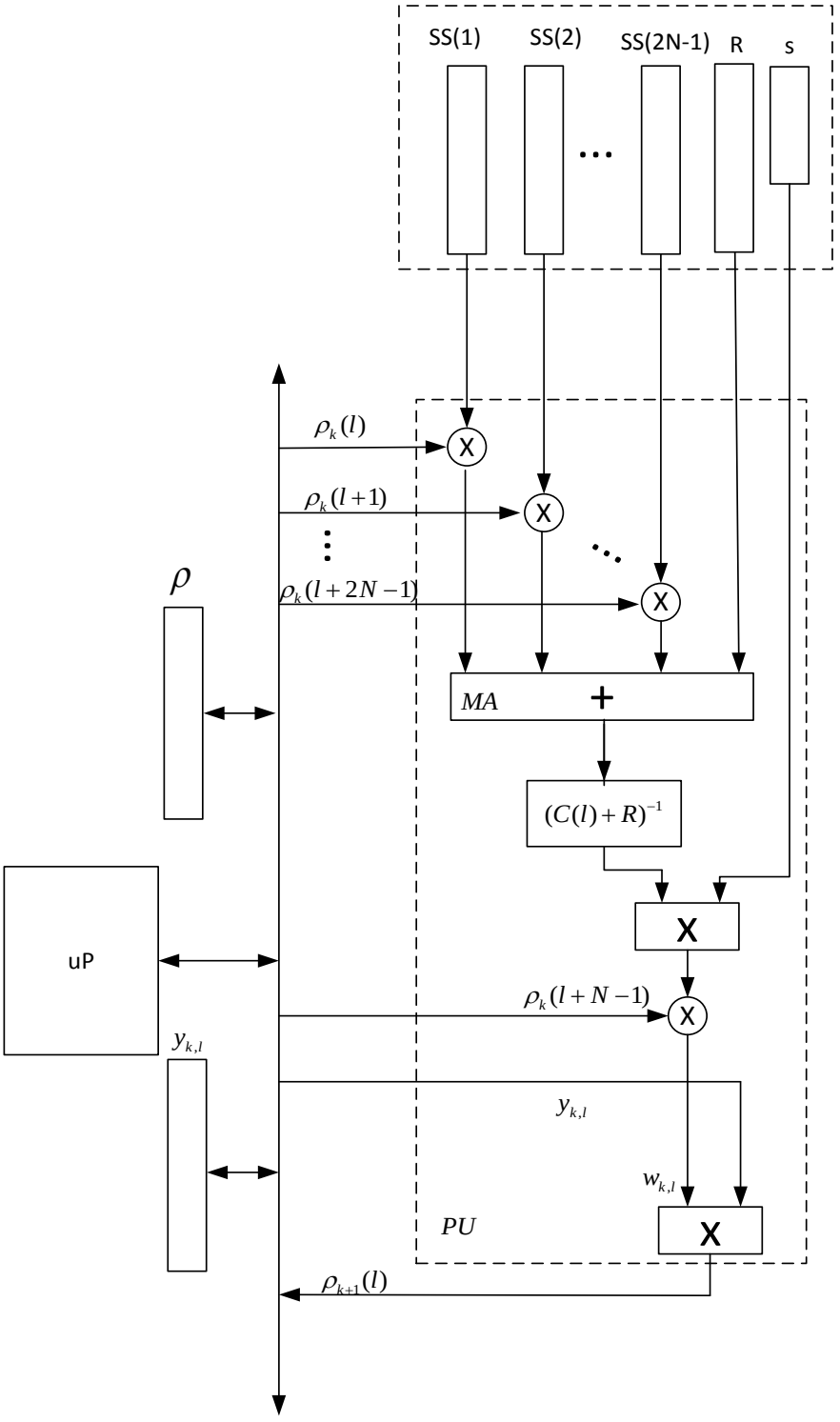


Figure 5-12: RMMSE coprocessor architecture.

The synthesis results for this architecture are shown in Table 5-8. The coprocessor performs the computation in a sequential mode using minimal hardware resources, in which hardware resources are reused in the different processing blocks for the estimation of the range gates.

Table 5-8: RMMSE coprocessor synthesis results.

N (Samples)	BRAM	DSP48E	FF	LUT
8	9 (1%)	12 (1%)	2726 (~0%)	4482 (2%)
10	13 (1%)	12 (1%)	2792 (~0%)	4630 (2%)
12	17 (1%)	12 (1%)	2823 (~0%)	4661 (2%)
14	25 (2%)	12 (1%)	2905 (~0%)	4700 (2%)
16	25 (2%)	12 (1%)	2789 (~0%)	4635 (2%)
18	42 (4%)	12 (1%)	2866 (~0%)	4783 (2%)
20	42 (4%)	12 (1%)	2864 (~0%)	4792 (2%)
22	74 (8%)	12 (1%)	2994 (~0%)	4907 (2%)

For this design, the achieved clock frequency was 118.76 MHz. The latency, which is number of clock cycles for a range gate estimation, versus the length of waveform s is displayed in Figure 5-13. It can be observed that the latency is on the order of 10^5 clock cycles, with maximum and minimum latency generated by the matrix inversion execution and increasing with the number of elements. Considering the achieved clock frequency, 500 range gates, and three iterations, the total latency over a range profile when N=16 would be approximately 1.6 seconds.

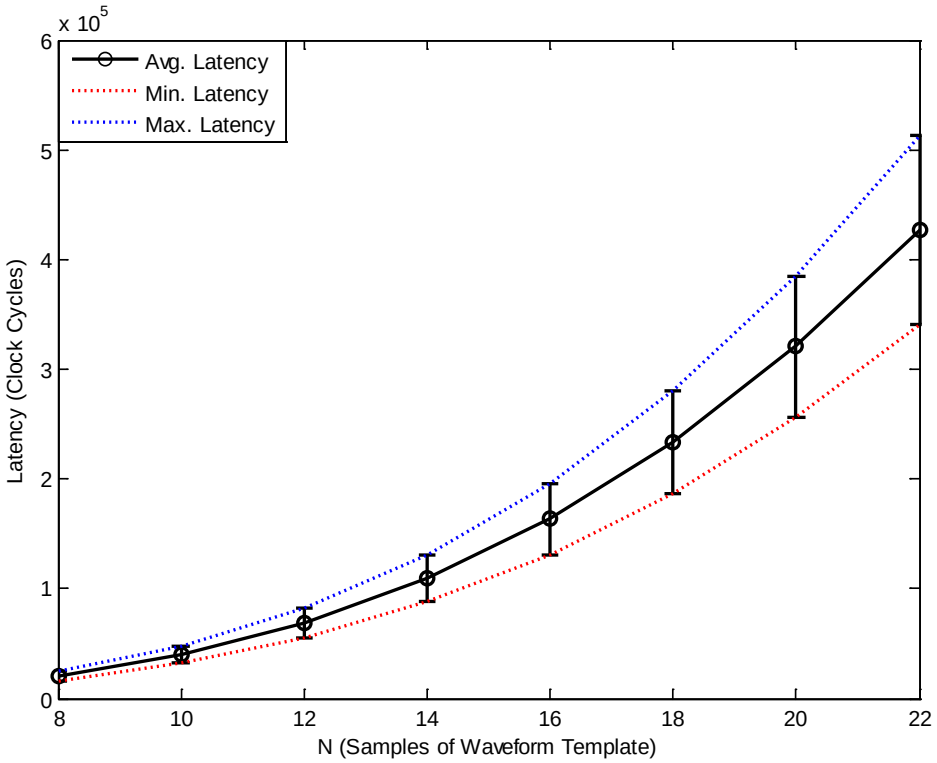


Figure 5-13: Latency estimation per range gate without optimization.

Moreover, for this implementation, besides matrix inversion, the scaling block and matrix addition units require 24% and 34% of the total latency, respectively. To reduce the critical path in the computation process, three different architectures were considered. In the first architecture, called partial pipelined, the matrix inversion is maintained in sequential mode, while the other processing units are pipelined. The initiation interval for the second and third architecture is constrained to be less than 2000 and 1000 clock cycles, respectively. As can be observed, latencies of the pipelined architectures, shown in Figure 5-14, are lower than those achieved in the sequential configuration. The latency reduction factor also varies smoothly linear with the number of samples (N). For instance, when N=16 the fully pipelined architectures

performs about 7 and 15 times faster than the partially pipelined and the sequential architecture respectively, which represents a latency of 94 ms, considering 8.42 ns of clock period, 500 range gates, and 3 iterations.

