A FPGA Based Low-cost High Speed

QRD-RLS Array Processing

A DISSERTATION SUBMITTED TO

THE CENTRE FOR INTELLIGENT DYNAMIC COMMUNICATIONS,

DEPARTMENT OF ELECTRONIC AND ELECTRICAL ENGINEERING,

AND THE COMMITTEE FOR POSTGRADUATE STUDIES

OF THE UNIVERSITY OF STRATHCLYDE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

By

Qiang Gao

May 2012

Declaration

I declare that this thesis is the result of my original research. It has been composed by myself and has not been previously submitted for examination which has led to the award of a degree

Qiang Gao

Abstract

Over the last 30 years, Digital Signal Processing algorithm implementation has been driven by the continued progress and availability of high speed FPGA/ASIC circuit technology. The classic method of CORDIC (Coordinate Rotation DIgital Computer) arithmetic has been widely implemented as part of the computational requirements of the well known QR decomposition - Recursive Least Squares (RLS) algorithm.

This thesis presents a novel FPGA implementation of a complex arithmetic valued QR decomposition which can function in adaptive filter applications and implement the adaptive filter weight extraction without using the traditional back-substitution method. In order to operate Givens rotation on a complex valued system, Double Angle Complex Rotation (DACR) is adopted to simplify the computational requirement of the classic Complex Givens Rotation (CGR). A new modified 'processor-like' architecture of a DACR based QR-RLS is presented. It features a single arithmetic Processing Element (PE) that has been extensively pipelined and efficiently shared for the implementation of the Givens transform associated with the QR algorithm.

Annihilation-Reordering look-ahead Transformation (ART) was used to pipeline and hence speed up the real valued QR-RLS adaptive signal processing array without changing the filter's convergence behavior. This thesis extends the ART technique to cope with a complex valued QR-RLS computation array, results in a novel Complex valued Annihilation-Reordering look-ahead Transformation (C-ART). The complex Givens rotation implemented in CORDIC arithmetic is suitable for the cut-set pipelining thus increasing the throughput.

Acknowledgements

I will take this oppotunity to thank my supervisor Professor Robert. W. Stewart for all the help throughout the years studying for my Ph.D. and for guiding and supporting me in my research work, and also for giving me the valuable insights into both the academia and industry.

A special thanks goes to Dr Louise Crockett. With her help, I had the oppotunity to experience many latest FPGA hardware and EDA design tools. We also had many fruitful collaborations on the development of several interesting industrial trainings in FPGA design.

Special thanks also goes to my colleague and good friend Ousman Sadiq for all the discussions over the years concerning adaptive DSP theory, beamforming and some more “crazy” ideas. I am also grateful to Andrew Kenyon and Rahul Summan, together with Ousman, thanks for all the funny discussions during our weekly lunch session.

I would like to express my appreciation to my colleagues and friends at DSP enabled Communications (DSPeC) group in University of Strathclyde with whom I have had many interesting and fruitful discussions, and who have also made my time at the department more enjoyable: Dr Faisal Darbari, Dr Konstantinos Mammasis, Ke He, Yousif Awad, Ross Elliot, Martin Enderwitz, Michael Hanson, Colin McGuire and Philipp Karagiannakis - thanks to each of you, it is a pleasure having you around.

Since my research work was spun off in Steepest Ascent Ltd., I made a detour into industry. The knowledge learned from taking a research project into a commercial product gave me many valuable experiences. I would like to give my thanks to the colleagues in Steepest Ascent whom we participated together in the R&D work: Dr Garrey Rice, Jamie Bowman, Dr Neil MacEwen, John O'Sullivan, Gregour Bolton, Adnan Ghauri etc. Thanks for the fantastic work we did together.

Also I want to show my gratitude to the Xilinx University Program (XUP) for financially supporting my research in design and implementation of DSP systems on the high performance devices from this world leading FPGA vendor.

Finally I would like to thank my family. My parents have been a great source of help and inspiration over the years. My father, who used to be an industrial automation engineer and currently expends his business in telecommunication industry, sparked my interest in electronics particular in the telecommunication applications. I would also like to express my appreciation to my wife Jingyu for her love, encouragment and support. The last few years have been incredibly remarkable thanks to her. We have develped ourselves as the explorers in many corners of the world and always had a new dream destination to discover alongside each other.

Symbols and Acronyms

Symbols

$x(\ )$	Input signal of adaptive filter
$y(\ )$	Output signal of adaptive filter
$d(\ )$	Desired signal of adaptive filter
$w(\ )$	Weights of adaptive filter
N	Number of weights $w(\ )$
$Re(\ )$	Real part of a complex number
$Im(\ )$	Imaginary part of a complex number
$O[\ ]$	Computational complexity
λ	Forgetting factor
$\boldsymbol{Q}$	Unitary matrix obtained through QR decomposition
$\boldsymbol{R}$	Upper triangular matrix obtained through QR decomposition
x_i	x component of CORDIC arithmetic
y_i	y component of CORDIC arithmetic
z_i	z component of CORDIC arithmetic
d_i	Decision factor of CORDIC arithmetic
K_i	Scaling factor of CORDIC arithmetic
n	CORDIC iteration number
$\boldsymbol{G}$	Givens rotation
c	Cosine
s	Sine
S	Scaling factor compensation of CORDIC (i.e. K_n^{-1})
ϕ	1st angle of DACR

θ	2nd angle of DACR
D	1 cycle delay
f_c	Clock rate
f	Delay caused by scaling compensation
m	Delay caused by single MAC unit
S_1	CORDIC Approximation-Scaling-Compensation
S_2	Correction of CORDIC Approximation-Scaling-Compensation
M	Look-ahead factor

Acronyms

DSP	Digital Signal Processing
SoC	System on Chip
GPU	Graphics Processing Unit
ASIC	Application Specific Integrated Circuit
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
VLIW	Very Long Instruction Word
SIMD	Single Instruction Multiple Data
CUDA	Compute Unified Device Architecture
OpenCL	Open Compute Layer
NRE	Non Recurring Engineering
IOB	I/O Block
CLB	Configurable Logic Block
LUT	Look Up Table
FF	Flip Flop
MAC	Multiplication and ACcumulation
LMS	Least Mean Squares

RLS	Recursive Least Squares
IIR	Infinite Impulse Response
FIR	Finite Impulse Response
SFG	Signal Flow Graph
BC	Boundary Cell
IC	Internal Cell
DC	Downdating Cell
EC	Extraction Cell
PE	Processing Element
IQRD-RLS	Inverse QRD-RLS
CORDIC	COordinate Rotation DIgital Computer
OQE	Overall Quantisation Error
SGR	Squared Givens Rotation
CSD	Canonical Signed Digit
MAG	Minimised Adder Graph
RSG	Reduced Slice Graph
BS	Barrel Shifter
AR	Angle Recording
MSR-CORDIC	Mixed Scaling Rotation CORDIC
AE	Approximation Error
RE	Rounding Error
CGR	Complex Givens Rotation
DACR	Double Angle Complex Rotation
ART	Annihilation-Reordering look-ahead Transformation
MVDR	Minimum Variance Distortionless Response
VLSI	Very Large Scale Integration

C-ART	Complex valued Annihilation-Reordering look-ahead Transformation

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Adaptive Digital Signal Processing

In recent decades, the field of Digital Signal Processing (DSP) has developed rapidly due to the increased availability of technology for the implementation of adaptive DSP algorithms for the next generation wireless communications, like the Least Mean Squares (LMS), Recursive Least Squares (RLS) and QR Decomposition (QRD) - RLS algorithm etc. In contrast to the ‘fixed coefficients’ FIR filter which is designed for applications where the required coefficients do not change over time, many real-world signal processing systems like the communication, radar and sonar require the ‘optimal’ coefficients, which adjust over time and depend on the input signal in order to minimize the error function. This error function, also known as the cost function, is a distance measurement between the desired and output signals of the filter. The adaptive filtering algorithms is to converge a optimum solution which is usually given by a reduction on the error distance.

These adaptive DSP algorithms have been applied to an extensive number of problems including adaptive beamforming, channel equalization, precoding, channel estimation as well as many others. The bandwidth and speed requirements of the communication systems have been increased over the years, hence the high speed DSP algorithm implementation should be provided to cope with these requirements.

1.2 Adaptive DSP on Field Programmable Gate Arrays

The Field Programmable Gate Array (FPGA) has become an important hardware implementation platform for the next generation wireless communication systems, due to the advantage of implementing the DSP systems on the parallel hardware structures, in place of the traditional throughput limited DSP microprocessors. Also FPGAs have more reconfigurable flexibility and can provide the benefits of full custom ASIC while avoiding the initial cost, time delay and risks. Although the FPGA device size increases, there is still a requirement to design the area efficient hardware systems to reduce the cost and power consumption.

DSP systems contain an abundance of mathematical operations from simple addition, substraction and multiplication to further complex linear algebra arithmetic which are all highly suitable for the parallel implementation on FPGAs. The adaptive DSP filtering is one of the highest profile DSP systems and the focus of this research is the area and throughput efficent implementation of the QR Decomposition - Recursive Least Squares (QRD-RLS) filtering.

1.3 Contributions

This thesis investigates a range of methdologies to efficiently implement the QRD-RLS algorithm on FPGAs. The main contributions of this research are summarized as follows:

The Pipeline-Interleaving Coarse Angle CORDIC for QR-RLS Array Processing

The classic method of CORDIC (COordinate Rotation Digital Computer) arithmetic has been widely implemented as part of the computational requirements of QR-RLS algorithm, which plays an important role in the physical layer of wireless communications, such as the adaptive equalization and beamforming etc. The CORDIC based QR-RLS systolic array is analysed in terms of the hardware cost and throughput performance. Despite the fully pipelined CORDIC inside the QR updating loop results in a shortened critical path, it cannot increase the throughput.

However, it is still possible to increase the throughput of the QR recursive loop by pipelining the CORDIC. This involves the pipeline-interleaving scheme where the same pipelined feedback loop is shared with the several seperated data channels.

A novel coarse angle CORDIC is presented to implement both the real and complex valued pipeline-interleaving QR-RLS architectures. The CORDIC architecture features an easily pipelinable single Processing Element (PE) and a Multiply and ACcumulation (MAC) unit (names 'single PE + MAC' architecture in this thesis), and can be used to implement both the Givens generations and Givens rotations associated with the QR update. The implementaton result on a Xilinx Virtex 4 FPGA suggests that the pipeline-interleaving architecture not only produces a high throughput, but also results in a more regular and lower cost structure than the existing benchmarks using the back-substitution.

CORDIC Approximate-Scaling-Compensation Architecture for QR-RLS

Many communication applications require complex valued DSP arithemetic. In order to operate the complex Givens rotation in the QR-RLS filtering, Double Angle Complex Rotation (DACR) is used to simplify the computational requirement of the conventional Complex Givens Rotation (CGR). An optimised CORDIC 'Quadrant-Correction-Free' mapping scheme is developed to further simplify the quandrant mapping and correction procedures required by the DACR implementation.

The DACR is then employed in the pipeline-interleaved QR-RLS architecture for achieving the low cost complex valued QR-RLS implementation and the high throughput. The throughput could be further improved by a proposed CORDIC Approximate-Scaling-Compensation algorithm.

An analytical comparison is made between a 'Single PE + MAC' QR-RLS processors with the standard CORDIC scaling compensation, and a design employing the Approximate-Scaling-Compensation scheme. The FPGA implementation result demonstrates that the CORDIC Approximate-Scaling-Compensation structure results in a better throughput performance.

Complex valued Annihilation-Reordering look-ahead Transformation (C-ART) QR-RLS Architecture

Besides the acceleration of QR recursive loop by pipeline-interleaving scheme, another method - the look-ahead transformation - is also investigated in this research to pipeline the recursive feedback loop without interleaving the multiple data channels.

Previously the Annihilation-Reordering look-ahead Transformation (ART) was presented to successfully speed up the real valued QR-RLS systolic array without degrading the filter convergence behavior. Similar to the traditional mult-add look-ahead, annihilation-reordering look-ahead transforms a sequential recursive algorithm to an equivalent concurrent one by creating the additional parallelism in the algorithm.

However, many communication applications require the complex valued DSP arithmetic and the aformentioned DACR is employed to optimisely realize the complex Givens rotation. Hence this thesis extends the ART to cope with the complex valued QR-RLS array results in a novel Complex valued Annihilation-Reordering Look-ahead Transformation (C-ART).

The C-ART is successfully applied to increase the throughput of the complex Givens rotation based QRD-RLS systolic array. Computer verifications in the adaptive equalization and beamforming environments prove that the new technology can achieve the same ensemble-average learning curve and beam pattern with those of the classic sequential updated QR-RLS algorithm. Finally, the methodology of fine-grain level mapping is presented, and the FPGA implementation of a single processing element confirms the improved throughput property of the C-ART structure.

1.4 Thesis Structure

The background information on the existing hardware platforms for implementing the complex DSP algorithms are discussed in **Chapter 2**. FPGA platform is investigated in more depth. The FPGA embedded blocks which closely tie to the DSP performance of this research, including the DSP48 macro, fast carry chain and shift register (SRL16), are demonstrated. These DSP-enabled components provide the design convenience and performance enhancement for the work in this thesis.

Chapter 3 discusses the main features of QR-RLS, including the algorithm which is derived from the conventional Recursive Least Squares (RLS) filtering, and its corresponding systolic array for hardware implementation. The classic back-substitution scheme for the serial extraction of weights is briefly reviewed. The extended QR-RLS algorithm/architecture is derived for the parallel extraction of weights, and in the last part of this chapter, the mapping strategies are discussed for optimisely targetting the QR-RLS systolic array on FPGAs.

In **Chapter 4**, the CORDIC arithmetic which is used for realizing the Givens rotation in QR-RLS systolic array is presented. The ‘speed-area’ analysis is carried to three well known CORDIC architectures. Related works on the CORDIC speed-up solution are briefly reviewed.

Chapter 5 concerns the optimised mapping methodology of QR-RLS array, and considers how the architecture with the low resource usage can be mitigated in a speed optimised manner. This chapter includes the development of a pipeline-interleaving coarse angle CORDIC, and the comparison of cost, speed and numerical performance between it and the conventional CORDICs.

The emphasis moves towards to the complex valued QR-RLS in **Chapter 6**, where the impact of employing the pipeline-interleaving scheme to complex valued QR-RLS array is evaluated. A novel acceleration scheme - ‘Scaling-Partial-Compensation’ is presented to further increase the throughput of the QR recursive loop.

In **Chapter 7**, a novel Complex valued Annihilation-Reordering Look-ahead Transformation (C-ART) is developed to further increase the throughput of the complex Givens rotation based QRD-RLS systolic array. The methodology of fine-grain level mapping of the resulted QR-RLS array is presented to mitigate this speed enhanced scheme in an area-efficient achitecture.

Finally, the thesis is concluded in **Chapter 8**, and suggestions for the future work are presented.

Chapter 2

Hardware Platforms for DSP Implementation

2.1 Introduction

The technology used for DSP implementation is very closely linked with the developments in the semiconductor technology. The decrease in transistor size has been the major driving force in creating and developing a new market for DSP technologies, particularly mobile communication and digital video industry. The improved silicon technology has not only offered a lower cost platform of implementation, but also allowed the DSP systems to achieve higher throughputs and lower power consumption.

A number of technologies have emerged, these range from the microcontroller which operates on the moderate sampling rate, to the embedded system-on-chip (SoC) microprocessor that targets high performance DSP applications. This processor style architecture has been exploited in various forms including the single to multi-core dedicated DSP microprocessors where the hardware has been included to allow specific DSP functionality to be realized efficiently [1], Graphics Processing Units (GPUs) [2], and also the variety of parallel machines [3] that have been developed for the specific application domains.

When discussing a DSP implementation, ultilised hardware resources and the interconnection between the digital logics play a major part in the performance (area, speed and power consumption). Comparing with the programmable processors which has a fixed type of architecture, Application-Specific Integrated Circuits (ASICs) and Field Programmable Gate Arrays (FPGAs) both provide the designers with the maximum freedom to create the hardware which can best match the requirements of algorithms [4]. Hence it is interesting to compare the above platform technologies for

DSP implementation.

2.2 DSP Microprocessors

The sequential nature of the conventional general-purpose processors (GPPs) architecture, such as Von Neumann architecture [3], makes it unsuitable for the efficient implementation of computationally complex DSP systems, either because it cannot achieve the desired sampling rate, or it meets the speed requirement but consumes too much power. A large number of the transistors on chip are not involved in this sequential computation process, thus they consume unnecessary power.

This spurs the motivation to look at other types of processor architectures to perform DSP. Although fundamentally related, the DSP processors are based on the Harvard architecture [3] are significantly different with GPPs. In contrast with the Von Neumann architecture which uses one memory for both the code and data, the Harvard architecture seperates the data memory and the program memory, allowing the program to be loaded into the processor independently from the data. A number of modifications have been made since the early DSP microprocessors, including pipelining, increased number of data buses, and Very Long Instruction Word (VLIW) [3]. VLIW increases the wordlength of internal bus and allows a number of operations performed by each instruction in parallel. Last but not least, some DSP processors also offer fixed point processing units along with the traditional floating point ones, to provide the throughput improved solution for many DSP systems which do not require the full precision arithmetic [3].

2.3 Graphics Processing Unit (GPU)

Graphics Processing Units (GPUs) [2] are massive parallel Single Instruction Multiple Data (SIMD) [3] - like processors. The current high end GPUs is capable to offer 500 GFLOP/s peak performance in single precision. As commodity accelerators for 3D graphics, GPUs offer tremendous computational performance at relatively low costs. GPUs are favourable in applications with high inherent parallelism and requirement of coarse-grain synchronisation between processors.

Simultaneously, GPU execution models have grown in flexibility in response to the needs of graphics programmers thereby enabling a wide range of computing tasks. GPU vendors have consequently developed graphics-unrelated programming models such as NVIDIA's Compute Unified Device Architecture (CUDA) [2] and Open Compute Layer (OpenCL) [5] to perform the general purpose computing on GPUs. The CUDA architecture includes several new components which are designed strictly for GPU computing and aimed to alleviate many of the limitations that prevented previous graphics processor from being useful for general purpose computing [2]. OpenCL is a framework for writing programme that execute across heterogeneous platforms including CPUs, GPUs etc [5].

The large register files and high-performance scratchpad memories on GPU are well-suited to be utilized as accelerators to speedup the algorithms with very high arithmetic intensity, like the matrix factorisation algorithms including Cholesky, LU, and QR decomposition [6][7][8]. Accelerating these matrix factorization algorithms is to partition a large matrix into blocks. Massively parallel tasks like matrix multiplication and inversion etc are performed on GPUs and other serial tasks such as triangular solver are executed on CPUs.

2.4 Dedicated ASIC and FPGA Solutions

Embedded processor technologies are in the form of pre-defined architectures. The major attraction of dedicated ASIC offerings (which largely apply to FPGA realizations) is that the architecture can be developed to allow the level of parallelism to be created to specifically match the performance requirements of algorithms [4].

When considering the programmability argument, ASIC solutions do not include additional hardware to provide programmability as additional levels of programmability cause difficulty in test and verification. Non-Recurring Engineering (NRE) costs of producing a number of ASIC prototypes is very high, particularly for using the latest transistor technology.

Field Programmable Gate Array (FPGA) is a reprogrammable integrated circuit which allows the fast prototyping and high design flexibility. FPGA solutions avoid the high NRE cost by giving the user a logical hardware that can be programmed. The reconfiguration property of FPGA allows an already implemented design to be configured if the mistakes have been made in the design process, or replaced with a completely different one. Like ASICs, FPGAs require the design of a suitable circuit architecture to best ultilise the underlying hardware.

Although the different technologies for implementing DSP systems have been presented and compared, the reality is that the modern DSP systems are collections of the previously mentioned platforms. Many companies are now offering heterogeneous DSP boards comprising embedded microprocessors, ASIC and FPGAs, giving the suitability of different platforms to different computational requirements. In this research, to create the highly efficient DSP designs, FPGA is chosen to implement the computational complex DSP systems in the most efficient manner. The following part of this chapter will focus on the FPGA technology for DSP implementation.

2.5 FPGA Technology for DSP

Today’s high-complexity FPGA devices are capable of housing an entire DSP system, with the cooperation of the embedded CPU such as PowerPC [9] and MicroBlaze [10] etc. The basics of current FPGAs will be demonstrated in this chapter as well as the features that target directly to the adaptive DSP systems in this PhD work. FPGAs are provided by a range of vendors, including Xilinx, Altera, Actel and Lattice. The digital logic blocks provided on fabric are different between the vendors. The logic blocks share many similarities, although might be different in size. The targeted implementation platform in this research is the Xilinx Viretx 4 FPGA.