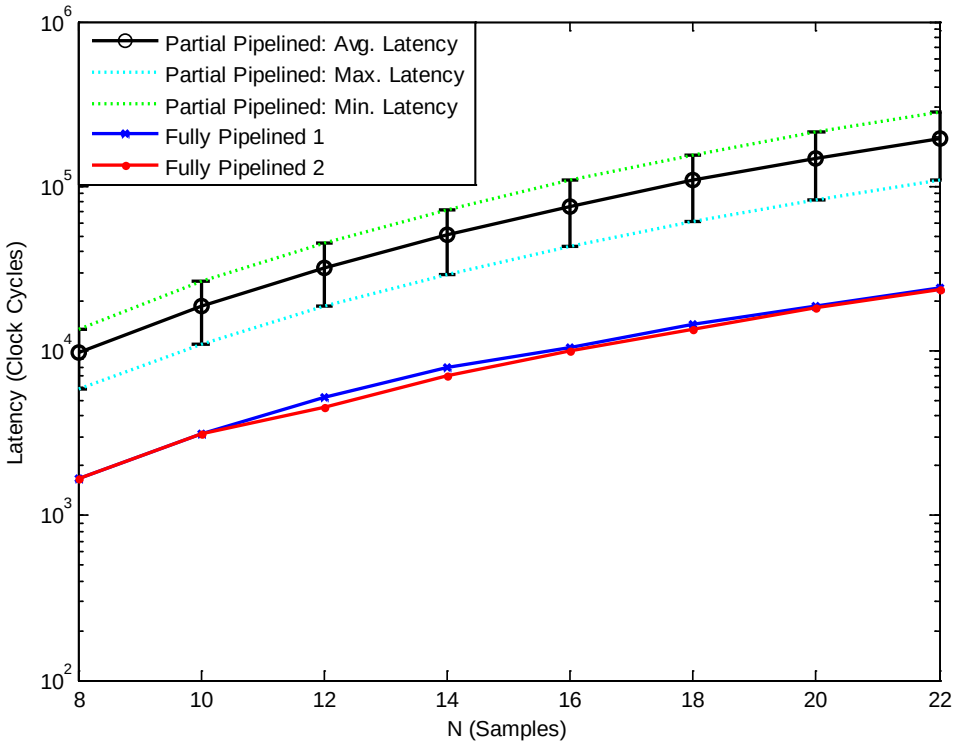


Figure 5-14: Latency comparison of implementation of RMMSE coprocessor.

In terms of hardware resource utilization, the partially pipelined architecture occupies more silicon area.

Table 5-9: Hardware resources for partially pipelined version of RMMSE coprocessor.

N (Samples)	BRAM	DSP48	FF	LUT
8	12 (1%)	17 (2%)	5906 (1%)	7794 (3%)
10	16 (1%)	17 (2%)	7139 (1%)	8937 (4%)
12	20 (2%)	17 (2%)	8907 (2%)	11093 (5%)
14	28 (3%)	17 (2%)	10682 (2%)	12887 (6%)
16	28 (3%)	17 (2%)	12637 (3%)	16868 (8%)
18	45 (5%)	17 (2%)	15005(3%)	18289 (8%)
20	45 (5%)	17 (2%)	17560 (4%)	21314 (10%)
22	77 (8%)	17 (2%)	20592 (5%)	25243 (12%)

However, constraining the initiation interval of the matrix inversion to be less than a determined number of clock cycles conditions the synthesis tool to use much more logic resources, as seen in Table 5-10.

Table 5-10: Hardware Resources for fully pipelined RMMSE coprocessor.

N (Samples)	BRAM	DSP48	FF	LUT
8	6 (~0%)	15 (1%)	12601 (3%)	14592 (7%)
10	8(~0%)	15 (1%)	17999 (4%)	21019 (10%)
12	12 (1%)	15 (1%)	24860 (6%)	30410 (14%)
14	20 (2%)	15 (1%)	32916 (8%)	42041 (20%)
16	20 (2%)	20 (2%)	40003 (9%)	53124 (26%)
18	37 (4%)	20 (2%)	49827 (12%)	65934 (32%)
20	37 (4%)	25 (2%)	59307 (14%)	83710 (41%)
22	69 (7%)	30 (3%)	70851 (17%)	106517 (52%)

Preserving a balance between latency and hardware resource utilization can let us find a better approach in order to reduce the processing latency by instantiating multiple processing units (PUs). In which, for each range gate estimation, a set of ρ values are streamed through a bidirectional bus to scale the matrix SS and form the $N \times N$ matrix C. $Tr = L + 2(M - k)(N - 1)$ and $Tw = [Tr + (k-1)(N-1) \dots Tr + k(N-1)]$. The number of concurrent PUs is determined by the hardware resources available in the FPGA.

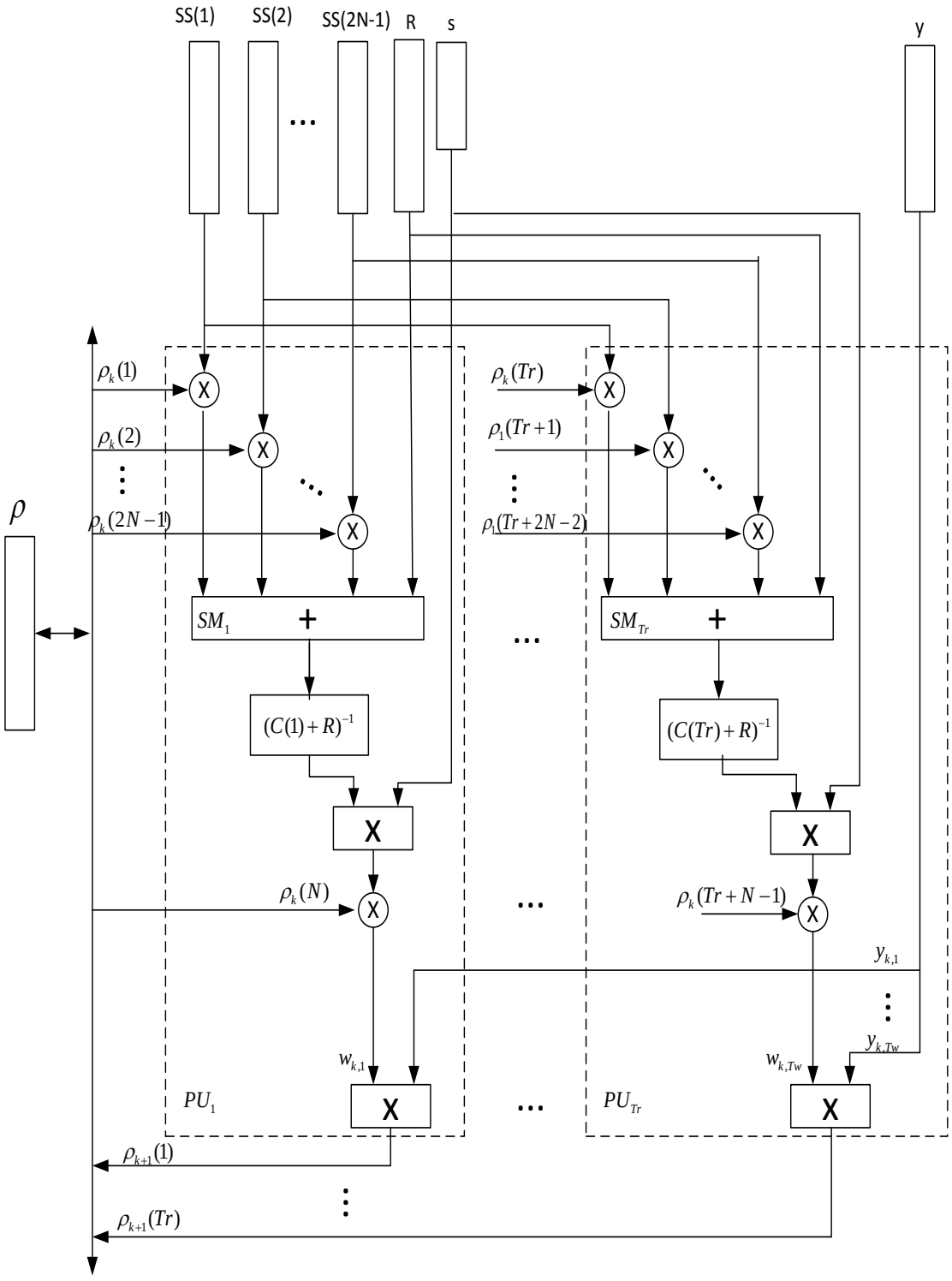


Figure 5-15: Architecture of RMMSE processor.

It has been also noticed that the addition of multiple matrices to form $c(l)+R$ also demands an equivalent number of clock cycles to that of the matrix inversion process. An alternative architecture to improve the latency is shown in Figure 5-16. The sum of 2N matrices, including the noise covariance matrix, is performed in the SM_l processing block. This matrix addition architecture requires a total of $m=\log_2(N)$ stages, and the computation of $\frac{N}{q}$ $N\times N$-matrix additions in parallel per stage, where $1\le q\le m$. The total latency for this block is determined by m times the latency of the addition of two matrices.