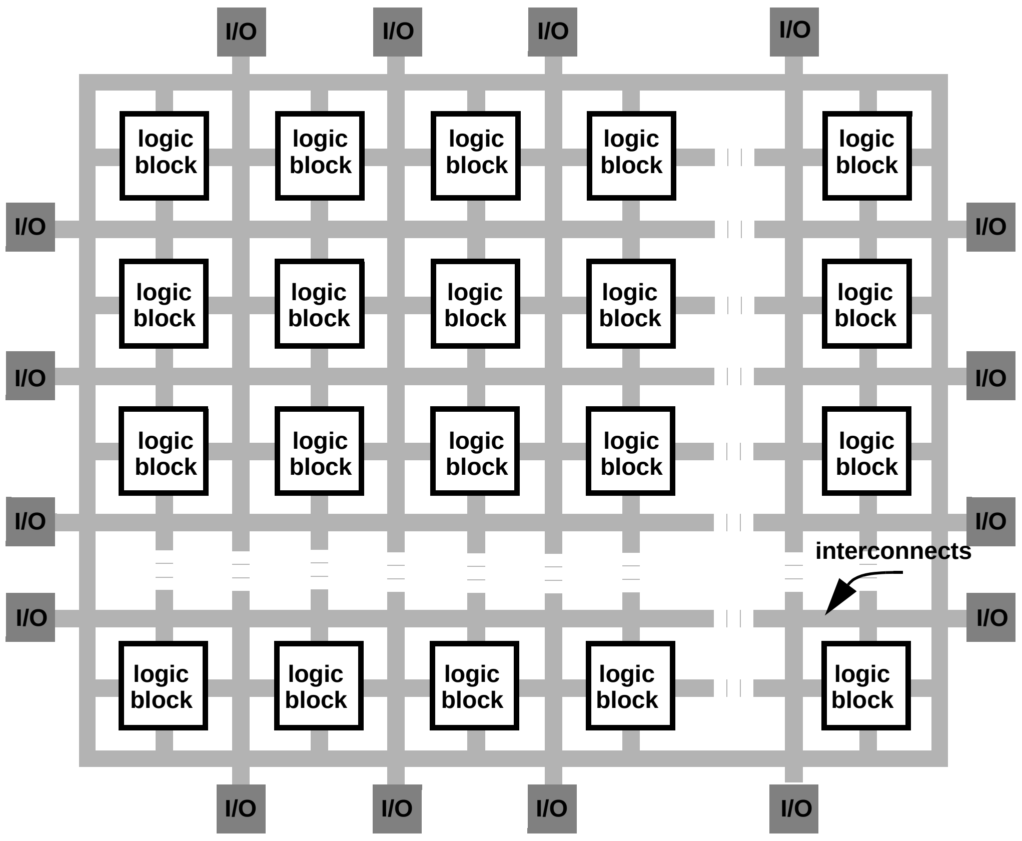


Figure: 2.1: Basic FPGA Architecture

2.5.1 Generic FPGA Architecture

The basic FPGA architecture follows the structure in Figure 2.1. The device is made up of the block memories, I/O Blocks (IOBs), digital logic blocks and programmable interconnections, the routing is omitted here for clarity. Programmable interconnects join the small user defined logic functions to form a large sequential and combinational digital design. Its reconfigurable flexibility suggests that the FPGA can be used to perform the DSP tasks in a range of communication applications, for example, a FPGA could be the basis for a system that performs one of several physical layer DSP functions.

The FPGA is usually a fully SRAM based device. As shown in Figure 2.1, the basic logic block in Xilinx terminology is called the Configurable Logic Block (CLB). A

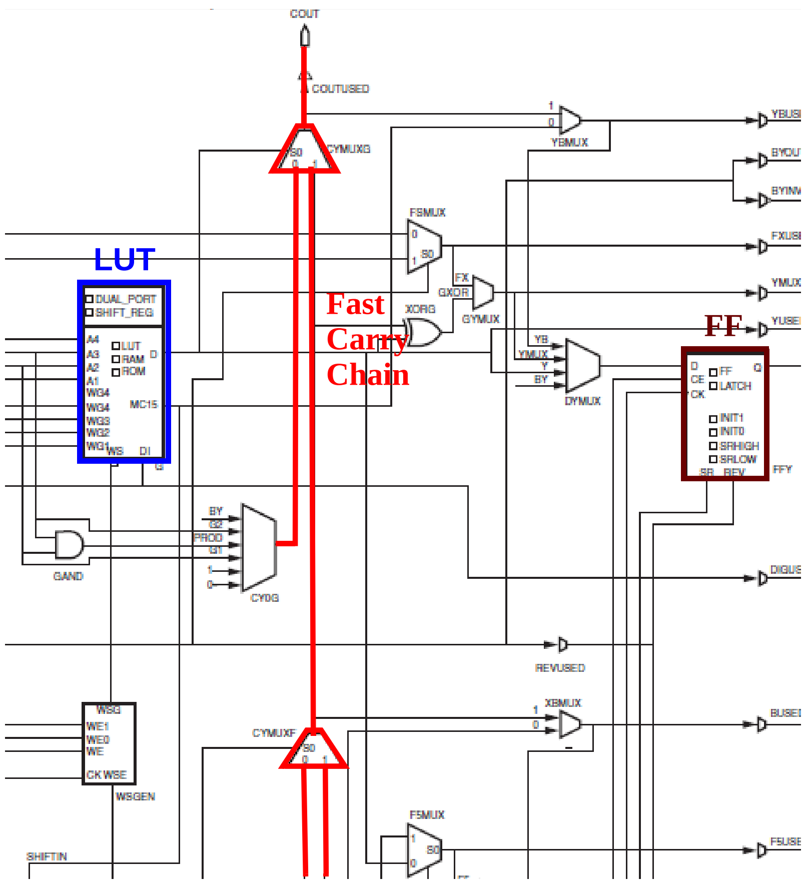


Figure: 2.2: Diagram of half Virtex 4 Slice [11]

single CLB on Xilinx Virtex-4 FPGA is composed of 4 Slices which is a Xilinx-specific term representing the smallest unit of FPGA logic area. The slices can be interconnected within the CLBs and the CLBs are linked together via the switch matrix box. The fine-grained level of half Slice is shown in Figure 2.2. Each Slice contains a carry chain, two 4-input lookup tables (LUTs), two flip-flops (FFs), two 2-input multiplexers (MUXes) names MUXF5 and MUXFX, and dedicated routing.

2.5.2 DSP Functionality

With the recent trend for high speed FPGA implementation of DSP algorithms, many FPGA manufarcturers include the dedicated DSP components on their chips to bring the design convenience and performance enhancement. These dedicated DSP features include the following:

• LUTs for Memory:

The general form of a LUT is a Static Random-Access Memory (SRAM) within the FPGA fabric, as shown in Figure 2.2. The LUTs are flexible to be used in several different ways, such as the dual port RAM, ROM and shift registers [11]. The LUT will be used in Chapter 5.6.

• Fast Carry Chain for Addition:

Addition is a very common operation in DSP. The full adders can be built on the look-up tables. The most general adder is the carry ripple adder [12]. However, the long carry ripple can harm the maximum clock rate of the adder, especially for the addition with the large wordlength. Hence it is necessary to include the fast carry chains into FPGAs to allow the carry bit propagating between each full adder as fast as possible. The presence of the fast carry logic in modern FPGA families, as shown in Figure 2.2, removes the need to develop the hardware intensive carry look-ahead schemes [12] by FPGA users. This feature will be discussed in Section 4.2.

• Flip-Flops for Pipelining:

In a digital circuit, the maximum propagation delay between two subsequent registers, which are formed by on-slice Flip-Flops (FFs), is known as 'critical path' [12]. The critical path decides the maximum clock rate the design can run at. After one clock edge occurs, the following edge cannot be allowed to occur until the signal with the longest delay has reached the destination Flip-Flop. The critical path includes: routing lines, LUTs, Multiplexers (MUXes) and carry-chains etc. For instance, the design with a cascade of MUXes in Figure 2.3 (*a*) has the critical path including the

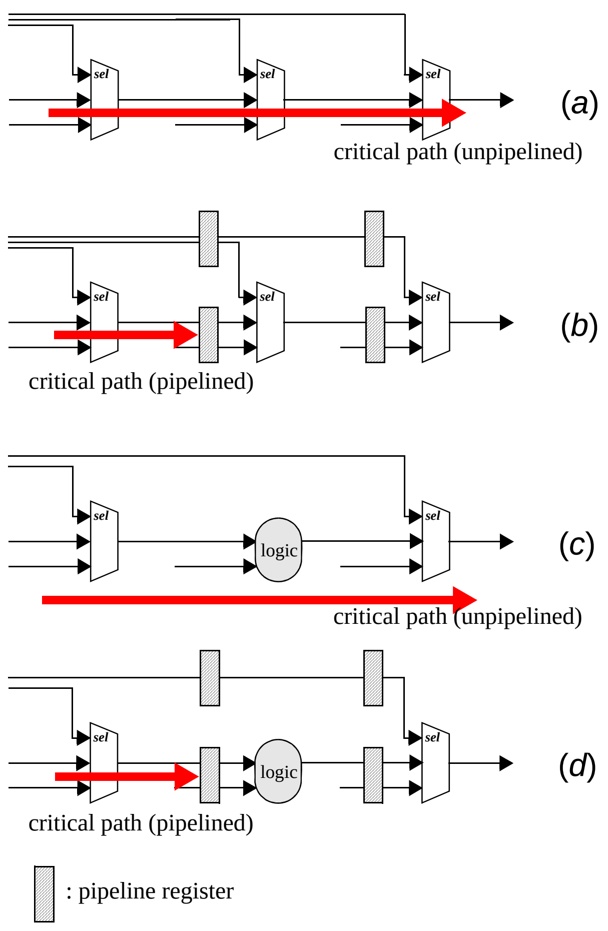

sel
(a)
critical path (unpipelined)
(b)
critical path (pipelined)
logic
(c)
critical path (unpipelined)
(d)
critical path (pipelined)
: pipeline register

Figure: 2.3: Cascade of multiplexers

propagation delays of all MUXes. After adding the pipeline registers to shorten the critical path, the new critical path is shown in Figure 2.3 (*b*). Similarly, the critical path in Figure 2.3 (*c*) is the total delays of all the MUXes and digital logic, the corresponding pipelined version is shown in Figure 2.3 (*d*).

To increase the clock rate of design, the critical path must be shortened. This feature will be discussed in Chapter 4.4.2 and Chapter 5.4. Closely related to the pipelining is the concept of retiming. Retiming a Signal Flow Graph (SFG) entails rearranging the registers/FFs to reduce the critical path. This feature will be covered in Chapter 7.2.

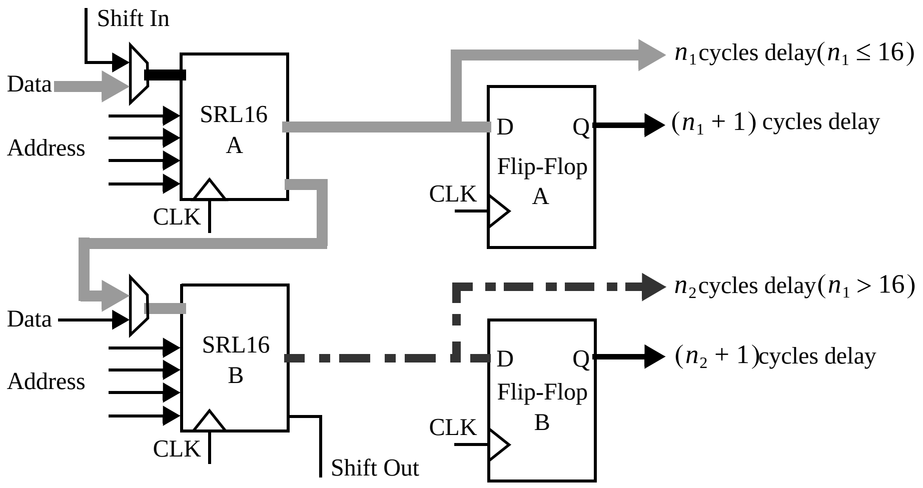


Figure: 2.4: Xilinx Shift Register - SRL16

• Shift-Registers for Delay Chain:

In discrete-time systems, the time delay is specified by an integer number of samples. On FPGAs, the delay lines are commonly implemented using FFs. However, FPGA also contains a special feature that allows each LUT to be configured as a 16-bit shift-register (known as SRL16) [13]. This technique will be used in Section 5.5 and Section 6.5. In Figure 2.4, if a delay line of less than 16 samples is required, the LUT input lines can be used as a 4-bit select address of SRL16 to specify the required length of delay, which is shown as SRL16 A. Seperated SRL16s can also be cascaded together to form a delay of over 16 samples, as the output data of SRL16 B. Connecting the output of SRL16 to an adjacent FF entails an additional one sample delay.

• DSP48 for Multiply-ACcumulation:

Multiplication and ACcumulation (MAC) is one of the most common operations in DSP design. The largest volume FPGA existing today offers several hundred dedicated DSP48 slices to optimise the MAC function [14]. A fully pipelined DSP48

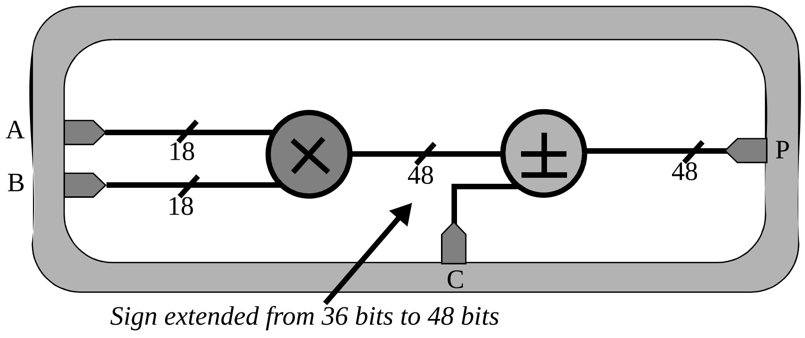


Figure: 2.5: DSP48 for MAC operation ($n_A \leq 18; n_B \leq 18; n_C \leq 48$)

can be clocked at up to 500MHz [14]. Figure 2.5 shows the diagram of DSP48 slice on Xilinx Virtex 4 FPGA, with one 18×18 bit multiplier and one 48 bits post adder. In Figure 2.5, when the bit-width of operand C is less than or equal to 48, the width n_A and n_B of operand A and B are both 18-bit or less, the MAC operation (A × B ± C) can be executed by a single DSP48 slice.

When $18 < n_A \leq 36$, $n_B \leq 18$ and $n_C \leq 48$, the multiplication operation A × B requires two 18×18 bit multipliers, namely two DSP48 slices, as shown in Figure 2.6. It is necessary to split wordlength n_A into two parts, as $n_A - 17$ and 17 bits. Both the most-significant and least-significant parts of operand A are multiplied by operand B. The resultant two partial products need to sum together, with the 17 bits right shift on the least-significant one.

This research involves the DSP48 slices in two places. The first one is using DSP48s to build a rolled CORDIC unit, this will be demonstrated in Section 4.4.3. The second application uses DSP48s for QR-RLS extraction cell (EC) which requires the Multiply-ACcumulation (MAC) operations. This point will be discussed in Section 5.5 and Section 6.5.

2.6 Conclusion

This chapter has highlighted a variety of different technologies used for

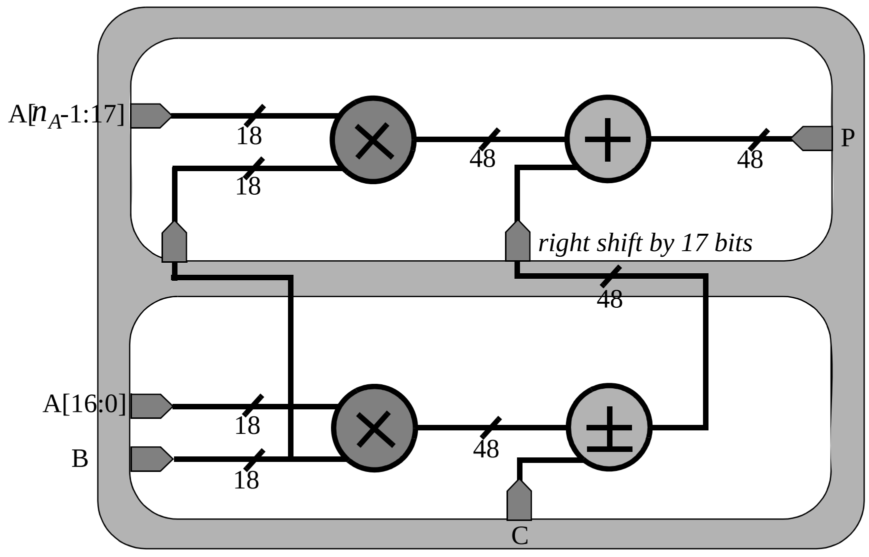


Figure: 2.6: DSP48 for MAC operation ($18 < n_A \le 36; n_B \le 18; n_C \le 48$)

implementing DSP systems. FPGA's parallel architecture and programmability are the dual factors that makes it very attractive for the implementation of many complex DSP systems. Some key DSP-enabled components, which are embedded on FPGA, are also illustrated in this chapter.

Chapter 3

Recursive Least Squares (RLS) using QR Method

3.1 Introduction

The Least Mean Squares (LMS) adaptive filtering algorithm [15][16] has been widely used for more than 40 years. It is well known that the LMS is an approximation to the least squares solution for adaptive filtering. The LMS algorithm has been one of the most commonly used adaptive algorithms due to its minimal structure, stable performance and relatively low computational requirements. Such advantages have made it the algorithm of choice for applications such as echo control and wireline channel equalisation such as the DSL modems [17], because for the wireline equalisation there is enough time to train the adaptive filter, LMS can track the slowly varying channels.

However for wireless applications including equalisation, multi-antenna beamformers, MIMO systems and so on, the time available for training the system is very small, the channel is time varying and a tracking of the system is required. The faster the channel changes (i.e. a user driving a car), the shorter time avaiable for training, consequently the training is required more frequently [18].

QR decomposition [15] [18] [19] is an important operation in linear algebra and can be used as a method for matrix inversion or solving a set of simultaneous equations using lower wordlength arithmetic than other methods. The computational complexity of Recursive Least Squares (RLS) adaptive algorithm can be simplified by employing the QR decomposition, this results in a new algorithm names QR-RLS [15] [18] [20]. The QR-RLS algorithm is well known to be more numerically robust than the conventional RLS algorithm as it operates directly on the incoming data matrix rather than forming the inverse of the input data correlation matrix [15] [18].

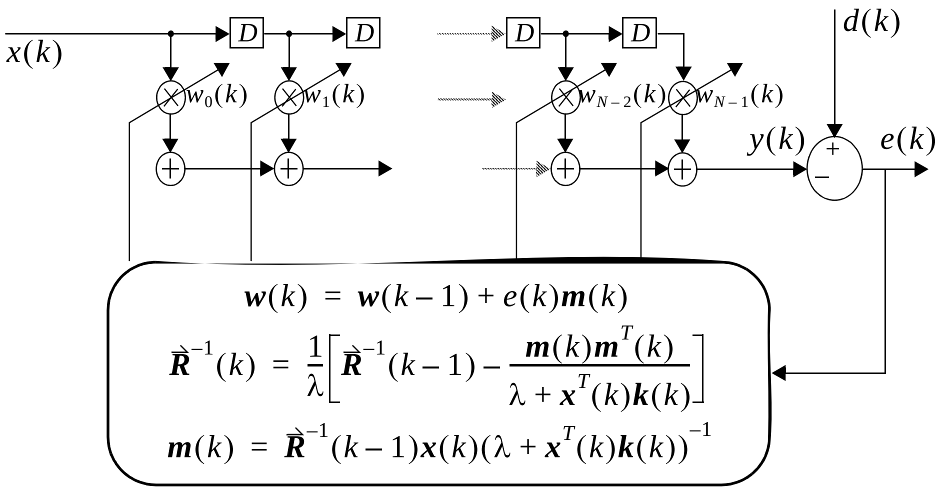

$x(k)$
D
D
D
D
$d(k)$
$w_0(k)$
$w_1(k)$
$w_{N-2}(k)$
$w_{N-1}(k)$
$y(k)$
$e(k)$
+
–
$w(k) = w(k-1) + e(k)m(k)$
$\tilde{R}^{-1}(k) = \frac{1}{\lambda}\left[\tilde{R}^{-1}(k-1) - \frac{m(k)m^T(k)}{\lambda + x^T(k)k(k)}\right]$
$m(k) = \tilde{R}^{-1}(k-1)x(k)(\lambda + x^T(k)k(k))^{-1}$

Figure: 3.1: RLS adaptive filter [15]

3.2 Adaptive Equalisation

In many communication applications, such as a wireless transceiver, the customers demand that mobile communications support high data rates with great levels of mobility, the characteristics of the radio channel are constantly changing and need to be equalised quickly. To cope with this scenario, adaptive equalisers are used in the receiver to adapt to the changes in the channel with very fast convergence. An equaliser is basically a digital filter with an adaptive algorithm (LMS, RLS, QR-RLS etc) as shown in Figure 3.1. The digital filter can be either Finite Impluse Response (FIR) or Infinite Impluse Response (IIR), although FIR adaptive equalisers are more common due to the fact that they do not suffer from instability issues that can occur in systems with feedback such as the IIR [15].

The adaptive algorithm tries to alter the weights in the filter so that the power of error signal $e(k)$ is minimised. When this occurs, the output signal $y(k)$ has matched the desired signal $d(k)$ in a least squares sense, and the channel has been equalised. The success of the filter in minimising the power of $e(k)$ will depend on the nature of input signals, the length of the adaptive filter and the adaptive algorithm used. The Recursive Least Squares (RLS) technique is able to adapt fast enough (5 or 10 $\times$ faster [15]) and with a smaller equalisation error relative to the LMS. However, RLS algorithm is more difficult to be efficiently implemented on hardware, due to its high computational complexity of order $O[N^2]$ Multiply-ACcummulate (MAC)s and one divide per iteration [15].

3.3 The RLS algorithm

The adaptive algorithm in Figure 3.1 tries to predict a desired signal, y. At the time instance, *k*, The output signal $y(k)$ is defined as:

$$y(k) = \sum_{i=0}^{N-1} w_i x(k-i) = \boldsymbol{w}^T \boldsymbol{x}(k) \tag{3.1}$$

where N is the filter order and $\boldsymbol{x}(k)$ and $\boldsymbol{w}$ are vectors composed of the input-signal samples and the filter coefficients, respectively

$$\boldsymbol{x}(k) = [\boldsymbol{x}(k) \ \ \boldsymbol{x}(k-1) \ \ \dots \ \ \boldsymbol{x}(k-N+1)]^T$$
$$\boldsymbol{w}^T = [w_0 \ \ w_1 \ \ \dots \ \ w_{N-1}]$$

In case of the complex valued implementations, the output signal $y(k)$ is represented as $\boldsymbol{w}^H \boldsymbol{x}(k)$, where the superscript H denotes the Hermitian operator.

The Recursive Least Squares (RLS) algorithm [15] is based on the least squares solution which minimises the total sum of squared errors $e(k)$. The Euclidean norm of the weighted estimation error vector corresponds to the deterministic cost function, $\varepsilon(k)$, which is given by:

$$\varepsilon(k) = \|\boldsymbol{e}(k)\|^2 = \sum_{i=0}^{k} \lambda^{k-i} |e(i)|^2 = \lambda^k e^2(0) + \lambda^{k-1} e^2(1) + \lambda^{k-2} e^2(2) + \dots + \lambda e^2(k)$$
$$= \boldsymbol{e}^T(k) \boldsymbol{e}(k) \tag{3.2}$$

where $0 < \lambda < 1$ is the forgetting factor. The parameter λ^{k-i} emphasizes the most recent error samples (where $i \approx k$) in the composition of the deterministic cost function $\varepsilon(k)$, giving to this function the ability of modeling non-stationary processes. Without a forgetting factor, the algorithm would attempt to minimise the least squares error of all the previous samples. $\boldsymbol{e}(k)$ is the system error at time instance k and is defined by

$$\boldsymbol{e}(k) = \begin{bmatrix} \lambda^{k/2}e(0) \\ \lambda^{(k-1)/2}e(1) \\ \lambda^{(k-2)/2}e(2) \\ : \\ \lambda e(k-1) \\ \lambda^{1/2}e(k) \end{bmatrix} = \begin{bmatrix} \lambda^{k/2}d(0) \\ \lambda^{(k-1)/2}d(1) \\ \lambda^{(k-2)/2}d(2) \\ : \\ \lambda d(k-1) \\ \lambda^{1/2}d(k) \end{bmatrix} - \begin{bmatrix} \lambda^{k/2}y(0) \\ \lambda^{(k-1)/2}y(1) \\ \lambda^{(k-2)/2}y(2) \\ : \\ \lambda y(k-1) \\ \lambda^{1/2}y(k) \end{bmatrix} = \boldsymbol{d}(k) - \boldsymbol{y}(k) \quad (3.3)$$

The sum of square errors can then be written as:

$$\begin{aligned} \varepsilon(k) &= \|\boldsymbol{e}(k)\|^2 = (\boldsymbol{e}^T(k)\boldsymbol{e}(k)) \\ &= [\boldsymbol{d}(k) - \boldsymbol{X}(k)\boldsymbol{w}]^T[\boldsymbol{d}(k) - \boldsymbol{X}(k)\boldsymbol{w}] \\ &= \boldsymbol{d}^T(k)\boldsymbol{d}(k) + \boldsymbol{w}^T\boldsymbol{X}^T(k)\boldsymbol{X}(k)\boldsymbol{w} - 2\boldsymbol{d}^T(k)\boldsymbol{X}(k)\boldsymbol{w} \end{aligned} \quad (3.4)$$

where

$$\boldsymbol{d}(k) = [d(k) \;\; \lambda^{1/2}d(k-1) \;\; \dots \;\; \lambda^{k/2}d(0)]^T$$

By using prewindowing on the input data, the input data matrix $\boldsymbol{X}(k)$ is defined by

$$\boldsymbol{X}(k) = \begin{bmatrix} \boldsymbol{x}^T(k) \\ \lambda^{1/2}\boldsymbol{x}^T(k-1) \\ \dots \\ \lambda^{k/2}\boldsymbol{x}^T(0) \end{bmatrix} \quad (3.5)$$

The minimum value of Eq. 3.4 is obtained by finding where the gradient vector is zero:

$$\nabla_w \varepsilon(k) = \boldsymbol{0}$$

There is one minimum point on the surface of quadratic function. The gradient vector is therefore:

$$\nabla_w \varepsilon(k) = -2\boldsymbol{X}^T(k)[\boldsymbol{d}(k) - \boldsymbol{X}(k)\boldsymbol{w}] = -2\boldsymbol{X}^T(k)\boldsymbol{d}(k) + 2\boldsymbol{X}^T(k)\boldsymbol{X}(k)\boldsymbol{w} \tag{3.6}$$

and therefore:

$$\begin{aligned} &-2\boldsymbol{X}^T(k)[\boldsymbol{d}(k) - \boldsymbol{X}(k)\boldsymbol{w}] = \boldsymbol{0} \\ &\Rightarrow \boldsymbol{X}^T(k)\boldsymbol{X}(k)\boldsymbol{w} = \boldsymbol{X}^T(k)\boldsymbol{d}(k) \end{aligned} \tag{3.7}$$

The least squares solution, denoted as $\boldsymbol{w}(k)$ and can be formed from the above equation:

$$\boldsymbol{w}(k) = [\boldsymbol{X}^T(k)\boldsymbol{X}(k)]^{-1}\boldsymbol{X}^T(k)\boldsymbol{d}(k) = \vec{\boldsymbol{R}}^{-1}(k)\vec{\boldsymbol{p}}(k) \tag{3.8}$$

[1]where $\vec{\boldsymbol{R}}$ and $\vec{\boldsymbol{p}}$ are the autocorrelation and cross-correlation matrix between the reference signal and the input signal, respectively, and are defined as

$$\vec{\boldsymbol{R}}(k) = \sum_{i=0}^{k} \lambda^{k-i}\boldsymbol{x}(i)\boldsymbol{x}^T(i+k) = \boldsymbol{X}^T(k)\boldsymbol{X}(k) \tag{3.9}$$

$$\vec{\boldsymbol{p}}(k) = \sum_{i=0}^{k} \lambda^{k-i} d(i)\boldsymbol{x}(i+k) = \boldsymbol{X}^T(k)\boldsymbol{d}(k) \tag{3.10}$$

In order to obtain the equation for the conventional RLS algorithm, the autocorrelation matrix and cross-correlation vector defined in Eq. 3.9 and 3.10, are rewritten as

$$\vec{\boldsymbol{R}}(k) = \boldsymbol{x}(k)\boldsymbol{x}^T(k) + \lambda\vec{\boldsymbol{R}}(k-1) \tag{3.11}$$

$$\vec{\boldsymbol{p}}(k) = \boldsymbol{x}(k)d(k) + \lambda\vec{\boldsymbol{p}}(k-1) \tag{3.12}$$

Substituting Eq. 3.11 and Eq. 3.12 into Eq. 3.8 yields:

1. Here $\vec{\boldsymbol{R}}$ represents an autocorrelation matrix, rather than the upper triangular matrix $\boldsymbol{R}$ obtained through the QR decomposition which appears in the remaining parts of this thesis.

$$\begin{aligned}\boldsymbol{w}(k) &= \hat{\boldsymbol{R}}^{-1}(k)[\boldsymbol{x}(k)d(k) + \lambda\boldsymbol{p}(k-1)]\\ &= \hat{\boldsymbol{R}}^{-1}(k)[\boldsymbol{x}(k)d(k) + \lambda\hat{\boldsymbol{R}}(k-1)\boldsymbol{w}(k-1)]\\ &= \hat{\boldsymbol{R}}^{-1}(k)[\boldsymbol{x}(k)d(k) + \lambda\hat{\boldsymbol{R}}(k-1)\boldsymbol{w}(k-1)\\ &\quad + \boldsymbol{x}(k)\boldsymbol{x}^T(k)\boldsymbol{w}(k-1) - \boldsymbol{x}(k)\boldsymbol{x}^T(k)\boldsymbol{w}(k-1)]\\ &= \hat{\boldsymbol{R}}^{-1}(k)\{\boldsymbol{x}(k)[d(k) - \boldsymbol{x}^T(k)\boldsymbol{w}(k-1)] + \hat{\boldsymbol{R}}(k)\boldsymbol{w}(k-1)\}\\ &= \hat{\boldsymbol{R}}^{-1}(k)\{\boldsymbol{x}(k)e(k) + \hat{\boldsymbol{R}}(k)\boldsymbol{w}(k-1)\}\end{aligned} \tag{3.13}$$

and then

$$\boldsymbol{w}(k) = \boldsymbol{w}(k-1) + e(k)\hat{\boldsymbol{R}}^{-1}(k)\boldsymbol{x}(k) \tag{3.14}$$

Solving $\hat{\boldsymbol{R}}^{-1}(k)$ is not easy as it requires matrix inversion, which is demanding in terms of the computation it requires (the inversion of $\hat{\boldsymbol{R}}(k)$ requires around $O[N^3]$ MACs, where N is the number of weights). As N grows larger, inversion of matrix $\hat{\boldsymbol{R}}(k)$ becomes more difficult. In Eq. 3.14, the computational burden for determining the inverse matrix $\hat{\boldsymbol{R}}^{-1}(k)$ can be reduced significantly by employing the *matrix inversion lemma* [19]:

$$[\boldsymbol{A} + \boldsymbol{BCD}]^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{B}[\boldsymbol{DA}^{-1}\boldsymbol{B} + \boldsymbol{C}^{-1}]^{-1}\boldsymbol{DA}^{-1} \tag{3.15}$$

which can transfer into that

$$\hat{\boldsymbol{R}}^{-1}(k) = \frac{1}{\lambda}\left[\hat{\boldsymbol{R}}^{-1}(k-1) - \frac{\hat{\boldsymbol{R}}^{-1}(k-1)\boldsymbol{x}(k)\boldsymbol{x}^T(k)\hat{\boldsymbol{R}}^{-1}(k-1)}{\lambda + \boldsymbol{x}^T(k)\hat{\boldsymbol{R}}^{-1}(k-1)\boldsymbol{x}(k)}\right] \tag{3.16}$$

For convenience of notation, consider the following auxiliary vectors:

$$\boldsymbol{m}(k) = \hat{\boldsymbol{R}}^{-1}(k)\boldsymbol{x}(k) \tag{3.17}$$

$$\boldsymbol{k}(k) = \hat{\boldsymbol{R}}^{-1}(k-1)\boldsymbol{x}(k) \tag{3.18}$$

which in conjunction to Eq. 3.16 yields that

$$\boldsymbol{m}(k) = \frac{\boldsymbol{k}(k)}{\lambda + \boldsymbol{x}^T(k)\boldsymbol{k}(k)} \tag{3.19}$$

By substituting Eq. 3.19 into Eq. 3.16, resulting

$$\begin{aligned}\vec{\boldsymbol{R}}^{-1}(k) &= \frac{1}{\lambda}[\vec{\boldsymbol{R}}^{-1}(k-1) - \vec{\boldsymbol{R}}^{-1}(k)\boldsymbol{x}(k)\boldsymbol{x}^{T}(k)\vec{\boldsymbol{R}}^{-1}(k-1)] \\ &= \frac{1}{\lambda}[\vec{\boldsymbol{R}}^{-1}(k-1) - \boldsymbol{m}(k)\boldsymbol{k}^{T}(k)]\end{aligned} \tag{3.20}$$

and the conventional RLS algorithm, in its complex valued version, can be implemented as indicated in Table 3.1:

Table 3.1: RLS [15]

```
$Initialize \ 0 \ll \lambda < 1$
$For \ each \ k$
{ $e(k) = d(k) - \boldsymbol{w}^{H}(k-1)\boldsymbol{x}(k);$
  $\boldsymbol{k}(k) = \vec{\boldsymbol{R}}^{-1}(k-1)\boldsymbol{x}(k)$
  $\boldsymbol{m}(k) = \dfrac{\boldsymbol{k}(k)}{\lambda + \boldsymbol{x}^{T}(k)\boldsymbol{k}(k)}$
  $\vec{\boldsymbol{R}}^{-1}(k) = \dfrac{1}{\lambda}\left[\vec{\boldsymbol{R}}^{-1}(k-1) - \dfrac{\boldsymbol{m}(k)\boldsymbol{m}^{H}(k)}{\lambda + \boldsymbol{x}^{H}(k)\boldsymbol{k}(k)}\right]$
  $\boldsymbol{w}(k) = \boldsymbol{w}(k-1) + e^{*}(k)\boldsymbol{m}(k)$
}
```

Although the use of matrix inversion lemma in Eq. 3.15 can significantly reduce the required computational complexity, however this inversion would must be calculated for every new sample of $\boldsymbol{x}(k)$ and $d(k)$. Hence, the matrix inversion lemma is still not a realistic option for hardware implementation. In addition to the complexity of matrix inversion, forming the product $\boldsymbol{X}^{T}(k)\boldsymbol{X}(k)$ doubles the wordlength. However, there is an approach known as QR Decomposition (QRD), which reduces the computational complexity of the least squares equation. The fundamental theory of QR decomposition is reviewed in the following section.

3.4 QR Decomposition

The QR decomposition is an extemely useful technique in least squares signal processing systems. A matrix $\boldsymbol{X}(k)$ can be decomposed into an upper triangular matrix, $\boldsymbol{R}(k)$ and orthogonal matrix $\boldsymbol{Q}(k)$ $(\boldsymbol{Q}^T(k)\boldsymbol{Q}(k) = \boldsymbol{Q}(k)\boldsymbol{Q}^T(k) = \boldsymbol{I}_{k+1})$.

$$\boldsymbol{X}(k) = \boldsymbol{Q}(k)\begin{bmatrix}\boldsymbol{R}(k)\\ \boldsymbol{0}\end{bmatrix} \tag{3.21}$$

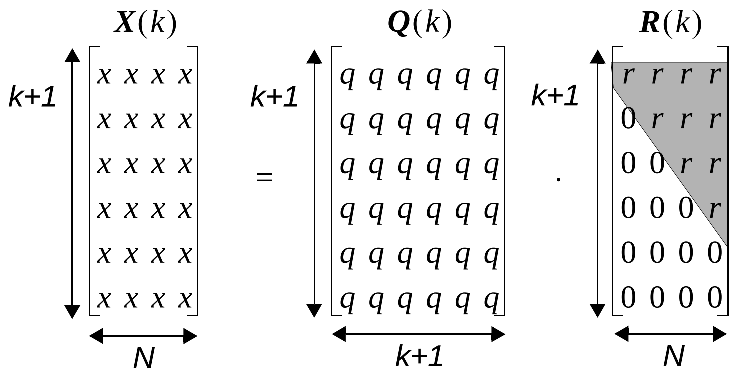


Figure: 3.2: QR Decomposition

As shown in Figure 3.2, $\boldsymbol{R}(k)$ is a N-by-N upper triangular matrix and **0** is an (k-N)-by-N null matrix. There are three methods commonly used to factor the maxtrix $\boldsymbol{X}$: Gram-Schmidt projections, Householder reflections, and a sequence of unitary transformations like the Givens rotations [18]. Gram-Schmidt and Householder reflections are good choices for the software implementation. The main advantage of unitary transformations is that one can employ vector rotations well preserving the total power of the operands [19], thus controlling the dynamic range of all variables which makes the algorithm well suited for fixed-point arithmetic. Givens rotation is more suitable and economical when the new data is added to the least squares problem and to take advantage of the previously transformed data.

Using QR decomposition to solve the least squares equation in Eq. 3.8 gives:

$$\begin{aligned}
\boldsymbol{w}(k) &= [\boldsymbol{X}^T(k)\boldsymbol{X}(k)]^{-1}\boldsymbol{X}^T(k)\boldsymbol{d}(k) \Rightarrow \boldsymbol{X}^T(k)\boldsymbol{X}(k)\boldsymbol{w}(k) = \boldsymbol{X}^T(k)\boldsymbol{d}(k) \\
[(\boldsymbol{Q}(k)\boldsymbol{R}(k))^T(\boldsymbol{Q}(k)\boldsymbol{R}(k))]\boldsymbol{w}(k) &= (\boldsymbol{Q}(k)\boldsymbol{R}(k))^T\boldsymbol{d}(k) \\
[\boldsymbol{R}^T(k)\boldsymbol{Q}^T(k)\boldsymbol{Q}(k)\boldsymbol{R}(k)]\boldsymbol{w}(k) &= \boldsymbol{R}^T(k)\boldsymbol{Q}^T(k)\boldsymbol{d}(k) \\
\cancel{\boldsymbol{R}^T(k)}\boldsymbol{R}(k)\boldsymbol{w}(k) &= \cancel{\boldsymbol{R}^T(k)}\boldsymbol{Q}^T(k)\boldsymbol{d}(k) \\
\boldsymbol{R}(k)\boldsymbol{w}(k) &= \boldsymbol{Q}^T(k)\boldsymbol{d}(k) = \boldsymbol{d}'(k)
\end{aligned} \tag{3.22}$$

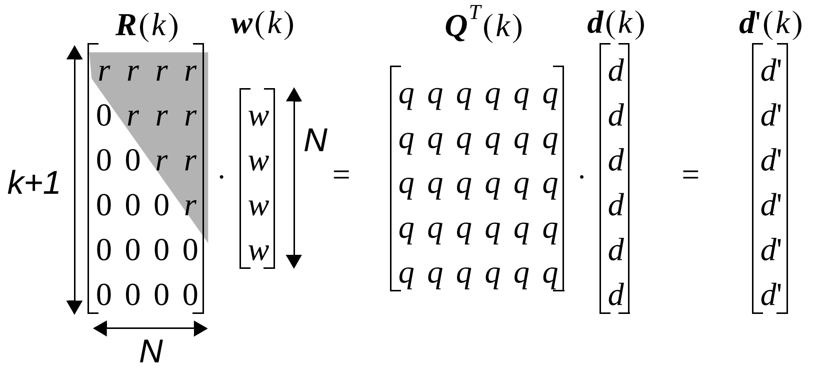


Figure: 3.3: QR Decomposition - Least Squares

The result of QR decomposition still requires the matrix $\boldsymbol{R}$ to be inverted before $\boldsymbol{w}$ can be found, as shown in Figure 3.3. This process for extracting the optimal weights is known as back-substitution. However, it is still possible to solve $\boldsymbol{w}$ without inverting any matrices, this point will be discussed in Chapter 3.6.

3.5 Standard QR-RLS Algorithm

The aformentioned QR decomposition only illustrates how the weights are calculated initially. In a real time system, the weights are computed for every new $x(k)$ and $d(k)$. However, rather than update the $\boldsymbol{X}(k)$ and $\boldsymbol{d}(k)$ matrices and repeat the process illustrated in Eq. 3.22, a recursive technique can be used to reduce the computational requirement, as shown in Figure 3.4. Here, an additional row of new x data is added to the upper triangular matrix $\boldsymbol{R}(k)$. Hence, the original matrix's dimension grows from $(k+1) \times N$ to $(k+2) \times N$.

Therefore, by using Given's rotations in Figure 3.5, the $\tilde{\boldsymbol{R}}(k)$ matrix can be

$$\boldsymbol{X}(k+1) = \begin{bmatrix} \boldsymbol{x}^T(k+1) \\ \boldsymbol{X}(k) \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{0}^T \\ \boldsymbol{0} & \boldsymbol{Q}(k) \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^T(k+1) \\ \boldsymbol{R}(k) \end{bmatrix} = \tilde{\boldsymbol{Q}}(k)\tilde{\boldsymbol{R}}(k)$$

$$and \;\; \boldsymbol{x}^T(k+1) = [\overset{\xleftrightarrow{\;N\;}}{x \;\; x \;\; \dots \;\; x}]$$

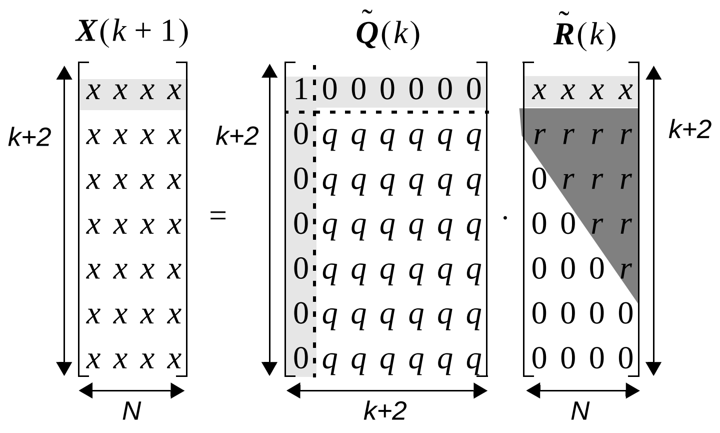


Figure: 3.4: Recursive QR Decomposition (1)

converted to the a new matrix $\boldsymbol{R}(k+1)$, where all the elements below the main diagonal of matrix are zeros,.