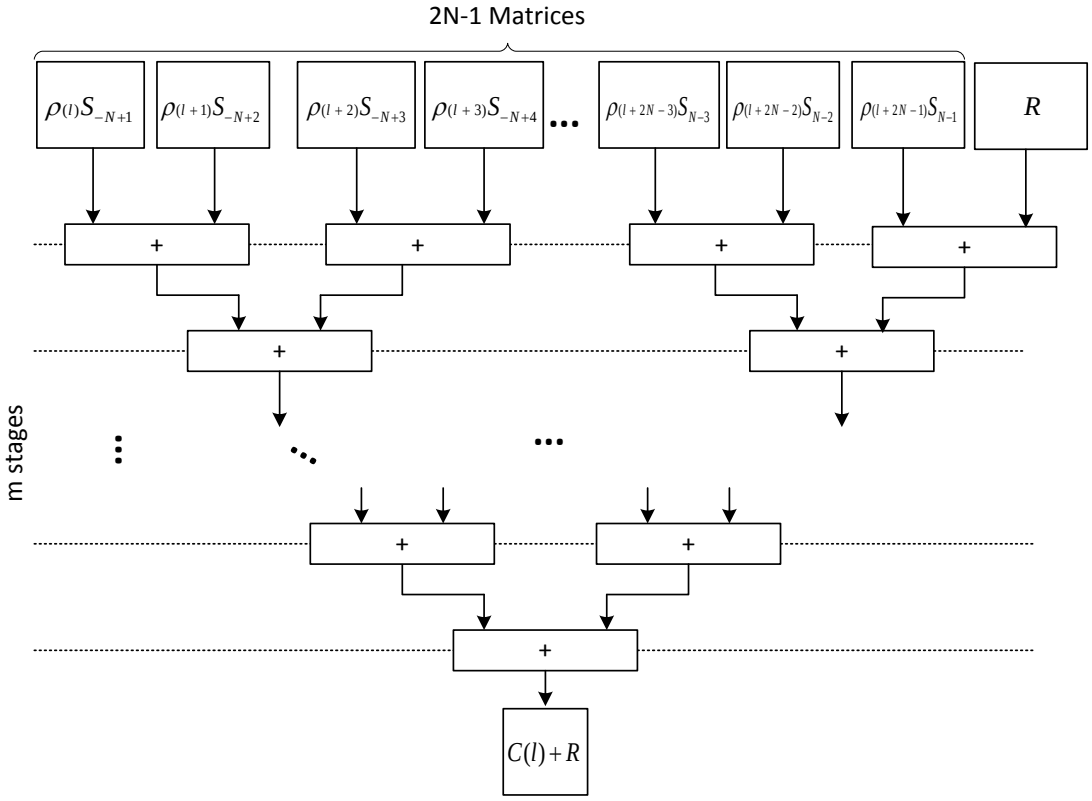


Figure 5-16: Architecture of the matrix summation to compute the matrix $C(l)+R$ for a range gate.

Using this architecture and a sequential micro-architecture for two-input matrix addition, the latency is reduced by about seven times compared to the sequential computation of matrix additions. Moreover, when the two-input matrix addition's micro-architecture is parallelized, the latency is reduced by about 25 times, but the number of hardware resources needed is increased significantly.

5.7 Summary

This chapter discusses the implementation of basic APC algorithms on FPGA as a part of the overall radar transceiver optimization processor. The flexibility of a SoC platform allows the implementation of different architectures in a single device. There are many different options. The first proposed architecture demonstrates that matrix multiplication and matrix inversion, as key operations for the implementation of APC algorithms, can be attached to the on-chip host processors to speed up the APC computation. The feasibility of implementing floating point version on FPGA is also validated, at the expense of more hardware resources, compared to the fixed point representation. Different techniques can improve the real-time performance of the on-chip system and reduce the overall latency, but demand more FPGA resources and power consumption. Hand tuning in the optimization for fixed-point implementation is required, especially when the computation involves arithmetic operations other than multiplication and addition. The focus at this point is Least-Squares (LS) operation while an initial RMMSE co-processor architecture is also presented.

Chapter 6

Conclusions

Adaptive Pulse Compression (APC) is a series of radar signal processing algorithms that are independent of waveforms and achieves an optimal estimation of ground-truth. APC algorithms have been shown to work for both point target and distributed target scenarios. The performance APC comes with the cost of computational load. In this dissertation, the performance of different fast arithmetic architectures on the Xilinx Kintex-7 FPGA is studied. Hardware accelerators were developed as coprocessors for APC, achieving performance improvements. Hardware implementation of pulse compression is presented. This study also seeks an optimal configuration for the tradeoffs between latency and hardware utilization for the implementation of APC, for which the RMMSE and LS algorithms are considered. The system architectures are based on the embedded processor which is interconnected with the logic resources through the on-chip AXI buses.

6.1 Achievements

The major innovative contributions of this work are summarized as follows:

(1) The performance of fast adder and multiplier architectures on a Kintex-7 FPGA device was rigorously investigated. It was also shown that LUT-

based/DSP-based adder and multiplier achieved better performance compared to the other fast architectures.

(2) The use of matrix multiplication and Cholesky-based matrix inversion as coprocessor units for embedded systems to accelerate intensive computational algorithms was studied. The feasibility of implementing floating-point matrix operations on an FPGA and the performance were analyzed.

(3) A waveform-independent, real-time pulse compression (matched filter) processor architecture was implemented on FPGA. Different windowing/ weighting functions were also included in the architecture, which can mitigate range sidelobes in real-time. Implementation of the architecture was demonstrated on a Kintex-7 FPGA, as a part of a Ku-band spaceborne radar transceiver testbed. As an IP-Core, this pulse compression processor can be easily reconfigured or used as a part of the SoC system architecture.

(4) Established a complete radar RF/IF transceiver processor based on SoC system architectures. A proposed implementation example, which is based on Xilinx Zynq, has included adaptive pulse compression and adaptive pre-distortion processing in the system. The example implementation laid out a basic framework for future more completed SoC implementations.

(5) Designed and developed different FPGA hardware implementations for typical adaptive pulse compression processing, i.e., LS and RMMSE. Architectures for these implementations are analyzed; these architectures

can significantly reduce the overall processing latency at the expenses of more hardware resources.

6.2 Future Work

(1) The integration of the APC architecture with adaptive pre-distortion will lead to a complete transceiver optimization system as shown in Figure 5-2. This system will improve detection, and will be very useful in airborne radars, such as the High-Altitude Imaging Wind and Rain Airborne Profiler (HIWRAP) [135], whose backend system is based on Virtex-5 FPGA and Power PC processor, and where strict size, weight and power constraints are required.

(2) Place-and-route improvements to achieve maximum throughput and reduced latencies. In addition, to reduce critical paths and achieve a higher computing clock frequency, the insertion of registers may also be required.

(3) Implementation of an optimized version of adaptive pulse compression with reduced matrix sizes can be considered.

(4) Other multiprocessor architectures with multiple cache levels for reducing memory latencies should be studied.

(5) Expand the SoC architecture to support 2D-LS and 2D-RMMSE processing and mitigate the sidelobes from both pulse compression and antenna pattern [136].

(6) As part of the future work, two aspects that need to be taken into consideration are the trends in the technology process of the semiconductor

devices, and also how the APC processor can be interfaced with the other elements of the system. This will be further elaborated in the following sections.

6.2.1 Technology Trend for FPGA-Based Signal Processing

As mentioned in Chapter 1, innovations in the semiconductor technology will allow the integration of more transistors in a single die. This scaling process will allow the incorporation of more hardware resources in a single FPGA, which will also improve the implementation and execution of more sophisticated radar algorithms in a single device using not only fixed-point numbers but also floating-point representation, and overcoming any dynamic range and scaling issues. Semiconductor companies have been working on shrinking a die even more. To the date of this work, the latest Xilinx's devices, called UltraScale+, are based on the 16 nm FinFET+ technology. For instance, the new Zynq UltraScale+, known as the multiprocessing system-on-chip (MPSoC) device, includes an application processing unit (dual-core ARM v8-based Cortex-R5 processor), a real-time processing unit (quad-core ARM Cortex A-53 processor), a graphic processing unit (ARM Mali-400 MP2 GPU), and the logic fabric which is based on the UltraScale+ architecture [137]. Moreover, Altera has introduced its new high-end FPGAs, the Stratix 10, based on the Intel 14nm technology, which also includes a quad-core ARM Cortex A-53 processor, and claimed to achieve up to 10 Tera FLOPS of IEEE 754 single-precision floating-point operations. Figure 6-1 shows the hardware architecture of the new Zynq device which includes the processing system and the programmable logic.