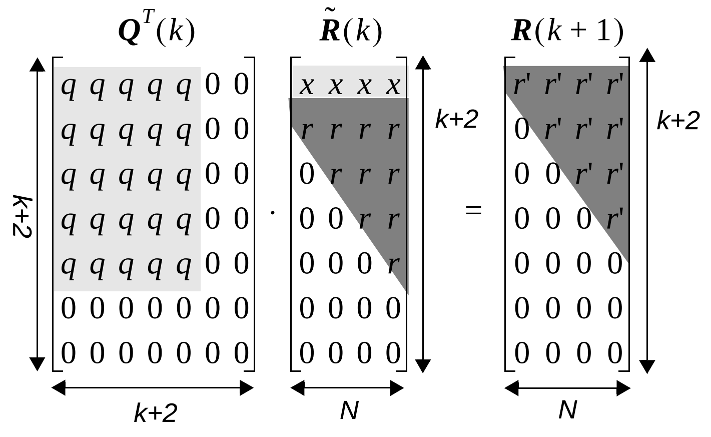


Figure: 3.5: Recursive QR Decomposition (2)

The orthogonal matrix $\boldsymbol{Q}^T(k)$ can be formed by a series of Givens rotations $\boldsymbol{G}(k)$ on $\tilde{\boldsymbol{R}}(k)$ matrix to produce a new upper triangular matrix $\boldsymbol{R}(k+1)$. The series of Givens rotations is achieved in a row-wise fashion in Figure 3.6. Thus the matrix $\boldsymbol{Q}^T(k)$ can be expressed as

$$\boldsymbol{Q}^T(k) = \boldsymbol{G}_{N-1}(k)\boldsymbol{G}_{N-2}(k)\ldots\boldsymbol{G}_1(k)\boldsymbol{G}_0(k) \tag{3.23}$$

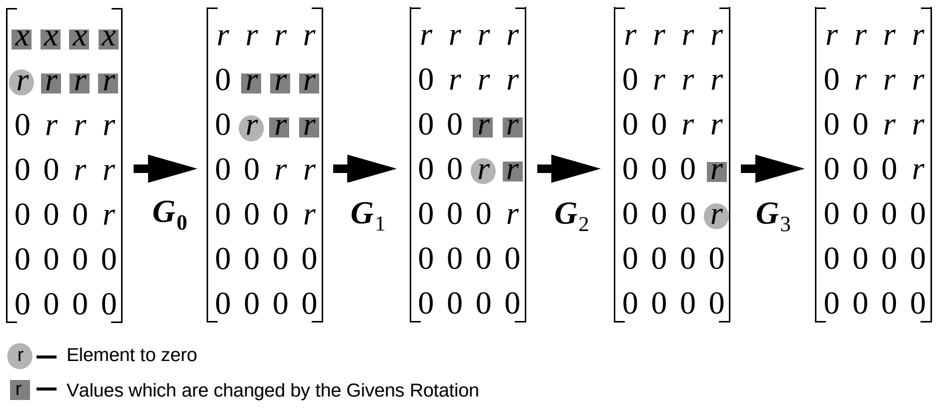


Figure: 3.6: Series of Givens rotations (1)

Each Givens rotation is defined as the following $(N+1) \times (N+1)$ matrix:

$$\boldsymbol{G}_i = \begin{bmatrix} \boldsymbol{I}_i & & \cdots & \\ & c_i & s_i & \\ \vdots & -s_i & c_i & \vdots \\ & \cdots & & \boldsymbol{I}_{(N-i-2)} \end{bmatrix} \tag{3.24}$$

where $i = 0,\ 1,\ 2\ \ldots\ N-1$. c_i and s_1 represent cosine and sine function respectively, as illustrated in Figure 3.7.

The same Givens rotations also needs to be applied to the vector containing the new d values. Eq. 3.22 can therefore be transferred to:

$$\boldsymbol{G}_3(k)\boldsymbol{G}_2(k)\boldsymbol{G}_1(k)\boldsymbol{G}_0(k)\boldsymbol{X}(k)\boldsymbol{w}(k) = \boldsymbol{R}(k)\boldsymbol{w}(k) = \boldsymbol{G}_3\boldsymbol{G}_2\boldsymbol{G}_1\boldsymbol{G}_0\boldsymbol{d}(k) = \boldsymbol{d}'(k) \tag{3.25}$$

where $\boldsymbol{G}_3(k)\boldsymbol{G}_2(k)\boldsymbol{G}_1(k)\boldsymbol{G}_0(k) = \boldsymbol{Q}^T(k)$

As the previous discussion in Chapter 3.4, it is possible to use back-substitution to find $\boldsymbol{w}$ without carrying out matrix inversion. In Eq. 3.22, it can be seen that $\boldsymbol{Q}^T$ is applied to both the $\boldsymbol{X}$ and the $\boldsymbol{d}$ matrices. Starting with w_{N-1}, the weights can be calculated by Eq. 3.26. Theoretically this process is easy, however, it is computationally expensive as it requires division, multiplication and addition.

$$c_1 = \frac{x_1}{\sqrt{x_1^2+\lambda r_1^2}} \qquad s_1 = \frac{r_1}{\sqrt{x_1^2+\lambda r_1^2}}$$

$$k+2\updownarrow\underbrace{\begin{bmatrix} c_2 & s_2 & 0 & 0 & 0 & 0 & 0 \\ -s_2 & c_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{G_0}} \begin{bmatrix} x_1 & x & x & x \\ r_1 & r & r & r \\ 0 & r & r & r \\ 0 & 0 & r & r \\ 0 & 0 & 0 & r \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} r_g & \bar{r} & \bar{r} & \bar{r} \\ 0 & r_h & \bar{r} & \bar{r} \\ 0 & r & r & r \\ 0 & 0 & r & r \\ 0 & 0 & 0 & r \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad r_g = \sqrt{x_1^2+\lambda r_1^2}$$

$$c_2 = \frac{r_h}{\sqrt{r_h^2+\lambda r_2^2}} \qquad s_2 = \frac{r_2}{\sqrt{r_h^2+\lambda r_2^2}}$$

$$k+2\updownarrow\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & c_2 & s_2 & 0 & 0 & 0 & 0 \\ 0 & -s_2 & c_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{G}_1} \begin{bmatrix} r_g & \bar{r} & \bar{r} & \bar{r} \\ 0 & r_h & \bar{r} & \bar{r} \\ 0 & r_2 & r & r \\ 0 & 0 & r & r \\ 0 & 0 & 0 & r \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} r_g & \bar{r} & \bar{r} & \bar{r} \\ 0 & r_j & \tilde{r} & \tilde{r} \\ 0 & 0 & \tilde{r} & \tilde{r} \\ 0 & 0 & r & r \\ 0 & 0 & 0 & r \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad r_j = \sqrt{r_h^2+\lambda r_2^2}$$

$$c_3 = \frac{r_m}{\sqrt{r_m^2+\lambda r_3^2}} \qquad s_3 = \frac{r_3}{\sqrt{r_m^2+\lambda r_3^2}}$$

$$k+2\updownarrow\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & c_3 & s_3 & 0 & 0 & 0 \\ 0 & 0 & -s_3 & c_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{G}_2} \begin{bmatrix} r_g & \bar{r} & \bar{r} & \bar{r} \\ 0 & r_j & \tilde{r} & \tilde{r} \\ 0 & 0 & r_m & \tilde{r} \\ 0 & 0 & r_3 & r \\ 0 & 0 & 0 & r \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} r_g & \bar{r} & \bar{r} & \bar{r} \\ 0 & r_j & \tilde{r} & \tilde{r} \\ 0 & 0 & r_n & \hat{r} \\ 0 & 0 & 0 & \hat{r} \\ 0 & 0 & 0 & r \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad r_n = \sqrt{r_m^2+\lambda r_3^2}$$

$$c_4 = \frac{r_p}{\sqrt{r_p^2+\lambda r_4^2}} \qquad s_4 = \frac{r_4}{\sqrt{r_p^2+\lambda r_4^2}}$$

$$k+2\updownarrow\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & c_4 & s_4 & 0 & 0 \\ 0 & 0 & 0 & -s_4 & c_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{G}_3} \begin{bmatrix} r_g & \bar{r} & \bar{r} & \bar{r} \\ 0 & r_j & \tilde{r} & \tilde{r} \\ 0 & 0 & r_n & \hat{r} \\ 0 & 0 & 0 & r_p \\ 0 & 0 & 0 & r_4 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} r_g & \bar{r} & \bar{r} & \bar{r} \\ 0 & r_j & \tilde{r} & \tilde{r} \\ 0 & 0 & r_n & \hat{r} \\ 0 & 0 & 0 & r_q \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad r_q = \sqrt{r_p^2+\lambda r_4^2}$$

Figure: 3.7: Series of Givens rotations (2)

Table 3.2: QRD-RLS

```
For each $k$
{Obtaining $\boldsymbol{Q}(k)$ and updating $\boldsymbol{R}(k)$;
    $\begin{bmatrix} 0 \\ \boldsymbol{R}(k) \end{bmatrix} = \boldsymbol{Q}(k)\begin{bmatrix} \boldsymbol{x}^T(k) \\ \lambda^{1/2}\boldsymbol{R}(k-1) \end{bmatrix}$
    Obtaining $\gamma(k)$;
    $\gamma(k) = \Pi_{i=0}^{N}\cos\theta_i(k)$
    Obtaining $\xi(k)$ and updating $d(k)$;
    $\begin{bmatrix} \xi(k) \\ \boldsymbol{p}(k) \end{bmatrix} = \boldsymbol{Q}(k)\begin{bmatrix} d(k) \\ \lambda^{1/2}\boldsymbol{p}(k-1) \end{bmatrix}$
    Obtaining $e(k)$;
    $e(k) = \xi^*(k)\gamma(k)$
    $e(k) = d(k) - \boldsymbol{w}^H(k)\boldsymbol{x}(k)$
}
```

$$\underbrace{\begin{bmatrix} r_{00} & r_{01} & \cdots & r_{0(N-1)} \\ 0 & r_{11} & \cdots & r_{1(N-1)} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & r_{(N-1)(N-1)} \end{bmatrix}}_{\boldsymbol{R}(k)} \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{(N-1)} \end{bmatrix}}_{\boldsymbol{w}(k)} = \underbrace{\begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{(N-1)} \end{bmatrix}}_{\boldsymbol{d}'(k)}$$

$$w_{N-1} = \frac{d_{N-1}}{r_{(N-1)(N-1)}}, \cdots\cdots w_1 = \frac{d_1 - r_{12}w_2}{r_{11}},\ w_0 = \frac{d_0 - r_{02}w_2 - r_{01}w_1}{r_{00}} \tag{3.26}$$

The equations of the complex valued conventional QR-RLS algorithm is summarised in Table 3.2, where the QR triangularization has no influence on the performance of the conventional RLS algorithm in Table 3.1.

The matrix $\boldsymbol{Q}(k)$ in Table 3.2 is the orthogonal rotation matrix, which triangularises the left hand side matrix of Eq. 3.26, λ is the *forgetting factor* (set to a value slightly less than 1), and $\xi(k)$ is the *posteriori* least-square residual, $\boldsymbol{R}(k)$ is an $N \times N$ upper triangular matrix, and the N element vector $\boldsymbol{x}(k)$ is the input data vector. $\boldsymbol{x}(k) = [x_0(k) \;\; x_1(k) \;\; \dots \;\; x_{N-1}(k)]$, N is the number of weights in the adaptive filter which finds the weight vector $\boldsymbol{w}^T(k) = [w_0(k) \;\; w_1(k) \;\; \dots \;\; w_{N-1}(k)]$ to minimise the least square error $e(k) = d(k) - y(k)$, where $y(k) = \boldsymbol{w}^T(k)\boldsymbol{x}(k)$, [1]$d(k)$ represents the desired signal.

3.6 Standard QR-RLS Systolic Array

Figure 3.8 shows the Signal Flow Graph (SFG) representation of the real valued QR-RLS systolic array for an $N = 3$ weight FIR filter [16] [20]. This array uses the Givens rotation [21] to convert the input data into an upper triangular matrix $\boldsymbol{R}$. Each r_{ij} stored and updated inside of the array is one element of this upper triangular matrix, where the subscript ij represents the location of the element in $\boldsymbol{R}$ matrix. The definitions of the boundary cell (BC) and internal cell (IC) of QR-RLS array are also described in Figure 3.8. Sine function $s(\;)$ and cosine term $c(\;)$ are used to calculate the Givens rotation within a boundary cell. The least square error $e(k)$ can be found from the single final multiplier. The non-recursive column of PEs highlighted by the dashed frame generates the likelihood vector γ, which is the product of all the c_i [22]. Note that this operation is normally undertaken in the BCs [16] [23] [24]. However by attaching the right most column of ICs, it becomes possible to construct the BC and IC with very similar architectures, which can be exploited with the efficient folding of both types of cell onto a single processing element (to be discussed in Section 5.5).

In many implementations [23] [24] the final coefficient weight vector is derived from the outputs of the QR decomposition algorithm using the aformentioned back-substitution. As shown in Figure 3.9, to implement a systolic structure for back-substitution, one approach is to append a linear array to the upper triangular QR-RLS array. Adaption is halted and the r and p values are clocked out from the triangular

1. Here $d(k)$ represents the desired signal, rather than the CORDIC decision factor d_i in Section 4.

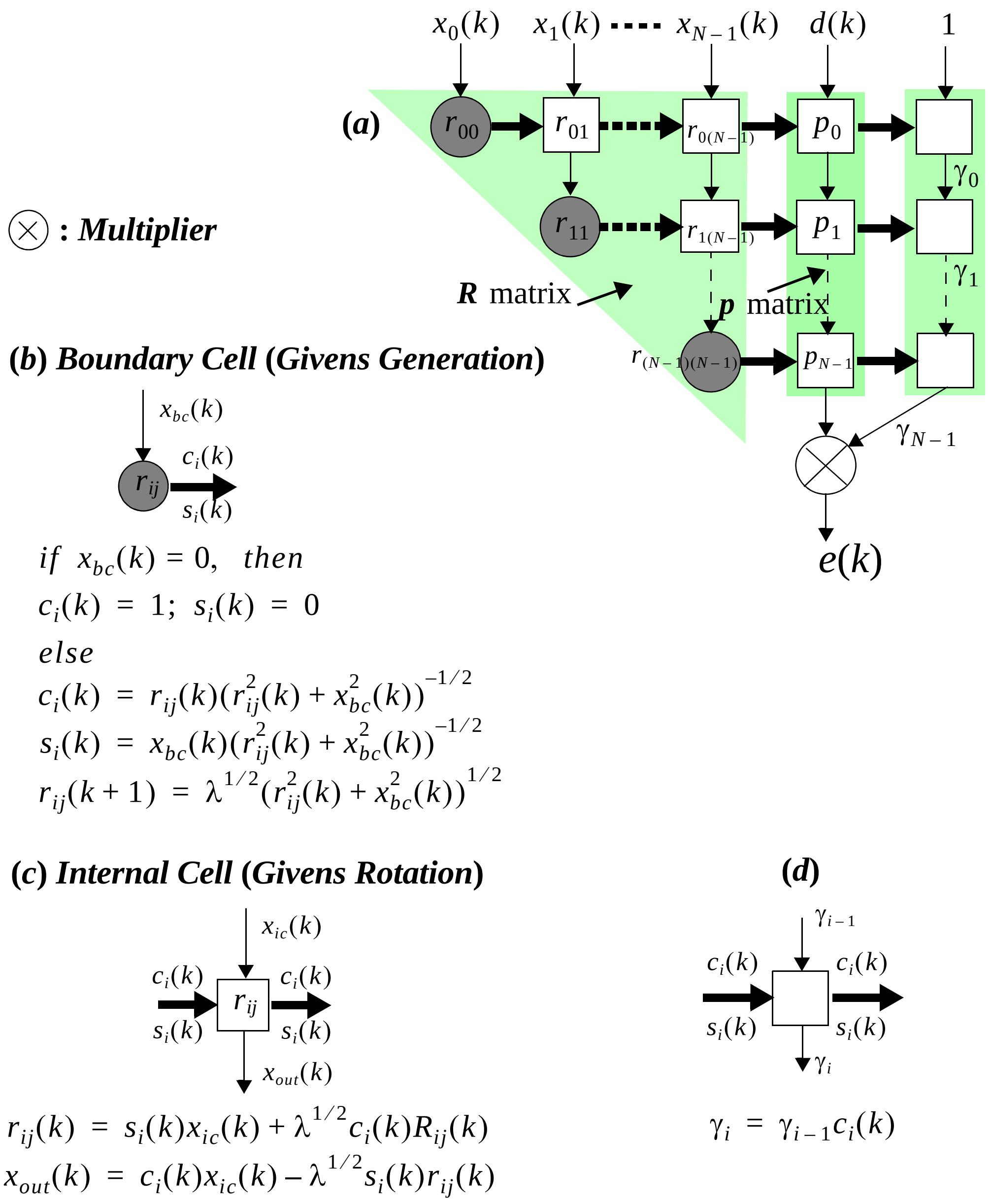

$x_0(k)$
$x_1(k)$
$x_{N-1}(k)$
$d(k)$
1
(*a*)
r_{00}
r_{01}
$r_{0(N-1)}$
p_0
γ_0
r_{11}
$r_{1(N-1)}$
p_1
γ_1
R matrix
p matrix
$r_{(N-1)(N-1)}$
p_{N-1}
γ_{N-1}
$e(k)$
: ***Multiplier***
(***b***) ***Boundary Cell*** (***Givens Generation***)
$x_{bc}(k)$
$c_i(k)$
r_{ij}
$s_i(k)$
$if\ x_{bc}(k) = 0,\ then$
$c_i(k) = 1;\ s_i(k) = 0$
$else$
$c_i(k) = r_{ij}(k)(r_{ij}^2(k) + x_{bc}^2(k))^{-1/2}$
$s_i(k) = x_{bc}(k)(r_{ij}^2(k) + x_{bc}^2(k))^{-1/2}$
$r_{ij}(k+1) = \lambda^{1/2}(r_{ij}^2(k) + x_{bc}^2(k))^{1/2}$
(***c***) ***Internal Cell*** (***Givens Rotation***)
$x_{ic}(k)$
$c_i(k)$
r_{ij}
$s_i(k)$
$x_{out}(k)$
$r_{ij}(k) = s_i(k)x_{ic}(k) + \lambda^{1/2}c_i(k)R_{ij}(k)$
$x_{out}(k) = c_i(k)x_{ic}(k) - \lambda^{1/2}s_i(k)r_{ij}(k)$
(***d***)
γ_{i-1}
$c_i(k)$
$s_i(k)$
γ_i
$\gamma_i = \gamma_{i-1}c_i(k)$

Figure: 3.8: Standard systolic array of real valued QR-RLS

array [23] [24].

However, the process of back-substitution implies division operations, and these are susceptible to divide-by-zero errors, so the procedure of back-substitution is a process that is not numerically stable or robust and hence may result in overflow or underflow unless performed with high precision fixed point, or indeed floating point [25].

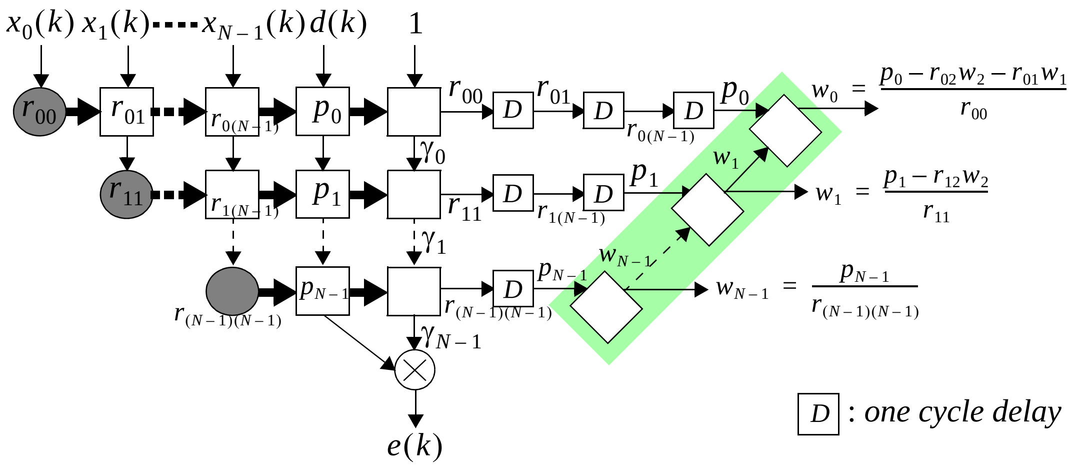


Figure: 3.9: Standard QR-RLS with back-substitution array

Another issue with the regular implementation of the QR array is that it is not easily interfaced to the back-substitution array, which is a linear array that requires the r and p values to be first "shifted" out from the QR array [24]. Therefore, although the QR decomposition is elegantly implemented on FPGA compatible data-flow triangular arrays, if the weights are required, then the back-substitution is not compatible for an FPGA dataflow array. One method of managing this risk is to offload the task to an embedded processor, using an embedded processor to execute a series of division operation. However compared to a data flow architecture, the latency incurred by this approach is likely to be significant [23].

In the next section, the alternative extended (also called downdating) QR-RLS algorithm is derived for weight extraction. It could mitigate the need for the back-substitution by adding a second lower triangular downdating array such that the final adaptive filter weights can then be extracted by a single multiplication and subtraction operation.

3.7 Extended QR-RLS Systolic Array

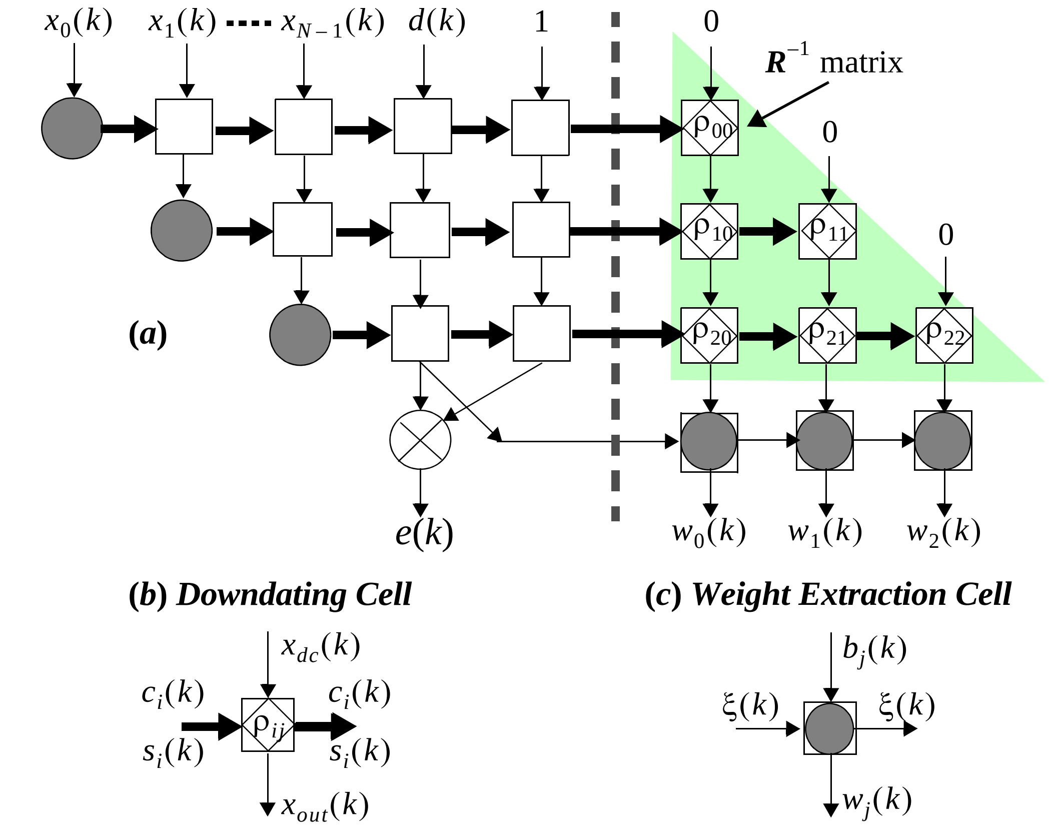
The extended QR-RLS, also known as QR-RLS with downdating, is derived for

Table 3.3: Extended QRD-RLS

```
For each $k$
{Obtaining $\boldsymbol{Q}(k)$ and updating $\boldsymbol{R}(k)$;
  $\begin{bmatrix} 0 \\ \boldsymbol{R}(k) \end{bmatrix} = \boldsymbol{Q}(k) \begin{bmatrix} \boldsymbol{x}^T(k) \\ \lambda^{1/2}\boldsymbol{R}(k-1) \end{bmatrix}$
  Obtaining $\gamma(k)$;
  $\gamma(k) = \Pi_{i=0}^{N} \cos\theta_i(k)$
  Obtaining $\xi(k)$ and updating $d(k)$;
  $\begin{bmatrix} \xi(k) \\ \boldsymbol{p}(k) \end{bmatrix} = \boldsymbol{Q}(k) \begin{bmatrix} \boldsymbol{d}(k) \\ \lambda^{1/2}\boldsymbol{p}(k-1) \end{bmatrix}$
  Obtaining $e(k)$;
  $e(k) = \xi^*(k)\gamma(k)$
  Obtaining $\boldsymbol{b}(k)$ and updating $\boldsymbol{R}^{-H}(k)$ in downdating;
  $\begin{bmatrix} \boldsymbol{b}(k) \\ \boldsymbol{R}^{-H}(k) \end{bmatrix} = \boldsymbol{Q}(k) \begin{bmatrix} \boldsymbol{0} \\ \lambda^{1/2}\boldsymbol{R}^{-H}(k-1) \end{bmatrix}$
  Updating the coefficient vector;
  $\boldsymbol{w}(k) = \boldsymbol{w}(k-1) - \boldsymbol{b}(k)\xi^*(k)$
}
```

parallel weight extraction. This can be implemented in the form of a COordinate Rotation DIgital Computer (CORDIC) based double triangular array [26]. Presented in Table 3.3, the extended QR-RLS algorithm has computational complexity of $O(N^2)$ MACs per sample.

The SFG of this QR-RLS array with downdating is shown in Figure 3.10. The triangular section on the left of dashed line, consists of the same BCs and ICs as Figure 3.8. The lower triangular (downdating) section (on the right hand side of dashed line) rotates the $\boldsymbol{R}^{-1}$ matrix stored in the ◇ cells and an externally applied vector of zeros. Obviously, both the ICs in Figure 3.8 and the Downdating Cell (DC) in Figure 3.10 execute nearly the same operation. The only difference is that the *forgetting factor* in



$$x_{out}(k) = c_i(k)x_{ic}(k) - \lambda^{-1/2}s_i(k)\rho_{ij}(k)$$
$$\rho_{ij}(k) = s_i(k)x_{ic}(k) + \lambda^{-1/2}c_i(k)\rho_{ij}(k)$$

$$w_j(k) = w_j(k-1) - \xi(k)b_j(k)$$

Figure: 3.10: Systolic array of QR-RLS with downdating

each DC is equal to $\lambda^{-1/2}$. Hence the DC could also be mapped together with the IC and BC. In Figure 3.10, the bottom row of Extraction Cells (EC), is employed to extract the weights $w(k)$ by the Multiply-ACcumulate (MAC) operation [26] [27].

Although the method of downdating for weight extraction requires more arithmetic computation than back-substitution, the downdating calculation can efficiently share the architecture of the main QR calculation. In Figure 3.10 the lower triangular (downdating) section on the right hand side denotes the $\boldsymbol{R}^{-H}$ matrix stored in the DC cells ◇ and an externally applied input of zeros. Both the ICs and DCs execute nearly the same operation. The only difference is that the *forgetting factor* in each DC is equal to $\lambda^{-1/2}$. Hence the DC could also be mapped together with the IC and BC, and include simple multiplexing/scheduling as appropriate between $\lambda^{1/2}$ and $\lambda^{-1/2}$. This mapping and folding strategy is described in Chapter 6.5.

The extended QR-RLS architecture provides the parallel extraction of filter coefficients, however the introduction of lower triangular downdating array doubles the number of PEs. An alternative approach, which allows the parallel calculation of the weight vector without the back-substitution, but has the same number of PEs compared to the standard QR-RLS array, is known as the Inverse QR-RLS (IQRD-RLS) algorithm [29] [30]. IQRD-RLS algorithm bases on the update of the inverse Cholesky factor. Different from the extended QR-RLS algorithm, rather than updating the $\boldsymbol{R}$ matrix during every QR decomposition iteration, the IQRD-RLS update the product of matrix $\boldsymbol{R}$ and input vector $\boldsymbol{x}$. After generating the new matrix $\boldsymbol{R}$, the matrix $\boldsymbol{R}$ must be loaded out to multiply with the input array $\boldsymbol{x}$ for the next iteration [29]. Similar with the back-substitution, this procedure of ‘shifting’ out matrix $\boldsymbol{R}$ leads to the high latency. Also it is almost impossible to map the systolic array to a cost efficient linear or processor-like architecture in Section 3.8.

3.8 Efficient QR-RLS

For a typical application, the cost is too large to directly map the QR-RLS systolic array in Figure 3.10 to FPGA, particularly when the array size N further increases. Therefore, a challenge in this research is the efficient mapping of QR-RLS systolic array. Two most widely used mapping strategies, the linear array wise and processor wise, are demonstrated as follows:

3.8.1 Linear Array Like QR-RLS Architecture

Figure 3.11 introduces one mapped transformation for QR-RLS. By mapping all the rows of PEs in Figure 3.11 (*a*) together, the original parallelogram architecture is mapped to a linearly interconnected array of PEs, the resulted linear QR array is shown in Figure 3.11 (*b*). Each of these ‘internal’ PE requires only the nearest neighbour interconnect, making it highly scalable and suitable for high-speed FPGA implementation. This kind of linear array reduces the required number of Processing Elements (PE) considerably.

Figure 3.12 is the second alternative improved architecture for QR-RLS. The upper triangular QR array could be assumed as a rectangular array. But the signal output-input relation between each row should be set as stated in Figure 3.12 (*a*). The folding technique could be applied as described in Figure 3.12 (*b*), i.e. still produce a linear array to execute QR-RLS filtering.

There is another mapping scheme which can transfer both the architectures in Figure 3.11 (*a*) and Figure 3.12 (*a*) into a linear column-like array [28], as shown in Figure 3.13 (*b*).

Compared with the conventional double triangulars array, the sizes of the above 3 types of linear architecture are significantly reduced. However, the linear architecture still utilizes large hardware when the QR-RLS systolic array’s size increases.

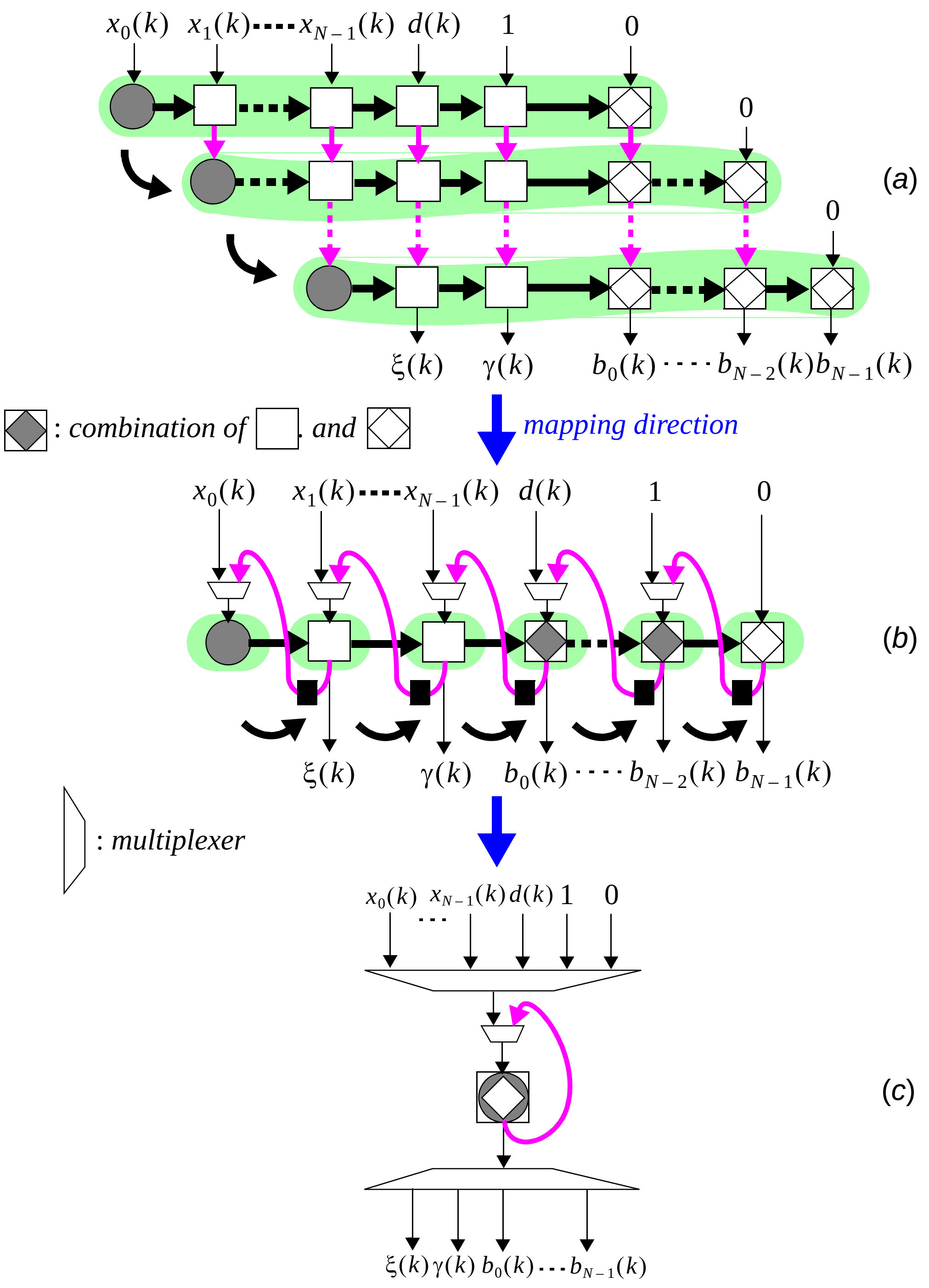

$x_0(k)$ $x_1(k)$ $x_{N-1}(k)$ $d(k)$ 1 0
0
0
(a)
$\xi(k)$ $\gamma(k)$ $b_0(k)$ $b_{N-2}(k)$ $b_{N-1}(k)$
: combination of and
mapping direction
$x_0(k)$ $x_1(k)$ $x_{N-1}(k)$ $d(k)$ 1 0
(b)
$\xi(k)$ $\gamma(k)$ $b_0(k)$ $b_{N-2}(k)$ $b_{N-1}(k)$
: multiplexer
$x_0(k)$ $x_{N-1}(k)$ $d(k)$ 1 0
(c)
$\xi(k)$ $\gamma(k)$ $b_0(k)$ $b_{N-1}(k)$

Figure: 3.11: Triangularization array transformation (1)

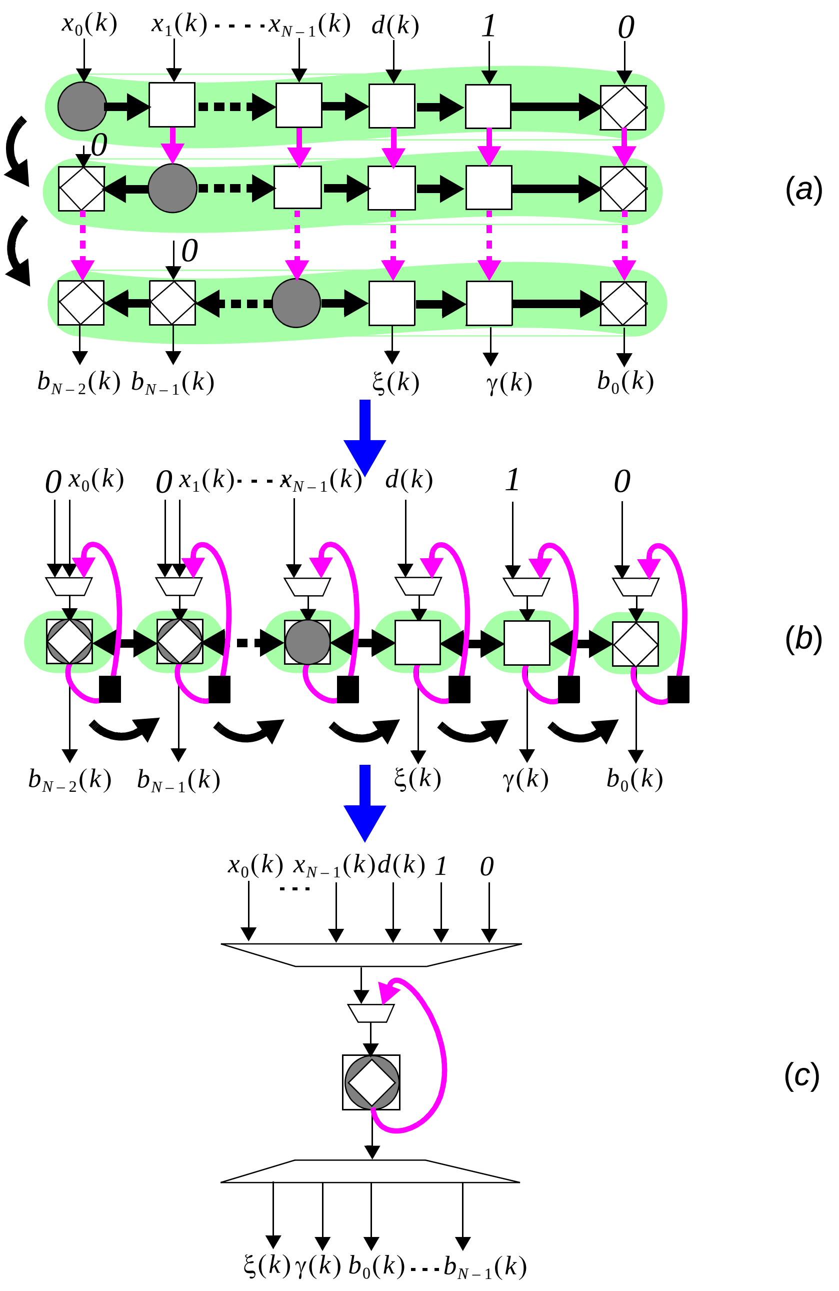


Figure: 3.12: Triangularization array transformation (2)

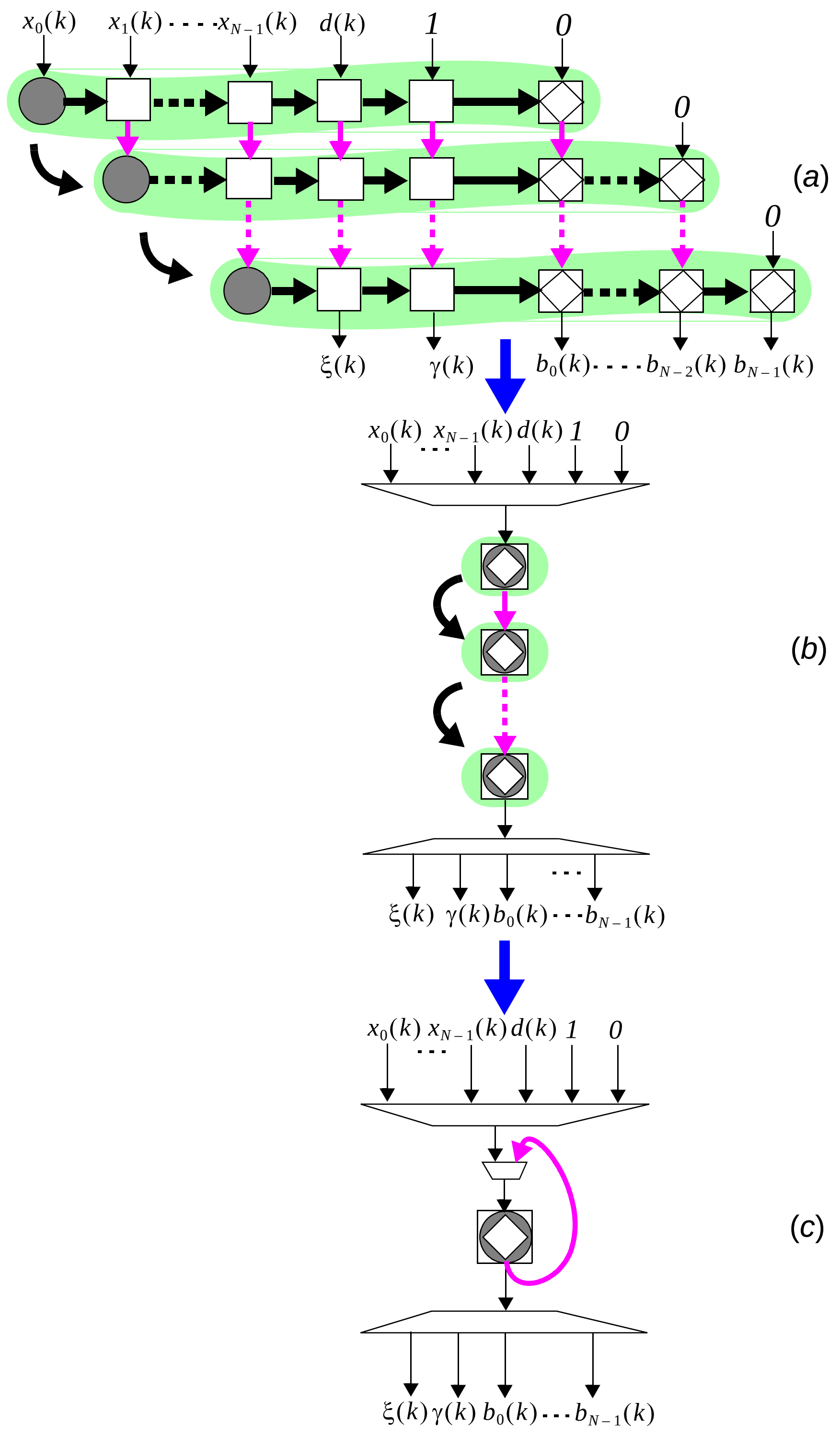


Figure: 3.13: Triangularization array

3.8.2 Processor-Like QR-RLS Architecture

Compared to the linear QR-RLS array, the processor-like architecture in Figure 3.10 (*c*) only consists of a single Givens rotation unit regardless of the size of systolic array. The data flow in the single processing element architecture mimics that of the original triangular array architcture. Since only one processing element is used, no parallel computing is possible. In Figure 3.11 (*c*). This folding technique combines the PEs in Figure 3.11 (*b*) to a single PE, which is the same with Figure 3.12 (*c*) and Figure 3.13 (*c*). Despite the linear array in Figure 3.11 (*b*), Figure 3.12 (*b*) and Figure 3.13 (*b*) are different, the resulted processor-like single PEs are exactly the same.