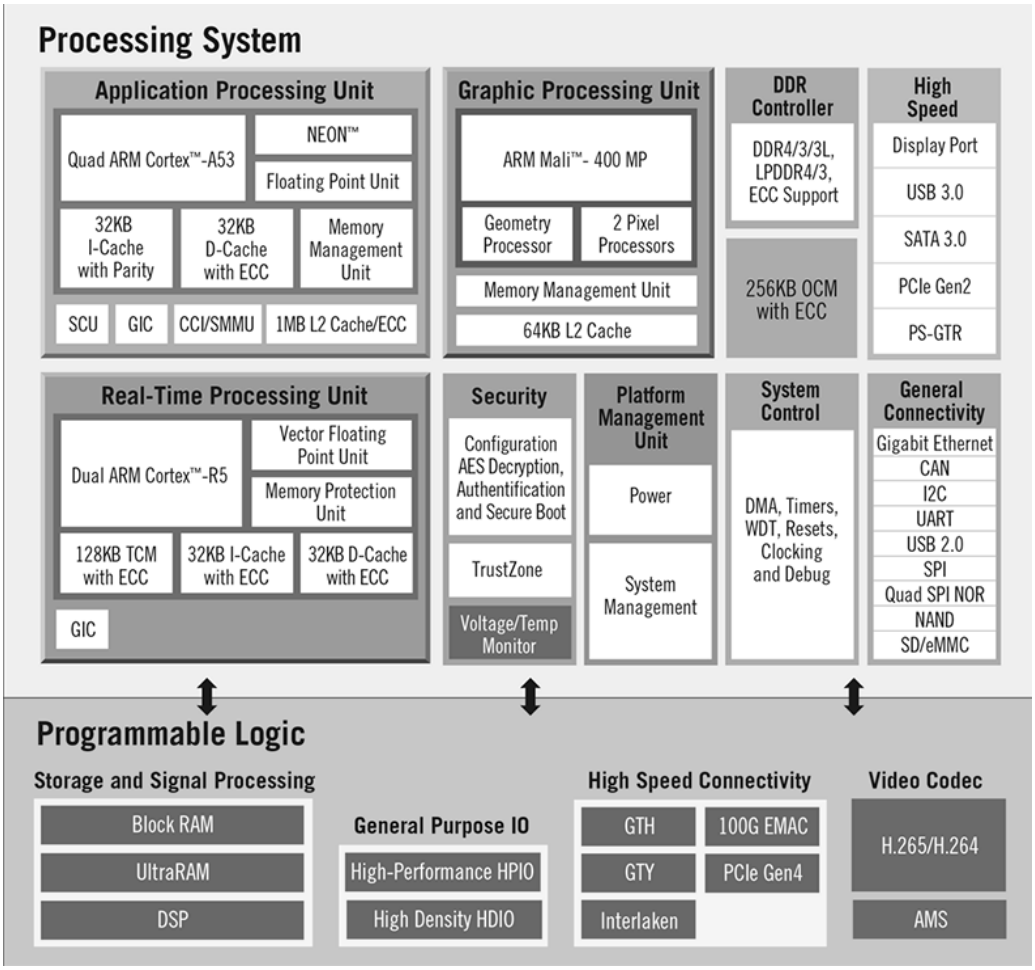


Figure 6-1: Illustration from Xilinx. The new Zynq UltraScale+ architecture [138]

Even smaller process technologies will still be possible, as stated in the International Technology Roadmap for Semiconductor (ITRS), which is published by a group of semiconductor experts, in which it was projected that the 7 nm technology will be reached by 2017, and the 5 nm technology by 2019 [139]. Up to the date of this work, Xilinx is planning to introduce its new FPGA family based on the 7 nm technology by 2017 [140]. However, with the increased complexity of new devices,

hardware implementation of advanced algorithms will also require more sophisticated software tools and pose bigger challenges to designers.

6.2.2 Integration of the APC processor to a Radar System

How the APC processor can be interfaced with other elements of the system is another important topic to consider. The use of a serial transceiver has been essential to move data efficiently within the system. A serial transceiver is generally comprised of serializer/deserializer (SerDes), buffers, encoder/decoder, PLLs, and data flow controllers. The new FPGAs families also incorporate more efficient and faster serial transceivers. The maximum number of transceivers included in a Xilinx Virtex UltraScale+ family is 128 with a speed of up to 32 Gbps per lane [137]. Figure 6-2 shows an open architecture for an airborne radar system based on the serial transceivers which can also be extended for other radar application such as phased array radars.

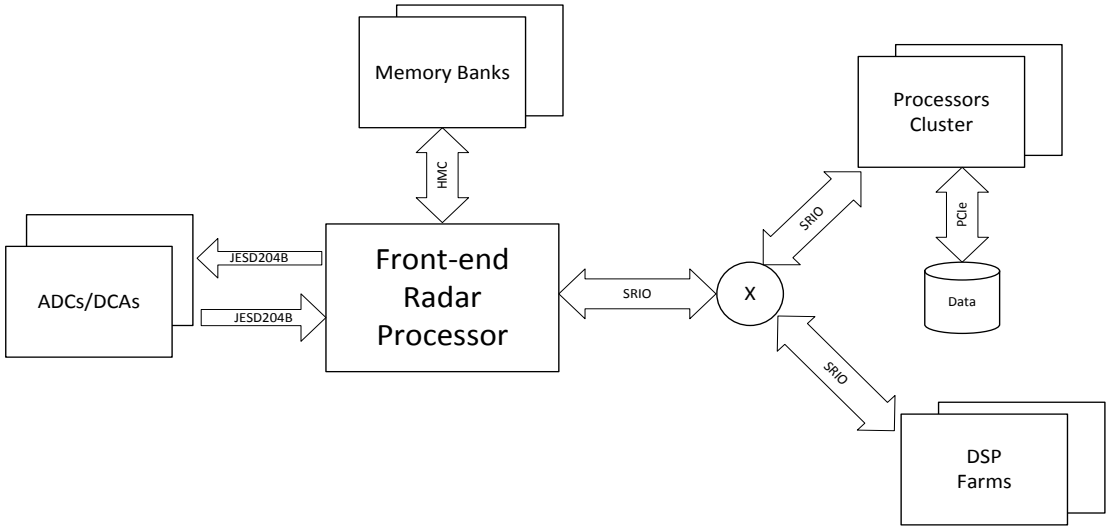


Figure 6-2: A general architecture of a radar processing system based on serial technologies.

Using a serial interface will provide some benefits such as lower number of I/O pins, smaller package size, and lower power consumption. A typical interconnection between ADC/DAC and FPGA has been through simple low-voltage differential signaling (LVDS) links. However, due to the increasing bandwidth needs of new radar applications, the new interfaces are based on multi-gigabit serial data link standard such as JESD204B which can achieve a speed rate of 12.5 Gbps per lane. The standard is developed by the Joint Electron Device Engineering Council (JEDEC) [141].

A similar situation can be observed when an embedded processor needs to interface with external memory. Due to limitations in the storage capacity and speed of the conventional double data rate memory (DDR), several serial memory technologies have been developed such as Bandwidth Engine (BE), Ternary Content Addressable Memory (TCAM), High Bandwidth Memory (HBM), and Hybrid

Memory Cube (HMC). The HMC technology (HMC 2.0), supported by the HMC consortium group, can use up to four 16-lane serialized links with a speed of 30 Gbps per lane, and provide up to 320 GB/s effective bandwidth with low power consumption [142].

Another important aspect to consider is the capacity for communicating efficiently with other processing elements of the system. Therefore, in order to increase the scalability, robustness and network performance of the system, several standards based on switched serial interconnects have been utilized. Some of those standards are Gigabit Ethernet (GbE), RapidIO (RIO) [143], PCI Express (PCIe), and InfiniBand (IB) [144]. However, the Serial RapidIO (SRIO) standard has been presented as a suitable technology for interconnecting elements in the backplane, providing low latency, high reliability, and routable interconnections. This technology is developed and supported by the RapidIO Trade Association [143]. The link width options for SRIO Gen2 are 1x, 2x, 4x, 8x, and 16x, with five possible lane speeds: 1.25, 2.5, 3.125, 5.0, and 6.25 Gbaud; up to 10.3125 Gbaud for SRIO Gen3 and 25 Gbaud for its next generation. The implementation of Xilinx's SRIO Gen2 IP for one lane requires about 5650 LUTs, 6050 FFs, and 2900 Slices [145]. SRIO is generally characterized in a three layer architectural hierarchy: physical, transport, and logical layer. The physical layer defines the electrical connection of devices on a board or across a backplane. The transport layer provides the route information to move packet from end to end in the system. The logical layer defines the overall protocol and packet format, and also initializes and completes transactions [146].