The single PE QR-RLS array is easier to scale if the larger QR matrix is processed. As shown in Figure 3.11 (*b*), Figure 3.12 (*b*) and Figure 3.13 (*b*), QR matrix size scaling leads to increase/decrease the number of PEs on the linear array. Comparing with the linear array-like QR-RLS in Section 3.8.1, the processor-like QR-RLS can further save the hardware area. Hence, it is necessary to develop a composed CORDIC which can be featured as BCs, ICs and DCs. This point will be discussed in Chapter 5.2.

3.9 Fast QR-RLS Algorithms

The extended QR-RLS architecture provides the computational complexity of $O(N^2)$ MACs per sample, , where N is the order of the adaptive filter. A variety of computation-efficient Fast QR-RLS algorithms have been presented during the last decade, which can reduce the computational complexity to $O[N]$, have been presented in the last decades [18]. Such these 'fast' QR-RLS algorithms are classified into the following types:

3.9.1 QR-RLS Lattice

The complexity of both the standard and extended QR-RLS filtering algorithms is given by $O[N^2]$. In the case of a large number of filter coefficients, N, the computational complexity required for both algorithms will be intense. The Fast QR-RLS algorithms [18] [31] [32], which have the computational complexity of $O[N]$, employ the well-known lattice architecture as this inherits the pipelinable properties of the classic least-squares lattice (LSL) algorithm [33] [34]. They do not, however, allow a straightforward and parallel computation of the weight vector, so an additional back-substitution procedure has to be used.

3.9.2 Algorithmic Engineering Applied to the QR-RLS

The use of algorithmic engineering, which was introduced by McWhirter [35], to design systolic array of QR-RLS adaptive filters is also an important research topic. Different from the Fast QR-RLS algorithm in Section 3.9.1 which targets the algorithm to the low complexity lattice architecture, the algorithmic engineering solution still use the systolic array architecture for efficiently prototyping parallel algorithms and architectures [36].

Reference [37] extended the work in [35] by applying the algorithmic engineering to transform the extended QR-RLS algorithm to an equivalent algorithm with a lower computational complexity, however the resultant algorithm is hard to implement or map to a cost-saving archiecture,

3.9.3 Multichannel QR-RLS

It is often possible to directly apply the standard single-channel algorithms to deal with the multichannel problem. In order to obtain a computationally efficient solution, the multichannel QR-RLS algorithms can be considered as extensions of the basic single-channel QR-RLS algorithms to the case of a multichannel input vector, where it can be assumed that each channel has a time-shift structure.

In [38] [39] and [40], QR multichannel RLS lattice algorithm have been addressed, for the case of unequal length input channel filters can be applied to obtain the exact

LS solutions with a reduced computational complexity. However compared to the stardard and the extended systolic array QR-RLS, either the single or multi-channel Fast QR-RLS lattice/systolic array filter is mathematically more complicated to implement. Though the pipelining registers are allowed to be inserted between the neighborhood lattice/systolic array stages, the throughputs of Fast QR-RLS adaptive filters are still limited by their inherent recursive update operations, i.e., they can not be pipelined at fine-grain level [41].

3.10 Look-ahead Transformation

The throughputs of all the Fast QR-RLS adaptive filters are still limited by the inherent recursive update operations, namely they can not be pipelined at fine-grain level [41]. To increase the throughput of adaptive filters, look-ahead transformations [41] can be applied as an alternative solution to achieve more fine-grain pipelining by retiming. In recent years, various Look-ahead technologies have been used in the architecture and VLSI implementation of the communication physical layer design to improve the clocking speed of adaptive DSP algorithms.

In [42], a pipelined architecture of soft-threshold-based multilayer decision feedback equalizer (STM-DFE) was presented, using the relax Look-ahead transformed delayed least mean-squared (DLMS) algorithm. Reference [43] presents an ASIC implementation of the relax look-ahead updated DLMS based joint automatic gain control (AGC)-equalization for a wireline transceiver. The classic Look-ahead technique has been successfully implemented on an FPGA for a standard LMS detector for the multiple input multiple output (MIMO) communication system [44]. Also an ASIC implementation of the look-ahead updated precoder was recently proposed in [45]. This paper aims to investigate an implementation of the Look-ahead technique for the QR-RLS algorithm, due to its faster convergence behavior than LMS. The aformentioned relaxed look-ahead technique based adaptive filtering, which was proposed in [46], maintains the functionality of the conventional algorithm rather than the input-output behaviour, hence certain approximations must be made in this algorithm which lead to slower convergence rate. In [47], an alternative Annihilation-Reordering look-ahead Transformation (ART) was selected to achieve fine-grain pipelining in real valued QRD-RLS adaptive filters and its application on adaptive MVDR beamforming has been addressed in [48]. Similar to the traditional mult-add Look-ahead, annihilation-reordering Look-ahead transforms a sequential recursive algorithm to an equivalent concurrent one by creating additional parallelism in the algorithm. The advantage of this technique is that it can transform an orthogonal sequential recursive DSP algorithm to an equivalent orthogonal concurrent one by creating additional concurrency in the algorithm [47]. The resulting transformed algorithm can then be pipelined, which is attractive for VLSI implementations.

However, most modern communication applications require complex valued DSP arithmetic for any equalization or beamforming algorithms, and ART proposed in [47] is only a real valued filter. Hence this chapter extends the ART to cope with the complex Givens rotation based QR-RLS systolic array which is clearly more difficult to implement and has higher computational complexity than the real valued system. This chapter presents a novel framework of Complex valued Annihilation-Reordering Look-ahead Transformation (C-ART) that is able to increase the throughput/sample-rate of QR-RLS based applications.

3.10.1Annihilation-Reordering Look-ahead Transformation (ART)

In this section, from both the block processing and the iteration point of view, the annihilation-reordering Look-ahead transformation, which was presented in [47] for real Givens rotation-based QR-RLS filtering algorithms, is reviewed.

Regarding the iteration point of view, the basic Givens rotation of the QR recursion is given in Eq. 3.27:

$$\begin{bmatrix} r(k) \\ 0 \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} r(k-1) \\ x(k) \end{bmatrix} \tag{3.27}$$

Assuming a 2-time speed up is desired for the QR update in Figure 3.14 (*a*). A sequence of Givens rotations, whose products form the orthogonal transformation matrix $\boldsymbol{Q}(k)$ to annihilate the block input data matrix $\boldsymbol{x}(k)$, is now determined. Consider the block update form of this QR update procedure, firstly the traditional sequential update operation is used. The input data $\boldsymbol{x}$ is annihilated row-by-row, and the diagonal r elements are involved in each update. A direct Look-ahead transformation by iterating Eq. 3.27 two times is represented by Eq. 3.28.

$$\begin{bmatrix} r(k) \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} c_1 & 0 & s_1 \\ 0 & 1 & 0 \\ -s_1 & 0 & c_1 \end{bmatrix} \begin{bmatrix} c_2 & s_2 & 0 \\ -s_2 & c_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r(k-2) \\ x(k-1) \\ x(k) \end{bmatrix} \tag{3.28}$$

The corresponding SFG is also shown in Figure 3.14 (*c*). It can be seen that the number of Givens rotations inside the feedback loop increases linearly with the number of delay elements in the loop. Therefore, there is no overall improvement in either the sample or clock speed, as the critical path is doubled.

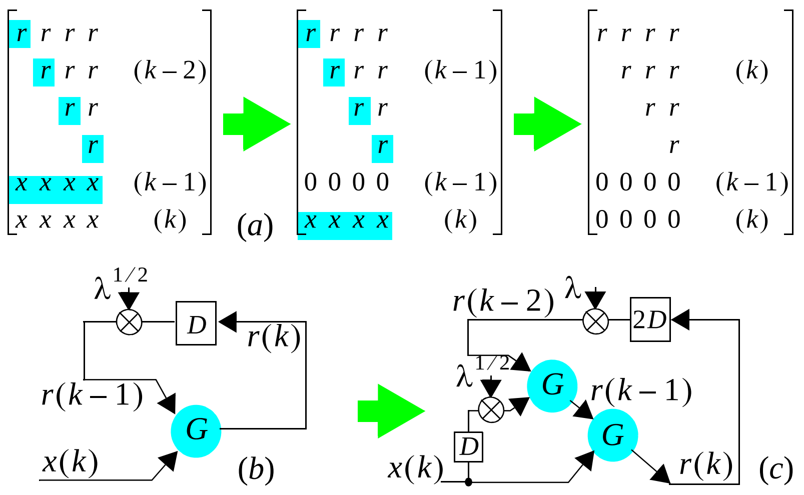


Figure: 3.14: (*a*) Direct look-ahead QR update procedure; (*b*) Sequential transform over one time step; (*c*) Direct look-ahead transform over two time steps

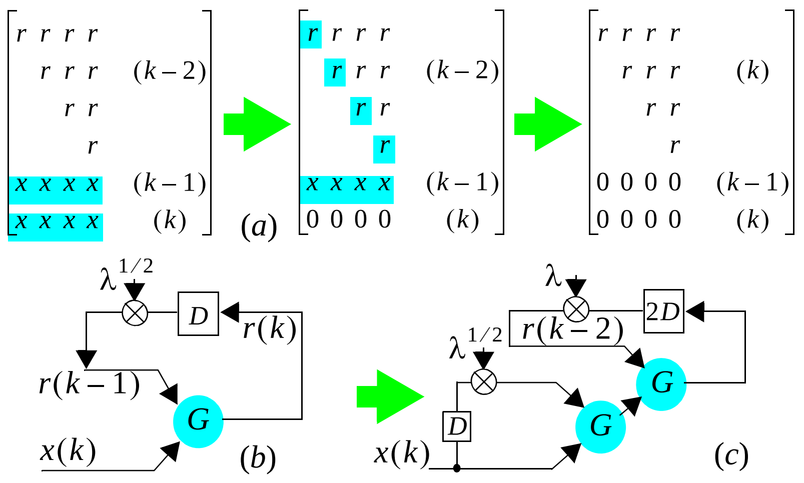


Figure: 3.15: (*a*)Annihilation-reordering look-ahead QR update procedure; (*b*) sequential transform; (*c*) ART

The annihilation-reordering look-ahead technique [47] is illustrated in Figure 3.15 (*a*). Here a sequence of Givens rotations is chosen to firstly annihilate the block data $\mathbf{x}(k-1)$ and $\mathbf{x}(k)$, and then update $\boldsymbol{R}(k-2)$ to $\boldsymbol{R}(k)$ (i.e. the diagonal r elements are updated only at the last step). This leads to the following 2-level annihilation-reordering look-ahead transformation for QR-RLS adaptive filters, this can be mathematically represented by Eq. 3.29). It can been seen in Figure 3.15 (*b*) that without increasing the complexity in the loop, the number of delay elements in the

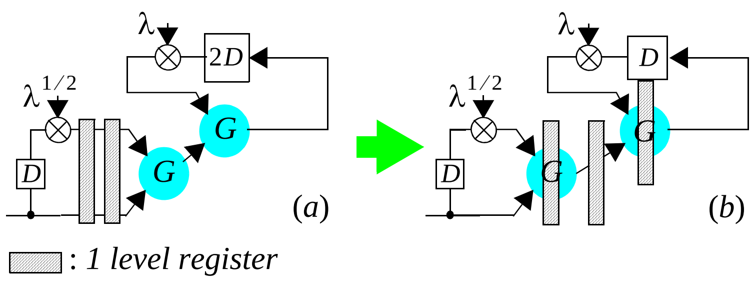


Figure: 3.16: Retiming technique for ART

feedback loop is increased from one to two. These two delay elements can then be redistributed around the loop using retiming techniques to achieve the fine-grain pipelining by the 2-level, as shown in Figure 3.16. The single Givens rotation unit outside the feedback loop is considered as a computation overhead resulting from the look-ahead transformation, hence, it can also be cut-set pipelined by 2-level registers.

$$\begin{bmatrix} r(k) \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} c_1 & s_1 & 0 \\ -s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_2 & s_2 \\ 0 & -s_2 & c_2 \end{bmatrix} \begin{bmatrix} r(k-2) \\ x(k-1) \\ x(k) \end{bmatrix} \tag{3.29}$$

Relationship with Multiply-Add Look-ahead

The above transformantion is similar to the well known simple multiply-add Look-ahead [47] procedure since both perform Look-ahead through iteration. Consider the SFG of a first-order IIR digital filter in Figure 3.17 (*a*). After applying the multiply-add Look-ahead transformation with pipelining level two, the resulting SFG is given in Figure 3.17 (*b*). The filter sample rate can be increased by a factor of two after redistributing (retiming) the two delay elements in the feedback loop (i.e. pipelined multiplier and adder trees). The SFG in Figure 3.18 (*a*) can be redrawn as shown in Figure 3.18 (*b*) (where the simple adders of Figure 3.17 are now the "*G*" Givens rotations), which is essentially similar with Figure 3.17 (*b*).

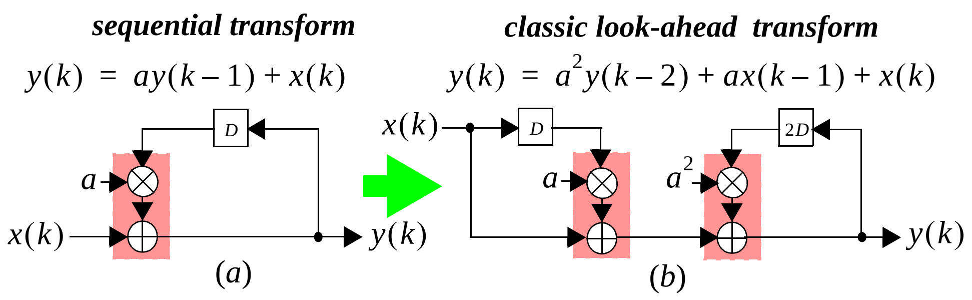


Figure: 3.17: Multiply-add Look-ahead transformation

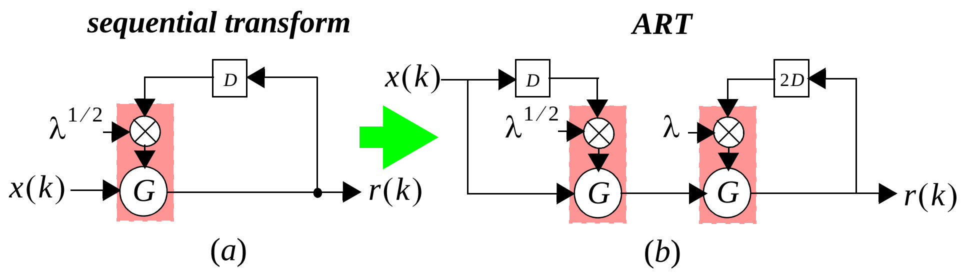


Figure: 3.18: Annihilation-reordering Look-ahead transformation

ART Real Valued QR-RLS

Regarding Figure 3.14, the recursive operations inside all the PEs (BCs, ICs & DCs) limit the throughput of the QR-RLS systolic array. To increase the sample rates, we now apply the annihilation-reordering look-ahead technique [47] to derive the concurrent real valued QRD-RLS algorithm for weight extraction. By block updating with the block size M, the real valued expression of original sequential updating Table 3.3 is transformed as follows

$$\boldsymbol{Q}(k)\begin{bmatrix} \lambda^{M/2}\boldsymbol{R}(k-M) & \lambda^{M/2}\boldsymbol{p}(k-M) & \lambda^{-M/2}\boldsymbol{R}^{-1}(k-M) \\ \boldsymbol{x}_M^T(k-1) & d_M(k-1) & 0_M \end{bmatrix}$$
$$= \begin{bmatrix} \boldsymbol{R}(k) & \boldsymbol{p}(k) & \boldsymbol{R}^{-1}(k) \\ 0 & \xi(k) & b(k) \end{bmatrix} \tag{3.30}$$

where $\boldsymbol{x}_M(k)$ is an M-by-N matrix (N is the number of weights in the adaptive

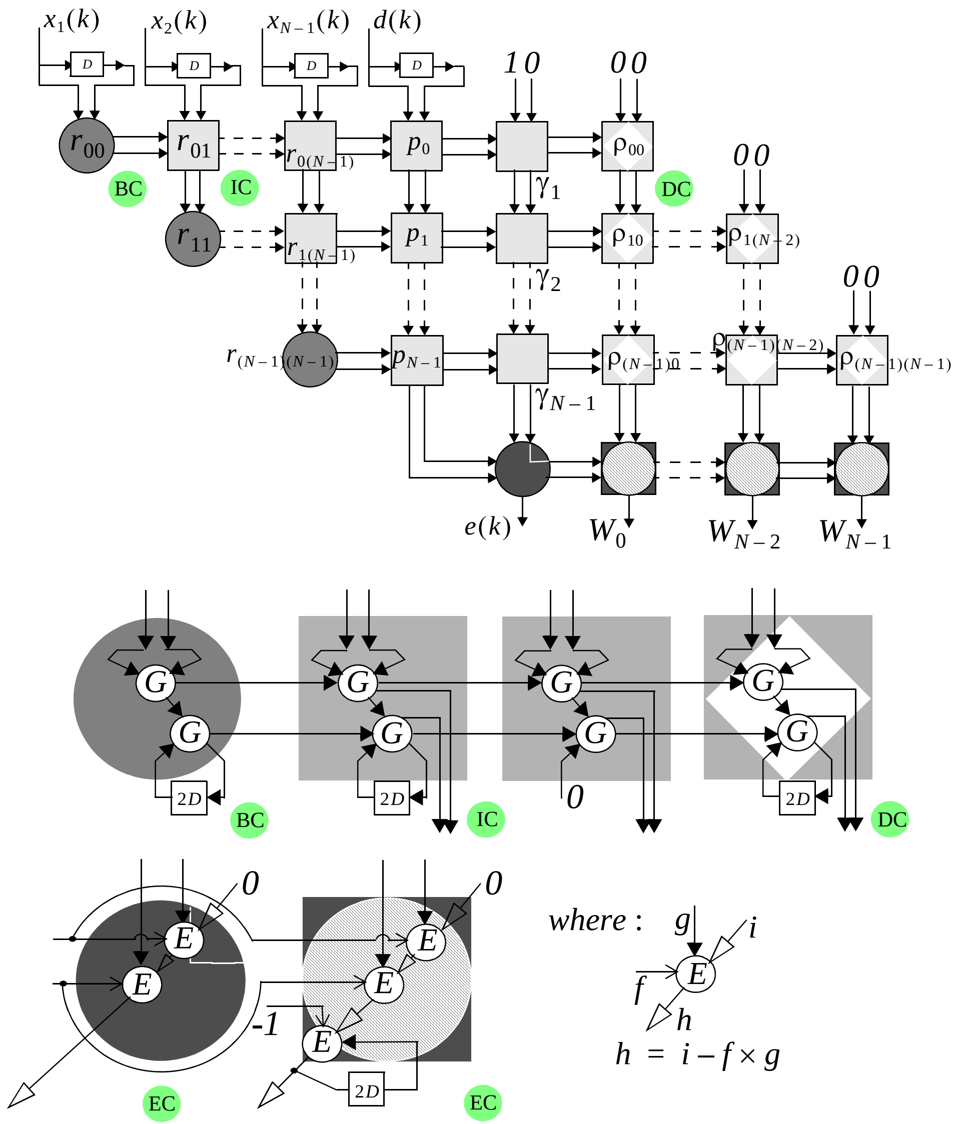


Figure: 3.19: ART updated QR-RLS ($M = 2$)

filter) defined as

$$\mathbf{x}_M(k) = [\mathbf{x}(k-M+1), \ldots, \mathbf{x}(k-1), \mathbf{x}(k)]^T$$

and $\boldsymbol{d}_M(k)$ is an M-by-1 vector defined as

$$\boldsymbol{d}_M(k) = [d(k-M+1), \ldots, d(k-1), d(k)]^T$$

In Eq. 3.30, $\boldsymbol{O}_{M \times p}$ denotes M-by-N null matrix.

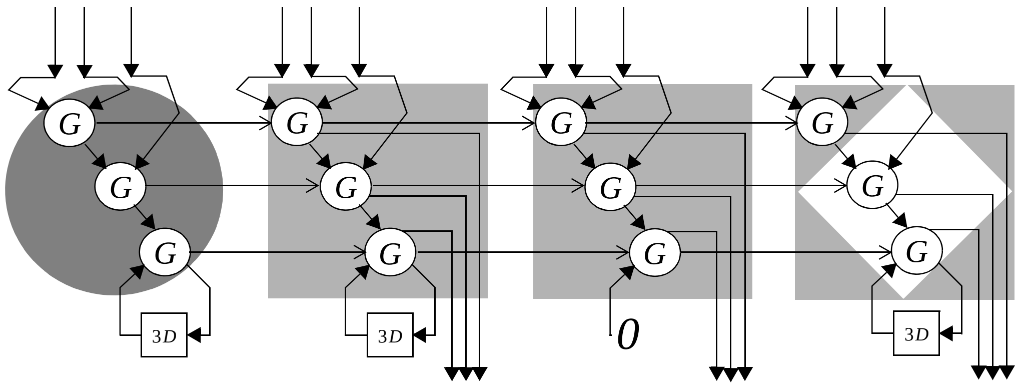


Figure: 3.20: ART updated QR-RLS ($M = 3$)

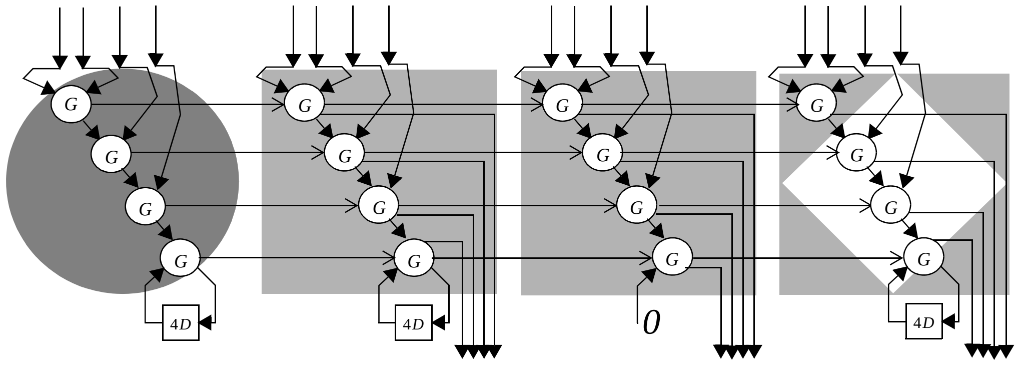


Figure: 3.21: ART updated QR-RLS ($M = 4$)

Based on Figure 3.15, the order of the Givens rotations is chosen such that the input x samples is preprocessed, and the block-data update has the same complexity as a single-data update. Hence, a 2-level pipelining (block updating size $M = 2$) topology for the CORDIC-based QRD-RLS filter is shown in Figure 3.19. As demonstrated in [47], the internal structure of each PE is also shown in the bottom half of this diagram. In comparison with the triangularisation part of the sequential updated QR-RLS systolic array, the 2-level pipelining architecture doubled the number of Givens (i.e. CORDIC) units which is linear with respect to the pipelining level. As shown in Figure 3.19, from an implementation point of view, for block updating by two, there are two circular CORDICs required by all the Givens rotaion based BCs, ICs and DCs, and the error calculator and each EC require two and three linear CORDIC units respectively. When the block updating size increases to three, the amount of needed CORDICs are three, as shown in Figure 3.20. In Figure 3.21, if the look-ahead pipelining level further rises to four, correspondingly four CORDICs are employed. Hence for a real value QR decomposition updated by the block size M, the systolic array requires M CORDICs in each PE (except the extraction cell which needs $M+1$ CORDICs). The complexity (regarding the NO. of CORDIC units per sample time) of the double triangular part is $O(MN^2)$, where N is the dimension of matrix $\boldsymbol{R}$, and M is the size of block update.

Many communication applications require the complex valued DSP arithmetic, hence it is necessary to extend ART to cope with complex valued QR-RLS array. Section 7 will present a novel Complex valued Annihilation-Reordering Look-ahead Transformation (C-ART).

3.11 Conclusion

The RLS algorithm is computationally intensive (order $O[N^2]$). The QR decomposition can be applied to RLS algorithm to avoid the inversion of the input signal's correlation matrix. The QRD-RLS implemented with the orthogonal Givens rotations is employed in this research. The advantages of this algorithm are its numerical stability and its systolic array implementation. A major drawback of the conventional QR-RLS algorithm is the back-substitution algorithm which is required for sequentially computing the weight vector. This research employs a weights parallel extraction approach known as the extended QR-RLS algorithm, which is implemented through a double triangular array without the back-substitution procedure.

The cost is too large for mapping all the PEs of the QR-RLS systolic array directly on FPGA, particularly with the increasing array size N. Therefore, a crucial challenge in design and hardware implementation of QR-RLS filtering is mapping this large systolic array to one with the lower complexity. The Fast QR-RLS algorithms with order N numerical operations per output sample have been discussed. The approaches to generate the Fast QR-RLS algorithm using lattice formulation and algorithmic engineering are also illustrated. Despite in terms of the computational complexity, the reviewed Fast QR-RLS algorithms are efficient, it is hard to implement and map to a cost-saving archiecture, moreover it can not be pipelined at the fine-grain level.

The throughputs of all the Fast QR-RLS adaptive filters are still limited by the inherent recursive update operations. To increase the throughput of QR-RLS adaptive filters, Annihilation-Reordering Look-ahead Transformation (ART) was proposed to achieve more fine-grain pipelining by retiming, to improve the clocking speed of QR-RLS implementation. However, ART was only proposed for the real valued QR-RLS system.

Chapter 4

Coordinate Rotation Digital Computer in QR-RLS

4.1 Introduction

Several methods have been proposed for the computation of QR decomposition. Givens rotation [21] has the advantage over Householder transformation [49] etc that it can be more parallelised and therefore more appropriate for the hardware implementation. The hardware implementation of Givens rotation requires the core operations of fixed point multiply-accummulate, square root and also divide.

Despite the QR triangularisation method has excellent numerical properties and is hence the favoured method for fixed point implementation, according to Figure 3.7, the direct arithmetic to calculate sine and cosine function in QR Boundary Cell (BC) requires both the division and square root operation, the implementation complexity is high.

One low complexity technique that performs the Givens generation and rotation is the COordinate Rotation DIgital Computer (CORDIC) algorithm, which was first developed by Volder [50] in the 1950s before Walther [51] extended the work in the 1970s. It is an iterative method which is used to calculate many trigonometric and algebraic functions [50]. CORDIC performs a series of micro-rotations with an architecture comprised almost entirely of shift-add operations, and hence results in an efficient hardware design.

This chapter will analysis the throughput vs area performances of three CORDIC architectures: pipelined parallel, non-pipelined parallel and serial structure. A special senario using FPGA on-chip DSP48 slice in serial CORDIC will also be discussed.

The precision of CORDIC can be estimated by the ‘rule of thumb’ firstly presented by Walther [51] [52]. This rule suggested the CORDIC arithmetic obtains about 1 bit of accuracy per iteration ($b+1$ iterations for b-bit precision). Beyond Walther's original work, the assessment of Overall Quantisation Error (OQE) was presented by Yu Hen Hu [53] in the 1980s to give the numerical accuracy achievable by n-rotations and b-bits for some aribitary setting.

4.2 CORDIC

All the boundary, internal and downdating cell of the QR-RLS array can be implemented with the CORDIC processors, which is suitable to execute Givens rotation in fixed point. CORDIC is an iterative algorithm for calculating trigonometric functions such as sine and cosine etc. It requires shifts and adds but needs no multiplications, so it requires less hardware resource.

The generalised CORDIC equation of circular Givens rotation is shown in [50] [54]:

$$\begin{aligned} x_{i+1} &= x_i - d_i(2^{-i}y_i) \\ y_{i+1} &= y_i + d_i(2^{-i}x_i) \\ z_{i+1} &= z_i - d_i \operatorname{atan}(2^{-i}) \end{aligned} \tag{4.1}$$

where the index i denotes the ith iteration. Also, d_i is known as the decision factor and used to control the direction which the vector is rotated at each iteration and hence takes the value of ±1 — If the vector is rotated in clockwise, d_i equals to -1, otherwise, $d_i = 1$. Circular rotation CORDIC has two modes: vectoring and rotation. The vectoring mode CORDIC, represents the BCs in Figure 3.8 and Figure 3.10, generates the magnitude of vector[1] (x, y) and the corresponding phase z (also known as Given generation), and the rotational mode performs the Givens rotation of the given angle z for the ICs and DCs. The decision factor d_i under both modes are given as follows:

$$\begin{aligned} \text{Vectoring Mode}: \ d_i &= -sign(x_i y_i) \\ \text{Rotation Mode}: \ d_i &= sign(z_i) \end{aligned} \tag{4.2}$$

1. Here (x, y) and d represent a vector and CORDIC decision factor respectively, rather than the input signal $x(k)$, output signal $y(k)$ and desired signal $d(k)$ of an adaptive filter.

The decision factor d_i with each rotation affects the accumulative angle that is rotated. Arbitrary angles can be rotated in the range $-99.7 \le \theta \le 99.7$. The sum of all angles obeying the law $\tan\theta_i = 2^{-i}$ is 99.7 [50]. For angles outside this range, the quadrant mapping can be used to convert the desired angle into one within the range, and this mapping needs to be corrected by the quadrant correction at the output of CORDIC.

According to [50], the resultant x_{i+1} and y_{i+1} values of CORDIC procedure are always scaled by a factor K_i. If the iteration number n is known then the scaling factor K_n can be precomputed and correct the final values of x_n and y_n by multiplying them by $1/K_n$. The scaling compensation employs binary shifts and adders to form a constant multiplication, which could be realised by binary shift-add [55].

$$\begin{gathered}
K_n = \prod_n 1/(\cos\theta_i) = \prod_n (\sqrt{1+\tan^2\theta_i}) = \prod_n (\sqrt{1+2^{(-2i)}}) \\
K_n \to 1.6476 \text{ as } n \to \infty \\
1/K_n \to 0.6073 \text{ as } n \to \infty \\
n = \text{number of iterations}
\end{gathered} \tag{4.3}$$

Figure 4.1 (*a*) shows the signal flow graph of the standard unpipelined parallel CORDIC. The architecture of a single CORDIC iteration cell is shown in Figure 4.1 (*b*). Series of such butterfly-like cross-additions (with the fast carry chain mentioned in Chapter 2.5.2) are employed to update the current estimate of the required function. Note that the iterations number i is a function of the desired numerical accuracy, the 'rule of thumb' presented by Walther [51] suggests that each the micro-rotation refines the estimate by approximately 1-bit of precision. The critical path of the unpipelined CORDIC is (n iteration cells + *one* scaling compensation cell + *one* quadrant mapping + *one* correction). The concept of critical path is the longest propagation delay of the circuit. The sampling rate of this unpipelined CORDIC is one new input/output per clock cycle. So the maximum clock rate should equal to the sampling rate. However the setback is the large hardware area and long critical path highlighted in Figure 4.1

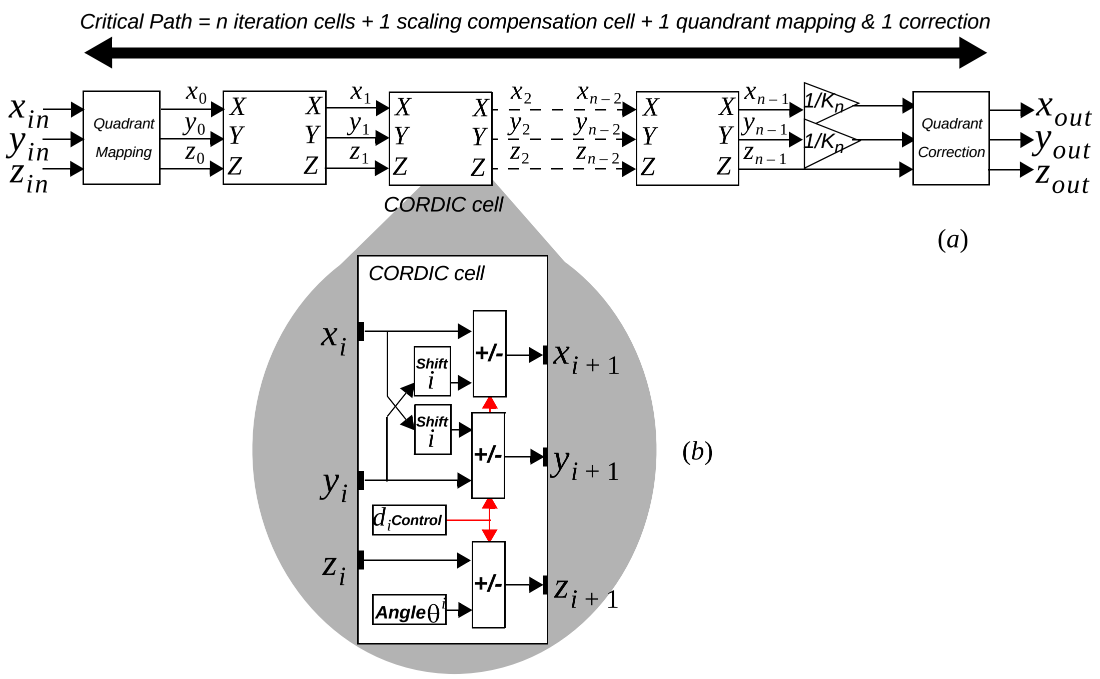


Figure: 4.1: (*a*) Unpipelined parallel CORDIC (*b*) Fine grain level of a single CORDIC microrotaion (*a*).

4.3 Squared Givens Rotation (SGR)

Another method of implementing Givens transform on hardware is the Squared Givens Rotation (SGR) algorithm [56]. Here the Givens algorithm has been manipulated to remove the need of square root operation, and leads to different architectures that do not use CORDIC but require division operations. In contrast with CORDIC, SGR requires dividers (hence always suggest floating point arithmetic [57] [58]) and essentially operates on the correlation matrix rather than the data matrix, and requires a longer fixed point wordlength compared to the standard Givens. As such there is a difference between the numerical properties of SGR and CORDIC, this thesis will not make a critical comparison between them.

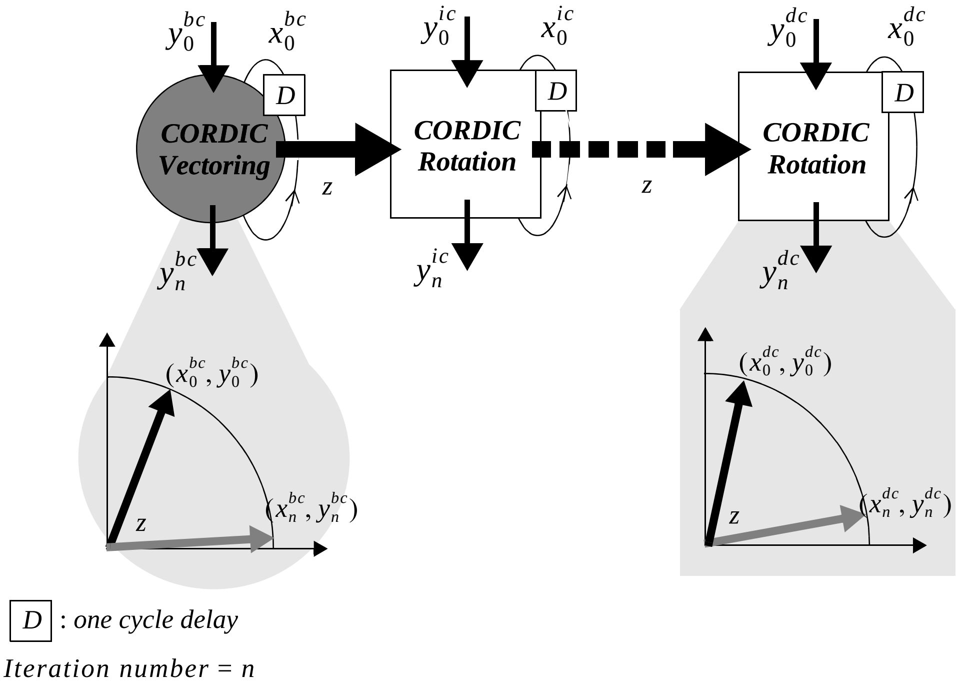


Figure: 4.2: Signal transfer between the real valued BC, IC and DC

4.4 CORDIC based QR-RLS Processing Elements

According to Section 3.6, the QR-RLS boundary cells execute Givens generation (namely generating the trigonometric functions), internal and downdating cells operform Givens rotation with the received trigonometric functions from BCs. Both Givens generation and rotation can be implemented on hardware by CORDIC arithmetic. Each boundary cell of QR array requires the vectoring mode CORDIC, and the internal and downdating cells require the rotation mode CORDICs. For the real valued QR-RLS array, the boundary cell needs to convey the generated phase to the internal cell, as illustrated in Figure 4.2. The dark vectors represent the input vectors of both boundary and internal cells. After the real Givens generation, namely n CORDIC micro-rotations, the dark vector are rotated to the positions of the grey ones and generate the phase information z. The ICs and DCs receive the angle z and rotate by the same degree. The decision factor d is chosen as a function of whether angle z

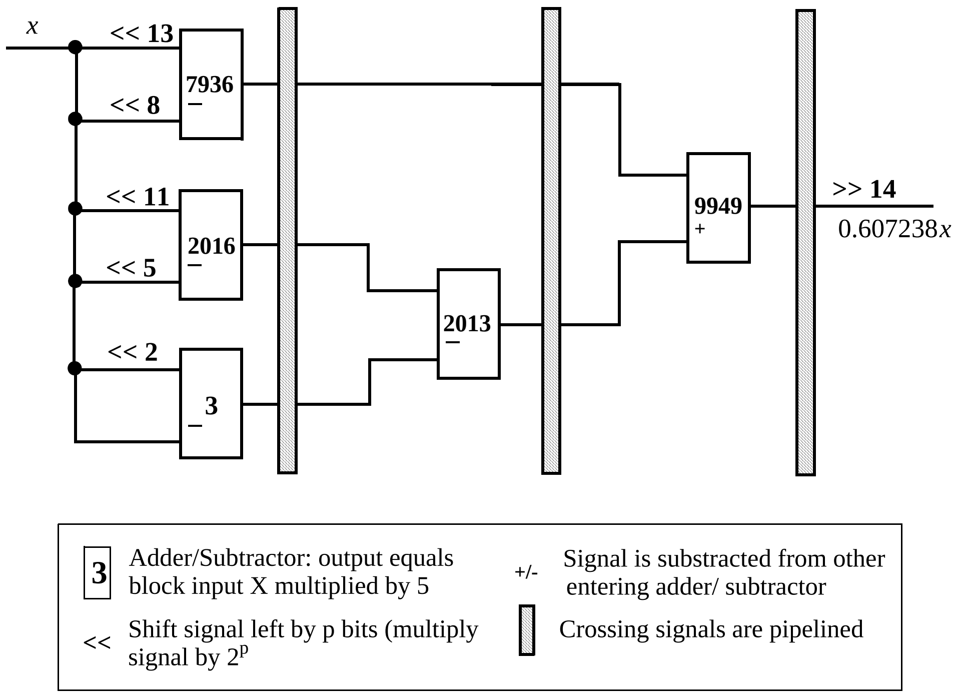

x
<< 13
<< 8
7936
−
<< 11
<< 5
2016
−
<< 2
3
−
2013
−
9949
+
>> 14
0.607238x
3
Adder/Subtractor: output equals block input X multiplied by 5
+/-
Signal is substracted from other entering adder/ subtractor
<<
Shift signal left by p bits (multiply signal by 2^p
Crossing signals are pipelined

Figure: 4.3: CSD coding based constant multiplier

is negative or positive.

4.4.1 CORDIC Scale Compensation

The CORDIC scaling factor K_n can be efficiently compensated by appropriate binary shifters and adders to form a constant multiplication. The binary shifts could be hardwired with adders. Since the binary shift costs no hardware resource, the corresponding ‘optimizing’ strategy is to reduce the complexity of constant multiplier by minimizing the number of adders. Rather than the conventional binary representation, the Canonical-Signed Digit (CSD) coding [12] is employed in [55] to produce the constant multiplier. Considering the instance when $n = 16$ CORDIC iterations and 14 binary bits are chosen for the fractional part of multiplier output, the constant coefficient is $1/k_{16} = 0.607238$. A 3 level pipelined shift-add based CSD multiplier is shown in Figure 4.3..

The Minimised Adder Graph (MAG) algorithm, specified by Dempster & Macleod

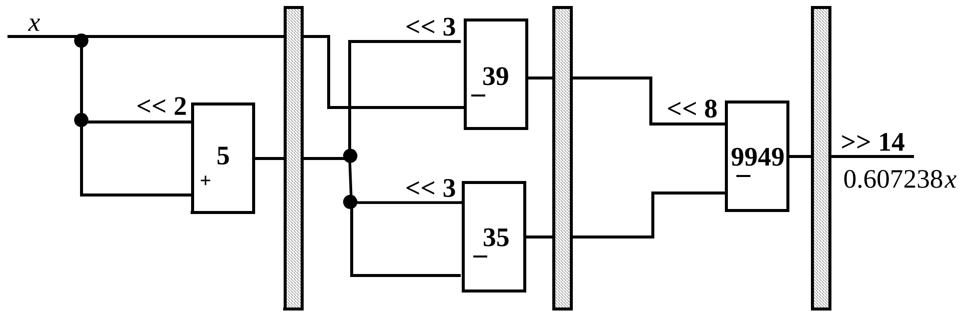


Figure: 4.4: RSG algorithm based constant multiplier

[59], is a good choice for the constant multiplication graph generation. Better than CSD, MAG can yield the minimum number of adders [59]. MAG generates all the constant multiplication graphs for the supplied constant, and uses a series of iterative searches to find the lowest cost generated graph — the least number of adders.