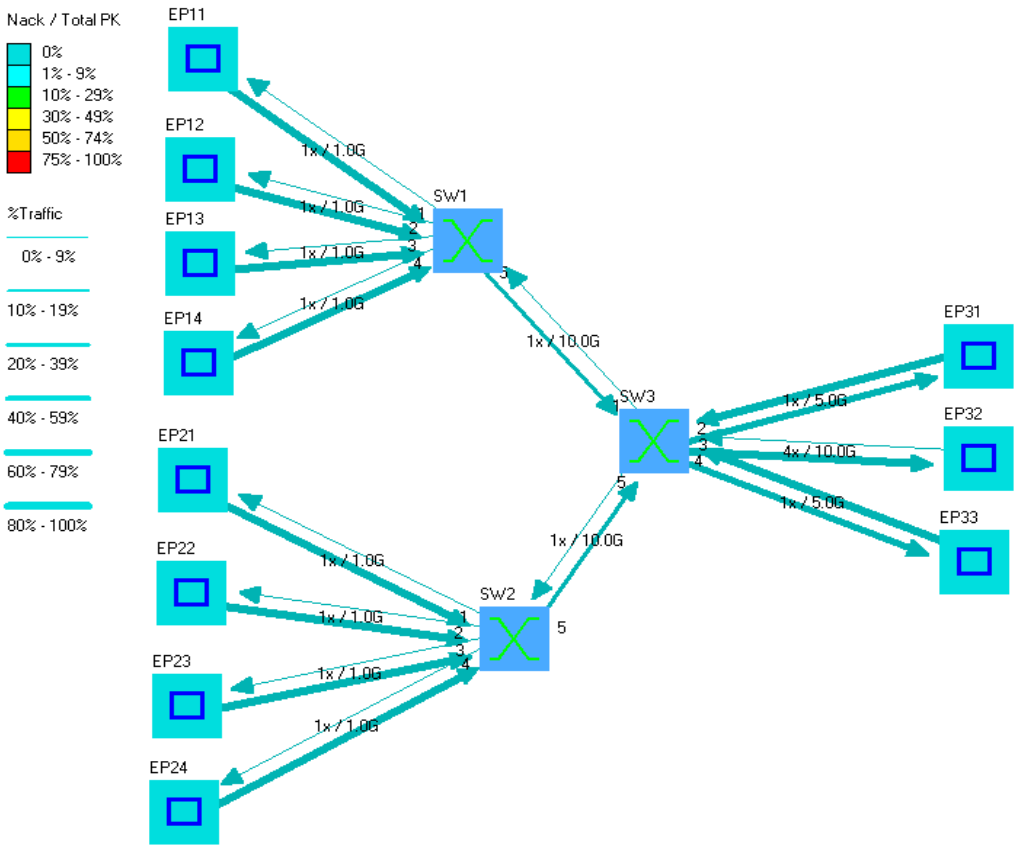


Figure 6-3: Simulation of a RapidIO-based network.

A simulated RapidIO-based network is shown in Figure 6-3, using Integrated Device Technology (IDT)‘s SRIO modeling tool [147], where the end-points EP1x and EP2x stream data to be computed in EP31 and EP33, respectively, through the switches (SWx). The results are then streamed to EP32.

Bibliography

[1] M. I. Skolnik, *Introduction to radar systems*, 3rd ed. Boston: McGraw Hill, 2001.

[2] S. D. Blunt and K. Gerlach, "Adaptive pulse compression via MMSE estimation," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. 42, pp. 572-584, 2006.

[3] S. D. Blunt and T. Higgins, "Dimensionality Reduction Techniques for Efficient Adaptive Pulse Compression," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. 46, pp. 349-362, 2010.

[4] S. D. Blunt, A. K. Shackelford, K. Gerlach, and K. J. Smith, "Doppler Compensation & Single Pulse Imaging using Adaptive Pulse Compression," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. 45, pp. 647-659, 2009.

[5] K. Gerlach, A. K. Shackelford, and S. D. Blunt, "Combined Multistatic Adaptive Pulse Compression and Adaptive Beamforming for Shared-Spectrum Radar," *Selected Topics in Signal Processing, IEEE Journal of,* vol. 1, pp. 137-146, 2007.

[6] S. D. Blunt and K. Gerlach, "Multistatic adaptive pulse compression," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. 42, pp. 891-903, 2006.

[7] A. Tanner, S. L. Durden, R. Denning, E. Im, F. K. Li, W. Ricketts, *et al.*, "Pulse compression with very low sidelobes in an airborne rain mapping radar," *Geoscience and Remote Sensing, IEEE Transactions on,* vol. 32, pp. 211-213, 1994.

[8] A. Johnston, "Improvements to a pulse compression radar matched filter," *Radio and Electronic Engineer,* vol. 53, pp. 138-140, 1983.

[9] M. Richards, *Fundamentals of Radar Signal Processing*: McGraw-Hill, 2005.

[10] M. A. Richards, J. Scheer, and W. A. Holm. (2010). *Principles of modern radar*.

[11] A. B. Robert and A. Masahiro, "Computational Characteristics of High Performance Embedded Algorithms and Applications," in *High Performance Embedded Computing Handbook*, ed: CRC Press, 2008, pp. 73-112.

[12] I. R. Albert and A. B. Robert, "Radar Signal Processing," in *High Performance Embedded Computing Handbook*, ed: CRC Press, 2008, pp. 113-145.

[13] G. E. Moore, "Cramming More Components Onto Integrated Circuits," *Proceedings of the IEEE*, vol. 86, pp. 82-85, 1998.

[14] Freescale, "MPC5500 Family Overview," ed, 2006.

[15] L. Collins and C. Edwards, "Head to head," *Engineering & Technology*, vol. 4, pp. 38-41, 2009.

[16] Xilinx, "7 Series FPGAs Overview (DS180)," ed, 2014.

[17] Altera, "Embedded Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide," ed, 2014.

[18] Xilinx, "7 series FPGAs Configurable Logic Block," 2014.

[19] C. Shang-Yi, "Foundries and the dawn of an open IP era," *Computer*, vol. 34, pp. 43-46, 2001.

[20] L. H. Crockett, R. A. Elliot, M. A. Enderwitz, and R. W. Stwart, *The Zynq Book: Embedded processing with the ARM Cortex-A9 on the Xilinx-7000 All Programmable SoC*, 1st ed.: Strathclyde Academic Media, 2014.

[21] J. R. Hauser and J. Wawrzynek, "Garp: a MIPS processor with a reconfigurable coprocessor," in *Field-Programmable Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on*, 1997, pp. 12-21.

[22] S. C. Goldstein, H. Schmit, M. Budiu, S. Cadambi, M. Moe, and R. R. Taylor, "PipeRench: a reconfigurable architecture and compiler," *Computer*, vol. 33, pp. 70-77, 2000.

[23] S. Hauck, T. W. Fry, M. M. Hosler, and J. P. Kao, "The Chimaera reconfigurable functional unit," in *Field-Programmable Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on*, 1997, pp. 87-96.

[24] P. M. Athanas, "A functional reconfigurable architecture and compiler for adaptive computing," in *Computers and Communications, 1993., Twelfth Annual International Phoenix Conference on*, 1993, pp. 49-55.

[25] M. Wazlowski, L. Agarwal, T. Lee, A. Smith, E. Lam, P. Athanas, *et al.*, "PRISM-II compiler and architecture," in *FPGAs for Custom Computing Machines, 1993. Proceedings. IEEE Workshop on*, 1993, pp. 9-16.

[26] R. D. Wittig and P. Chow, "OneChip: an FPGA processor with reconfigurable logic," in *FPGAs for Custom Computing Machines, 1996. Proceedings. IEEE Symposium on*, 1996, pp. 126-135.

[27] T. Miyamori and K. Olukotun, "REMARC: reconfigurable multimedia array coprocessor," presented at the Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays, Monterey, California, USA, 1998.

[28] S. Vassiliadis, S. Wong, and S. Cotöfană, "The MOLEN ρμ-Coded Processor," in *Field-Programmable Logic and Applications*. vol. 2147, G. Brebner and R. Woods, Eds., ed: Springer Berlin Heidelberg, 2001, pp. 275-285.

[29] A. Lodi, A. Cappelli, M. Bocchi, C. Mucci, M. Innocenti, C. De Bartolomeis, *et al.*, "XiSystem: a XiRisc-based SoC with reconfigurable IO module," *Solid-State Circuits, IEEE Journal of,* vol. 41, pp. 85-96, 2006.

[30] Xilinx, "MicroBlaze Processor Reference Guide," Xilinx, Ed., ed, 2013.

[31] Altera, "Nios II Classic Processor Reference Guide," 2015.

[32] Xilinx, "AXI Reference Guide (UG761)," ed, 2012.

[33] H. D. Griffiths, "Design of low-sidelobe pulse compression waveforms," *Electronics Letters,* vol. 30, pp. 1004-1005, 1994.

[34] T. Ihara, K. Okamoto, T. Kozu, J. Awaka, K. Nakamura, and M. Fujita, "Development Of Key Devices For TRMM Rain Radar," in *Geoscience and Remote Sensing Symposium, 1991. IGARSS '91. Remote Sensing: Global Monitoring for Earth Management., International*, 1991, pp. 513-516.

[35] H. D. Griffiths, L. Vinagre, and W. K. Lee, "Developments in radar waveform design," in *Microwaves and Radar, 1998. MIKON '98., 12th International Conference on*, 1998, pp. 56-76 vol.4.

[36] A. S. Mudukutore, V. Chandrasekar, and R. J. Keeler, "Pulse compression for weather radars," *Geoscience and Remote Sensing, IEEE Transactions on,* vol. 36, pp. 125-142, 1998.

[37] C. E. Cook and M. Bernfeld, *Radar signals; an introduction to theory and application*. New York,: Academic Press, 1967.

[38] A. H. Nuttall, "Some windows with very good sidelobe behavior," *Acoustics, Speech and Signal Processing, IEEE Transactions on,* vol. 29, pp. 84-91, 1981.

[39] F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proceedings of the IEEE,* vol. 66, pp. 51-83, 1978.

[40] E. de Witte and H. D. Griffiths, "Improved ultra-low range sidelobe pulse compression waveform design," *Electronics Letters,* vol. 40, pp. 1448-1450, 2004.

[41] J. M. Kurdzo, C. Boon Leng, R. D. Palmer, and Z. Guifu, "Optimized NLFM pulse compression waveforms for high-sensitivity radar observations," in *Radar Conference (Radar), 2014 International*, 2014, pp. 1-6.

[42] J. George, N. Bharadwaj, and V. Chandrasekar, "Considerations in Pulse Compression Design for Weather Radars," in *Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International*, 2008, pp. V - 109-V - 112.

[43] M. N. Cohen, "An overview of high range resolution radar techniques," in *Telesystems Conference, 1991. Proceedings. Vol.1., NTC '91., National*, 1991, pp. 107-115.

[44] R. L. Frank, "Polyphase codes with good nonperiodic correlation properties," *Information Theory, IEEE Transactions on,* vol. 9, pp. 43-45, 1963.

[45] B. L. Lewis and F. F. Kretschmer, "A New Class of Polyphase Pulse Compression Codes and Techniques," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. AES-17, pp. 364-372, 1981.

[46] B. L. Lewis and F. F. Kretschmer, "Linear Frequency Modulation Derived Polyphase Pulse Compression Codes," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. AES-18, pp. 637-641, 1982.

[47] T. Felhauer, "Design and analysis of new p(n,k) polyphase pulse compression codes," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. 30, pp. 865-874, 1994.

[48] J. P. Costas, "A study of a class of detection waveforms having nearly ideal range—Doppler ambiguity properties," *Proceedings of the IEEE,* vol. 72, pp. 996-1009, 1984.

[49] G. Welti, "Quaternary codes for pulsed radar," *Information Theory, IRE Transactions on,* vol. 6, pp. 400-408, 1960.

[50] M. J. E. Golay, "Complementary series," *Information Theory, IRE Transactions on,* vol. 7, pp. 82-87, 1961.

[51] Z. Li, Z. Yan, S. Wang, L. Li, and M. Mclinden, "Fast adaptive pulse compression based on matched filter outputs," *IEEE Trans. on Aerospace and Electronic Systems,* Submitted.

[52] M. H. Ackroyd and F. Ghani, "Optimum Mismatched Filters for Sidelobe Suppression," *Aerospace and Electronic Systems, IEEE Transactions on,* vol. AES-9, pp. 214-218, 1973.

[53] T. Felhauer, "Digital signal processing for optimum wideband channel estimation in the presence of noise," *Radar and Signal Processing, IEE Proceedings F,* vol. 140, pp. 179-186, 1993.

[54] R. C. Daniels and V. Gregers-Hansen, "Code inverse filtering for complete sidelobe removal in binary phase coded pulse compression systems," in *Radar Conference, 2005 IEEE International*, 2005, pp. 256-261.

[55] J. M. Lewis, S. Lakshmivarahan, and S. K. Dhall, *Dynamic Data Assimilation: A Least Squares Approach*. Cambridge, UK: Cambridge University Press, 2006.

[56] S. D. Blunt and K. Gerlach, "A novel pulse compression scheme based on minimum mean-square error reiteration [radar signal processing]," in *Radar Conference, 2003. Proceedings of the International*, 2003, pp. 349-353.

[57] R. D. Fierro, G. H. Golub, P. C. Hansen, and D. P. O'Leary, "Regularization by truncated total least squares," *SIAM Journal of Scientific Computing,* vol. 18, pp. 1223-1241, 1997.

[58] J. M. Cioffi and T. Kailath, "Fast, recursive-least-squares transversal filters for adaptive filtering," *IEEE Trans. on Acoustics, Speech and Signal Processing,* vol. 32, pp. 304-337, 1984.

[59] T. K. Moon and W. C. Stirling, *Mathematical Methods and Algorithms for Signal Processing*. Upper Saddle River, NJ: Prentice-Hall, 1999.

[60] J. W. Arthur, "Modern SAW-based pulse compression systems for radar applications. 2. Practical systems," *Electronics & Communication Engineering Journal,* vol. 8, pp. 57-78, 1996.

[61] J. W. Arthur, "Modern SAW-based pulse compression systems for radar applications. I. SAW matched filters," *Electronics & Communication Engineering Journal,* vol. 7, pp. 236-246, 1995.

[62] D. P. Morgan, "History of SAW devices," in *Frequency Control Symposium, 1998. Proceedings of the 1998 IEEE International*, 1998, pp. 439-460.

[63] M. B. N. Butler, "Radar applications of s.a.w. dispersive filters," *Communications, Radar and Signal Processing, IEE Proceedings F*, vol. 127, pp. 118-124, 1980.

[64] MESL. (2015, Jun 18). *SAW pulse compression*. Available: http://www.meslmicrowave.com/saw-pulse-compression/our-experience/

[65] J. J. Alter, J. B. Evins, J. L. Davis, and D. L. Rooney, "A programmable radar signal processor architecture," in *Radar Conference, 1991., Proceedings of the 1991 IEEE National*, 1991, pp. 108-111.

[66] Z. Xinggan and Z. Zhaoda, "A pulse compression processor implementation with DSP for airborne pulse Doppler radar," in *Digital Avionics Systems Conference, 1994. 13th DASC., AIAA/IEEE*, 1994, pp. 421-425.

[67] P. Tortoli, F. Guidi, and C. Atzeni, "Digital vs. SAW matched filter implementation for radar pulse compression," in *Ultrasonics Symposium, 1994. Proceedings., 1994 IEEE*, 1994, pp. 199-202 vol.1.

[68] M. Baldanzi, P. Tortoli, and C. Atzeni, "Programmable wideband signal generation and matched filtering through a full digital approach," in *Spread Spectrum Techniques and Applications Proceedings, 1996., IEEE 4th International Symposium on*, 1996, pp. 42-46 vol.1.

[69] R. H. Day, R. Germon, and B. C. O'Neill, "A pulse compression radar signal processor," in *DSP Chips in Real-Time Instrumentation and Display Systems (Digest No: 1997/300), IEE Colloquium on*, 1997, pp. 4/1-4/5.

[70] R. H. Day, R. Germon, and B. C. O'Neill, "A real time digital signal processing solution for radar pulse compression," in *Digital Filters: An Enabling Technology (Ref. No. 1998/252), IEE Colloquium on*, 1998, pp. 6/1-6/5.

[71] F. Li and T. Long, "A high-speed real-time digital pulse compression system based on TMS320C6201," in *Radar, 2001 CIE International Conference on, Proceedings*, 2001, pp. 557-561.

[72] H. Wei, H. Zhiming, L. Yajing, and X. Jingcheng, "Parallel processing algorithm study for pulse compression in general-purpose radar signal processing system," in *Communications, Circuits and Systems and West Sino Expositions, IEEE 2002 International Conference on*, 2002, pp. 953-957 vol.2.

[73] R. Andraka and A. Berkun, "FPGAs make a radar signal processor on a chip a reality," in *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on*, 1999, pp. 559-563 vol.1.

[74] A. C. Berkun, M. A. Fischman, and E. Im, "An advanced FPGA-based processor and controller for the Next-Generation Precipitation Radar," in *Geoscience and Remote Sensing Symposium, 2002. IGARSS '02. 2002 IEEE International*, 2002, pp. 780-782 vol.2.

[75] Y. Xie and T. Long, "Implementation of two dimensional pulse compression based on embedded processor in FPGA," in *Radar Conference, 2009 IET International*, 2009, pp. 1-4.

[76] M. Bahtat, S. Belkouch, P. Elleaume, and P. Le Gall, "Efficient implementation of a complete multi-beam radar coherent-processing on a telecom SoC," in *Radar Conference (Radar), 2014 International*, 2014, pp. 1-6.

[77] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*: Oxford University Press, Inc., 2009.

[78] Xilinx, "LogiCORE IP Floating-Point Operator v7.0 (PG060)," ed, 2014.

[79] J. Sklansky, "Conditional-Sum Addition Logic," *Electronic Computers, IRE Transactions on*, vol. EC-9, pp. 226-231, 1960.

[80] A. D. BOOTH, "A SIGNED BINARY MULTIPLICATION TECHNIQUE," *The Quarterly Journal of Mechanics and Applied Mathematics*, vol. 4, pp. 236-240, January 1, 1951 1951.

[81] H. Yajuan and C. Chip-Hong, "A New Redundant Binary Booth Encoding for Fast-Bit Multiplier Design," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 56, pp. 1192-1201, 2009.

[82] C. S. Wallace, "A Suggestion for a Fast Multiplier," *Electronic Computers, IEEE Transactions on*, vol. EC-13, pp. 14-17, 1964.

[83] L. Dadda, "Some Schemes for Parallel Multipliers," *Alta Frequenza*, vol. 34, pp. 349-356, August 1965.

[84] A. Avizienis, "Signed-Digit Numbe Representations for Fast Parallel Arithmetic," *Electronic Computers, IRE Transactions on*, vol. EC-10, pp. 389-400, 1961.

[85] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi, "A high-speed multiplier using a redundant binary adder tree," *Solid-State Circuits, IEEE Journal of*, vol. 22, pp. 28-34, 1987.

[86] N. Takagi, H. Yasuura, and S. Yajima, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree," *Computers, IEEE Transactions on*, vol. C-34, pp. 789-796, 1985.

[87] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara, and K. Mashiko, "An 8.8-ns 54×54-bit multiplier with high speed redundant binary architecture," *Solid-State Circuits, IEEE Journal of,* vol. 31, pp. 773-783, 1996.

[88] J. P. Elliott. (1999). *Understanding Behavioral Synthesis a Practical Guide to High-Level Design*. Available: SpringerLink http://dx.doi.org/10.1007/978-1-4615-5059-4 MIT Access Only

[89] N. Instruments. (2015, Aug. 18). *SDR Hardware Products*. Available: http://www.ni.com/sdr/products/

[90] Nutaq. (2015, Aug 21). *SDR Comparison Chart*. Available: http://www.nutaq.com/sdr-comparison-chart

[91] E. R. LLC. (2013). *Universal Software Radio Peripheral (USRP)*. Available: https://www.ettus.com/product/details/UN210-KIT

[92] Datasoft. (2015). *Microburst Software Defined Radio*. Available: http://www.datasoft.com/products/microburst/index.html

[93] Nuand. (2015, May 18). *BladeRF- The USB 3.0 Superspeed Software Defined Radio*. Available: http://nuand.com/

[94] A. Prabaswara, A. Munir, and A. B. Suksmono, "GNU Radio based software-defined FMCW radar for weather surveillance application," in *Telecommunication Systems, Services, and Applications (TSSA), 2011 6th International Conference on*, 2011, pp. 227-230.

[95] S. Heunis, Y. Paichard, and M. Inggs, "Passive radar using a software-defined radio platform and opensource software tools," in *Radar Conference (RADAR), 2011 IEEE*, 2011, pp. 879-884.

[96] C. Boon Leng, R. Palmer, Z. Yan, M. Yeary, and Y. Tian-You, "A software-defined radar platform for waveform design," in *Radar Conference (RADAR), 2012 IEEE*, 2012, pp. 0591-0595.

[97] A. L. Pazmany, J. B. Mead, H. B. Bluestein, J. C. Snyder, and J. B. Houser, "A Mobile Rapid-Scanning X-band Polarimetric (RaXPol) Doppler Radar System," *Journal of Atmospheric and Oceanic Technology,* vol. 30, pp. 1398-1413, 2013/07/01 2013.

[98] L. Chiou-Yng and P. K. Meher, "Efficient Subquadratic Space Complexity Architectures for Parallel MPB Single- and Double-Multiplications for All Trinomials Using Toeplitz Matrix-Vector Product Decomposition," *Circuits and Systems I: Regular Papers, IEEE Transactions on,* vol. 62, pp. 854-862, 2015.

[99] M. Yuelin, Y. Yamao, Y. Akaiwa, and Y. Chunlei, "FPGA Implementation of Adaptive Digital Predistorter With Fast Convergence Rate and Low Complexity for Multi-Channel Transmitters," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 61, pp. 3961-3973, 2013.

[100] N. Sufeng, W. Sizhou, S. Aslan, and J. Saniie, "Hardware and software design for QR Decomposition Recursive Least Square algorithm," in *Circuits and Systems (MWSCAS), 2013 IEEE 56th International Midwest Symposium on*, 2013, pp. 117-120.

[101] V. Mahalingam, K. Bhattacharya, N. Ranganathan, H. Chakravarthula, R. R. Murphy, and K. S. Pratt, "A VLSI Architecture and Algorithm for Lucas–Kanade-Based Optical Flow Computation," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, pp. 29-38, 2010.

[102] A. Abba, A. Manenti, A. Suardi, A. Geraci, and G. Ripamonti, "Non-linear least squares fitting in FPGA devices for digital spectroscopy," in *Nuclear Science Symposium Conference Record (NSS/MIC), 2009 IEEE*, 2009, pp. 563-568.

[103] L. Boher, R. Rabineau, and M. Helard, "FPGA Implementation of an Iterative Receiver for MIMO-OFDM Systems," *Selected Areas in Communications, IEEE Journal on*, vol. 26, pp. 857-866, 2008.

[104] I. LaRoche and S. Roy, "An efficient regular matrix inversion circuit architecture for MIMO processing," in *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, 2006, p. 4 pp.

[105] C. Dick, F. Harris, M. Pajic, and D. Vuletic, "Real-Time QRD-Based Beamforming on an FPGA Platform," in *Signals, Systems and Computers, 2006. ACSSC '06. Fortieth Asilomar Conference on*, 2006, pp. 1200-1204.

[106] A. Nakajima, K. Minseok, and H. Arai, "FPGA implementation of MMSE adaptive array antenna using RLS algorithm," in *Antennas and Propagation Society International Symposium, 2005 IEEE*, 2005, pp. 303-306 vol. 3A.

[107] T. Lok-Kee, R. Woods, and C. F. N. Cowan, "Virtex FPGA implementation of a pipelined adaptive LMS predictor for electronic support measures receivers," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 13, pp. 86-95, 2005.

[108] E. N. Frantzeskakis and K. J. R. Liu, "A class of square root and division free algorithms and architectures for QRD-based adaptive signal processing," *Signal Processing, IEEE Transactions on*, vol. 42, pp. 2455-2469, 1994.

[109] Y. Zhu Liang, S. Wee, and S. Rahadja, "QR-RLS Based Minimum Variance Distortionless Responses Beamformer," in *Acoustics, Speech and Signal*

Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on, 2006, pp. III-III.

[110] P. S. R. Diniz, *Adaptive Filtering*, 2nd ed.: Boston : Kluwer Academic Publishers, 1997.

[111] S. Haykin, *Adaptive Filter Theory*, 3rd ed.: Prentice-Hall, 1996.

[112] N. R. Shanbhag and K. K. Parhi, *Pipelined adaptive digital filters*. Boston: Kluwer Academic Publishers, 1994.

[113] B. Widrow and S. D. Stearns, *Adaptive signal processing*. Englewood Cliffs, N.J.: Prentice-Hall, 1985.

[114] L. Guoz-hu, L. Fuyun, and J. G. Proakis, "The LMS algorithm with delayed coefficient adaptation," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, pp. 1397-1405, 1989.

[115] T. Kumura, M. Ikekawa, M. Yosbida, and I. Kuroda, "VLIW DSP for mobile applications," *Signal Processing Magazine, IEEE*, vol. 19, pp. 10-21, 2002.

[116] N. Matsumoto, K. Ichige, and H. Arai, "Fixed-point digital processing of recursive least-square algorithm toward FPGA implementation of MMSE adaptive array antenna," in *Signal Processing and Its Applications, 2003. Proceedings. Seventh International Symposium on*, 2003, pp. 615-617 vol.2.

[117] G. Jian-Feng, S. C. Chan, Z. Wei-Ping, and M. N. S. Swamy, "Joint DOA Estimation and Source Signal Tracking With Kalman Filtering and Regularized QRD RLS Algorithm," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 60, pp. 46-50, 2013.

[118] M. Shoaib, S. Werner, and J. A. Apolinario, "Multichannel Fast QR-Decomposition Algorithms: Weight Extraction Method and Its Applications," *Signal Processing, IEEE Transactions on*, vol. 58, pp. 175-188, 2010.

[119] A. Elnashar, S. Elnoubi, and H. A. El-Mikati, "Performance Analysis of Blind Adaptive MOE Multiuser Receivers Using Inverse QRD-RLS Algorithm," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 55, pp. 398-411, 2008.

[120] S. D. Muruganathan and A. B. Sesay, "A QRD-RLS-Based Predistortion Scheme for High-Power Amplifier Linearization," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 53, pp. 1108-1112, 2006.

[121] G. Rombouts and M. Moonen, "Fast QRD-lattice-based unconstrained optimal filtering for acoustic noise reduction," *Speech and Audio Processing, IEEE Transactions on*, vol. 13, pp. 1130-1143, 2005.

[122] C. Shiunn-Jang and C. Chung-Yao, "Adaptive linearly constrained inverse QRD-RLS beamforming algorithm for moving jammers suppression," *Antennas and Propagation, IEEE Transactions on,* vol. 50, pp. 1138-1150, 2002.

[123] M. Jun, K. K. Parhi, and E. F. Deprettere, "Annihilation-reordering look-ahead pipelined CORDIC-based RLS adaptive filters and their application to adaptive beamforming," *Signal Processing, IEEE Transactions on,* vol. 48, pp. 2414-2431, 2000.

[124] L. Gao and K. K. Parhi, "Hierarchical pipelining and folding of QRD-RLS adaptive filters and its application to digital beamforming," *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on,* vol. 47, pp. 1503-1519, 2000.

[125] K. J. Raghunath and K. K. Parhi, "Finite-precision error analysis of QRD-RLS and STAR-RLS adaptive filters," *Signal Processing, IEEE Transactions on,* vol. 45, pp. 1193-1209, 1997.

[126] S. F. Hsieh, K. J. R. Liu, and K. Yao, "A unified square-root-free approach for QRD-based recursive-least-squares estimation," *Signal Processing, IEEE Transactions on,* vol. 41, pp. 1405-1409, 1993.

[127] K. R. Liu, S. F. Hsieh, K. Yao, and C. T. Chiu, "Dynamic range, stability, and fault-tolerant capability of finite-precision RLS systolic array based on Givens rotations," *Circuits and Systems, IEEE Transactions on,* vol. 38, pp. 625-636, 1991.

[128] A. Rosado, T. Iakymchuk, M. Bataller, and M. Wegrzyn, "Hardware-efficient matrix inversion algorithm for complex adaptive systems," in *Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on*, 2012, pp. 41-44.

[129] S. H. Ardalan, "Floating-point error analysis of recursive least-squares and least-mean-squares adaptive filters," *Circuits and Systems, IEEE Transactions on,* vol. 33, pp. 1192-1208, 1986.

[130] B. W. Robinson, D. Hernandez-Garduno, and M. Saquib, "Fixed and Floating-Point Implementations of Linear Adaptive Techniques for Predicting Physiological Hand Tremor in Microsurgery," *Selected Topics in Signal Processing, IEEE Journal of,* vol. 4, pp. 659-667, 2010.

[131] J. G. McWhirter and I. K. Proudler, "A systolic array for recursive least squares estimation by inverse updates," in *Control, 1994. Control '94. International Conference on*, 1994, pp. 1272-1277 vol.2.

[132] M. Moonen and J. G. McWhirter, "Systolic array for recursive least squares by inverse updating," *Electronics Letters,* vol. 29, pp. 1217-1218, 1993.

[133] C. M. Rader, "VLSI systolic arrays for adaptive nulling [radar]," *Signal Processing Magazine, IEEE*, vol. 13, pp. 29-49, 1996.

[134] Y. V. Zakharov, G. P. White, and L. Jie, "Low-Complexity RLS Algorithms Using Dichotomous Coordinate Descent Iterations," *Signal Processing, IEEE Transactions on*, vol. 56, pp. 3150-3161, 2008.

[135] L. Li, G. Heymsfield, J. Carswell, D. H. Schaubert, M. L. McLinden, J. Creticos, *et al.*, "The NASA High-Altitude Imaging Wind and Rain Airborne Profiler," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. PP, pp. 1-13, 2015.

[136] S. Wang, "Waveform and transceiver optimization for multi-functional airborne radar through adaptive processin," University of Oklahoma, 2013.

[137] Xilinx, "UltraScale Architecture and Product Overview," in *DS890(v2.3)*, ed, 2015.

[138] Xilinx. (2015, Oct. 15). *Zynq UltraScale+ MPSoC Devices*. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc/silicon-devices.html

[139] ITRS. (2013). *International Technology Roadmap for Semiconductors - Executive Summary*. Available: http://www.itrs.net/

[140] Xilinx. (2015, October 5). *Xilinx Collaborates with TSMC on 7nm for Fourth Consecutive Generation of All Programmable Technology Leadership and Multi-node Scaling Advantage*. Available: http://press.xilinx.com/2015-05-28-Xilinx-Collaborates-with-TSMC-on-7nm-for-Fourth-Consecutive-Generation-of-All-Programmable-Technology-Leadership-and-Multi-node-Scaling-Advantage

[141] JEDEC. (2015, Aug. 18). Available: http://www.jedec.org/

[142] H. M. C. Consortium, "Hybrid Memory Cube Specification 2.0," ed, 2014.

[143] RapidIO. (2015, Aug. 18). Available: http://www.rapidio.org/

[144] InfiniBand. (2015, Aug. 18). Available: http://www.infinibandta.org/

[145] Xilinx, "Serial RapidIO Gen2 Endpoint v3.3," ed, 2015.

[146] S. H. Fuller and A. Gatherer, *RapidIO : the embedded system interconnect*. Chichester, England ; Hoboken, NJ: Wiley, 2005.

[147] I. D. Technology. (2015, Aug. 15). *IDT Serial RapidIO System Modeling Tool software*. Available: http://www.idt.com/products/interface-connectivity/serial-rapidio-solutions/serial-rapidio-tools/SRIOGen2-ModelingTool

Appendix - List Of Acronyms and Abbreviations

APC	Adaptive Pulse Compression
AXI	Advanced eXtensible Interface
AMBA	Advanced Microcontroller Bus Architecture
AMPP	Altera Megafunctions Partner Program
AF	Ambiguity Function
ADC	Analog-to-Digital Converter
APU	Application Processing Unit
ASIC	Application-Specific Integrated Circuit
BW	Bandwidth
BRAM	Block Random Access Memory
CSA	Carry-Save Adder
CPU	Central Processing Unit
CPI	Coherent Processing Interval
CMOS	Complementary Metal-Oxide Semiconductor
CLB	Configurable Logic Block
C-SWaP	Cost, Size, Weight and Power
DCR	Device Control Register
DSP	Digital Signal Processor
DSO	Digital Storage Oscilloscope
DAC	Digital-to-Analog Converter
DMA	Direct Memory Access
EPLD	Erasable Programmable Logic Device
EPROM	Erasable Programmable Read-Only Memory
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array

FPLD	Field-Programmable Logic Device
FLOPs	Floating-Point Operations
FLOPS	Floating-Point Operations per Second
FPU	Floating-Point Unit
GPP	General Purpose Processor
GPU	Graphics Processing Unit
HDL	Hardware Description Language
IQ	In-phase Quadrature
I/O	Input-Output
IC	Integrated Circuit
IP	Intellectual Property
IF	Intermediate Frequency
IFFT	Inverse Fast Fourier Transform
LS	Least Square
LFM	Linear Frequency Modulation
LE	Logic Element
LUT	Look Up Table
LUT	LookUp Table
MHz	Mega Hertz
MMU	Memory Management Unit
MMSE	Minimum Mean Squared Error
MPAR	Multifunction Phased Array Radar
MAC	Multiply Accumulate
NEXRAD	Next Generation Weather Radar
NLFM	Non-Linear Frequency Modulation
OPB	On-chip Processor Bus
OS	Operating System
PLL	Phase-Locked Loop
PR	Precipitation Radar
PS	Processing System
PLB	Processor Local Bus

PL	Programmable Logic
PLD	Programmable Logic Device
PRF	Pulse Repetition Frequency
QRD-RLS	QR-Decomposition Recursive Least-Squares
RF	Radio Frequency
RISC	Reduced Instruction Set Computing
RTL	Register-Transfer Level
RMMSE	Reiterative Minimum Mean Squared Error
RCA	Ripple-Carry Adder
SRIO	Serial RapidIO
SPE	Signal Processing Engine
SNR	Signal to Noise Ratio
SIMD	Single Instruction Multiple Data
SVD	Singular Value Decomposition
SWaP	Size, Weight and Power
SCU	Snoop Control Unit
SDR	Software Defined Radio
STAP	Space-Time Adaptive Processing
SAW	Surface Acoustic Wave
SDRAM	Synchronous Dynamic Random Access Memory
SAR	Synthetic Aperture Radar
SoC	System-on-Chip
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver/Transmitter
USRP	Universal Software Radio Peripheral
UAV	Unmanned Aerial Vehicle
VLIW	Very Long Instruction Word
VLSI	Very-Large-Scale Integration
VHDL	VHSIC Hardware Description Language
XPE	Xilinx Power Estimator