However, MAG is more suitable for the ASIC implementation rather than FPGAs, because FPGAs have predefined architectures and the FPGA area is measured in slices (for Xilinx). According to Chapter 2, compared with an ASIC design, each pipeline stage results flip-flops and hence utilizes FPGA slices. So the multiplier block synthesis algorithms for low FPGA area should minimise the consumption of slice rather than that of adder.

Reduced Slice Graph (RSG) [60] synthesises the low FPGA cost multiplier block. RSG combines the graph data generated by MAG as part of the multiplier block synthesis process. Modification and extensions were made to original MAG algorithm that enhance its functionality. Figure 4.4 shows a 3 level pipelined shift-add based RSG multiplier, which saves one adder from the CSD representation.

4.4.2 Throughput of CORDIC in a Recursive Loop

The pipelined CORDIC design has pipeline registers inserted between micro-rotations, to break the long critical path highlighted in Figure 4.1 (*a*). Figure 4.5 shows the pipelined CORDIC structure. After the initial latency of the circuit has been

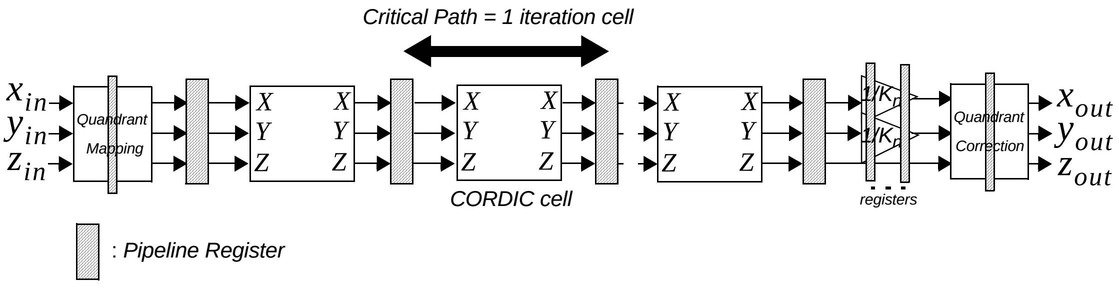


Figure: 4.5: Pipelined parallel CORDIC

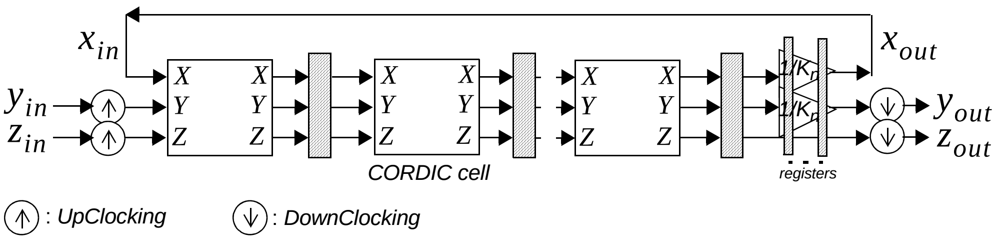


Figure: 4.6: Pipelined CORDIC based real valued QR PE

absorbed, the sampling rate is still one input/output per clock cycle.

According to Figure 4.2, each real valued QR-RLS PE requires one CORDIC unit inside a recursive loop for RLS filtering. A pipelined CORDIC in Figure 4.6 will change the input-output behaviour of the recursive loop unless the clock rate of CORDIC is higher than the rate of input samples, as shown in Figure 4.6. The calculated magnitude of vector (x_{in}, y_{in}), i.e. x_{out} in Figure 4.6, needs to be fed back as the next iteration's inputs x_{in}. Since the magnitude x_{out} is always positive, So the rotated angles for all the BCs, ICs and DCs are always in the 1st and 4th quadrant, and the CORDIC quadrant mapping and correction operations are not needed inside the real valued QR-RLS PEs. For a single input data channel, a sample will enter the pipeline and clock through each stage with nothing following it until it has passed through completely. This means that the majority of the logic is redundant for the majority of the time. The pipeline will never fill up because a new sample must wait on the result of the previous one before it can enter the pipeline. Hence the CORDIC pipelining can not improve the throughput of design. Assuming there are n micro-rotations in the CORDIC operation, despite the introduction of pipeline registers

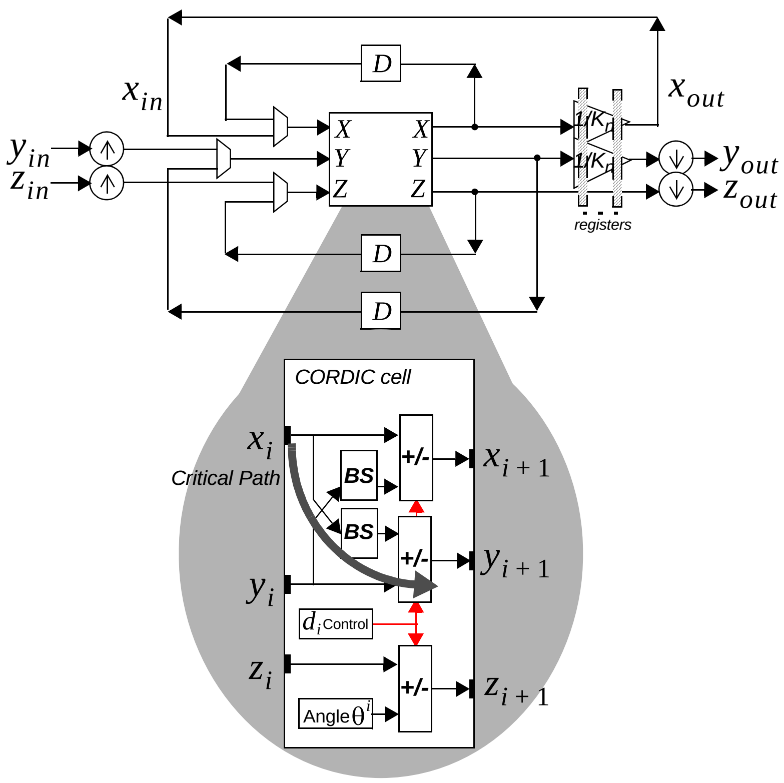


Figure: 4.7: Serial CORDIC based real valued QR PE

reduces the critical path to its $1/(n+f)$, where f is the number of pipeline registers for scaling compensation. The data rate should be $(n+f)$ new inputs/1 output per $(n+f)$ clock cycles.

An alternative architecture is the serial CORDIC in Figure 4.7. Similar to the pipelined CORDIC, the serial CORDIC can generate 1 output value per $(n+f)$ micro-rotations. So the sampling rate is $(n+f)$ new inputs/1 output every $(n+f)$ clock cycles.

In the serial CORDIC design, bit shifting is achieved by the barrel shifter (BS). As shown in Figure 4.8, each seperated bit of the 'i' signal drives one 2-to-1 multiplexer. For instance when i = "01010", i.e. decimal 10, the input x and y data follows the path highlighted by the grey arrow.

The two Barrel Shifters (BS) are the most complex components of the serial CORDIC in Figure 4.7. In [12], it is suggested that the two barrel shifters can be

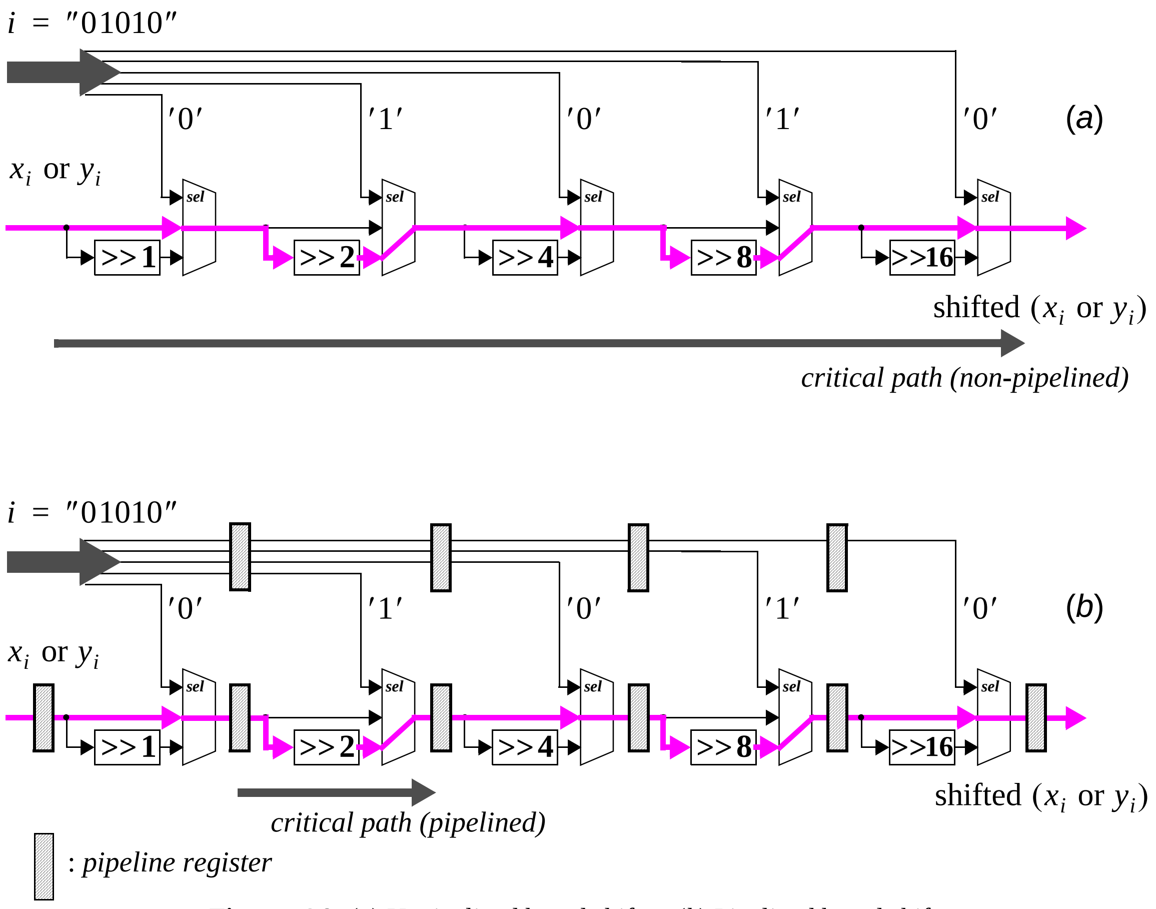
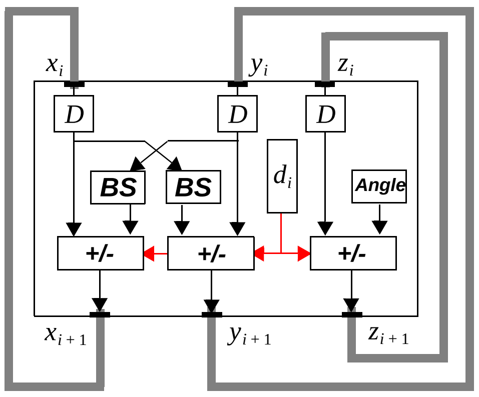


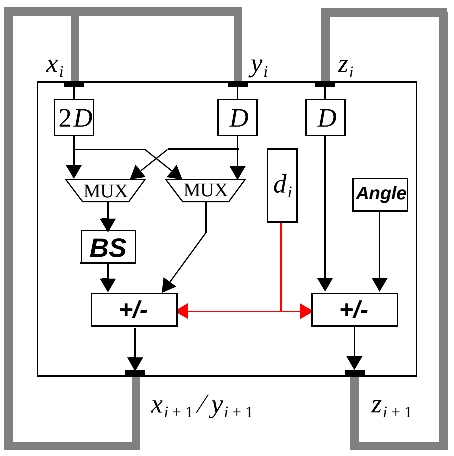
Figure: 4.8: (*a*) Unpipelined barrel shifter; (*b*) Pipelined barrel shifter

mapped to a single one, using two multiplexers and one adder/subtractor, as shown in Figure 4.9 (*b*). Despite this method can further reduce the hardware usage of serial CORDIC, but the throughput is reduced by the time-sharing scheme. According to [12], for 13-bit implementation, the single BS based architecture only saves 15% cost, with the 67% drop in the throughput. Hence, this architecture only features the designs whose speed requirement is not critical.

Performing the k-fold loop rolling on the unpipelined circuit in Figure 4.1 yields the circuit with k micro-rotations in Figure 4.10. This architecture performs the same Givens operation in $(n/k)+f$ clock cycles. Although this partial rolled architecture results in a simpler barrel shifter, it has a longer critical path through k iteration cells instead of one.



(*a*)



(*b*)

Figure: 4.9: (*a*) Dual-BS serial CORDIC; (*b*) Single-BS serial CORDIC

The critical path of the serial CORDIC cell is highlighted by the grey line in Figure 4.7. In contrast to the general hardwired bit shifter in Figure 4.1 (*b*), the barrel shifter will increase the critical path latency due to the cascade of LUT based multipliexers, which was discussed in Chapter 2.5.2. To reduce the propagation delay of BS, the barrel shifters also need to be pipelined. By employing the pipelined barrel shifter in the serial CORDIC, its maximum clock rate reaches just slightly lower than that of the pipelined design in Figure 4.6, due to the overhead of the additional pipeline registers. The serial CORDIC based PEs with the pipelined barrel shifters can generate 1 output value during $(n \times s) + f$ clock cycles, where s is the number of pipeline stages. So the

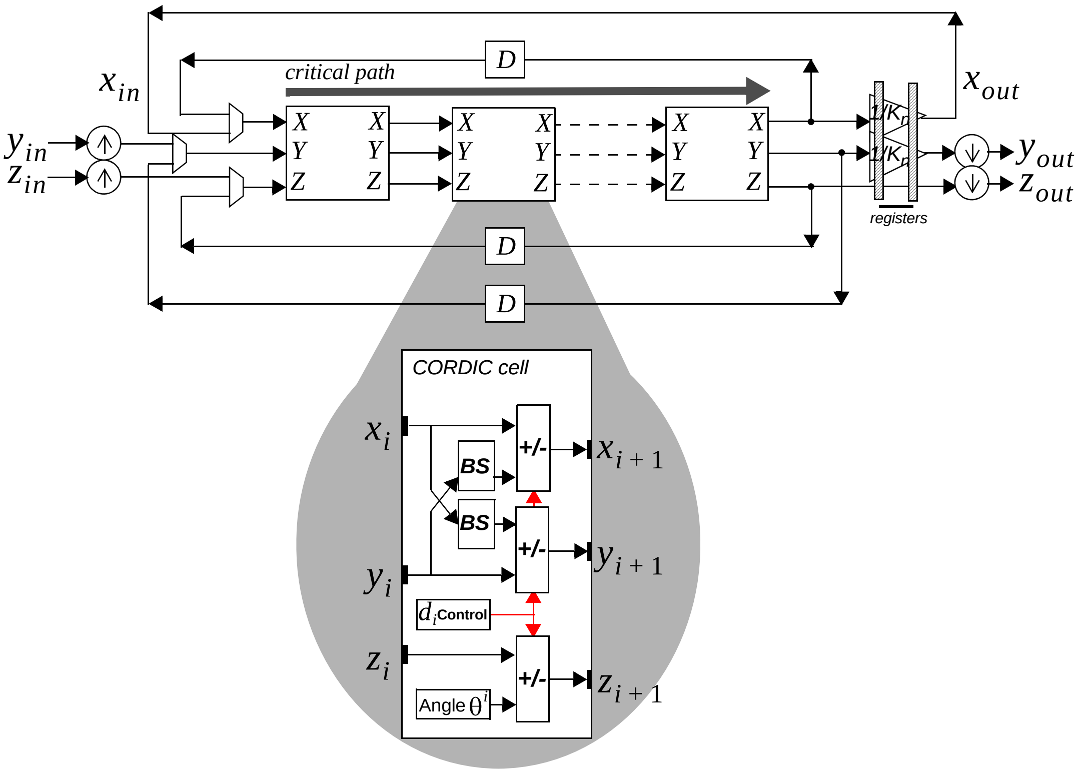


Figure: 4.10: K-fold loop rolled CORDIC

sampling rate should be $(n \times s) + f$ new inputs/1 output per $(n \times s) + f$ clock cycles.

4.4.3 DSP48 based Serial CORDIC

The long propagation delay of barrel shifter results in the low throughput of serial CORDIC based QR-RLS PEs, an improved solution is to replace the single CORDIC micro-rotation with the multiply-add function of the DSP48 slices, which was addressed in Chapter 2.

According to Figure 2.5, when the iteration number $n < 18$ and the x and y width $n' \leq 18$ bits, the i-bit barrel shift is replaced by one $2^{-i} \times$ operation, where the 2^{-i} sequence is pre-stored in the FPGA on-chip Block RAM. In Figure 4.11, the 18×18 bit multiplier is enough to represent a 17-bit shifting 2^{-17}. According to [51], this indicates less than 17 CORDIC micro-rotations, due to CORDIC obtaining 1-bit

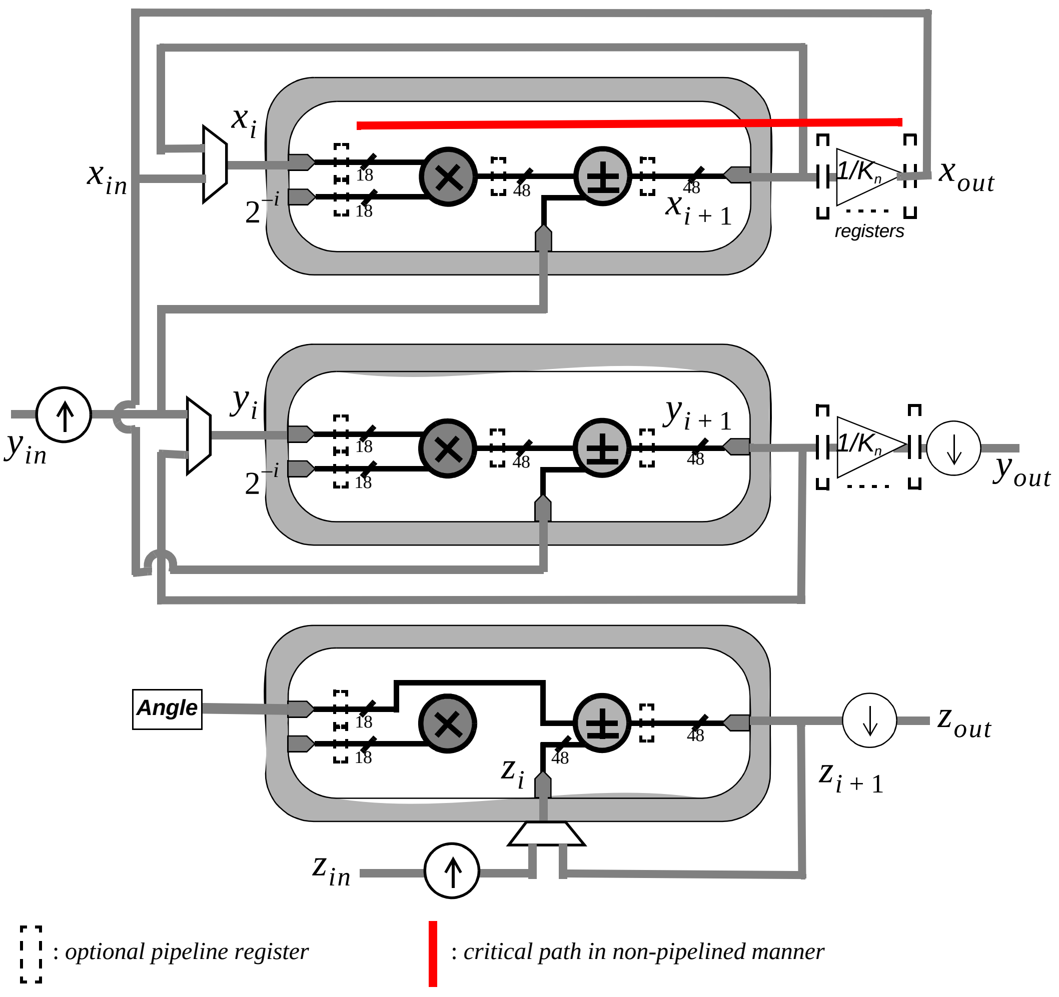


Figure: 4.11: Using DSP48 in Serial CORDIC (x, y width $n' \le 18$-bit)

precision per iteration. Then the input x and y in Figure 4.11 will add/subtract the partial products from the multiplier 2 and 1. Hence in this case, the single CORDIC cell costs 3 DSP48 Slices, and the critical path is the total propagation delays of one multiplier, one adder and one scaling compensator which can also be implemented using DSP48s.

When the iteration number $n < 18$ and the x and y wordlengths are between 18 and 36 bits, two DSP48s execute the ‘bit-shift’ work by spliting and expanding x or y into two 18-bit parts. As is the case shown in Figure 4.12, the single CORDIC iteration cell contains 5 DSP48 Slices. When no pipeline registers are used in the DSP48s, the

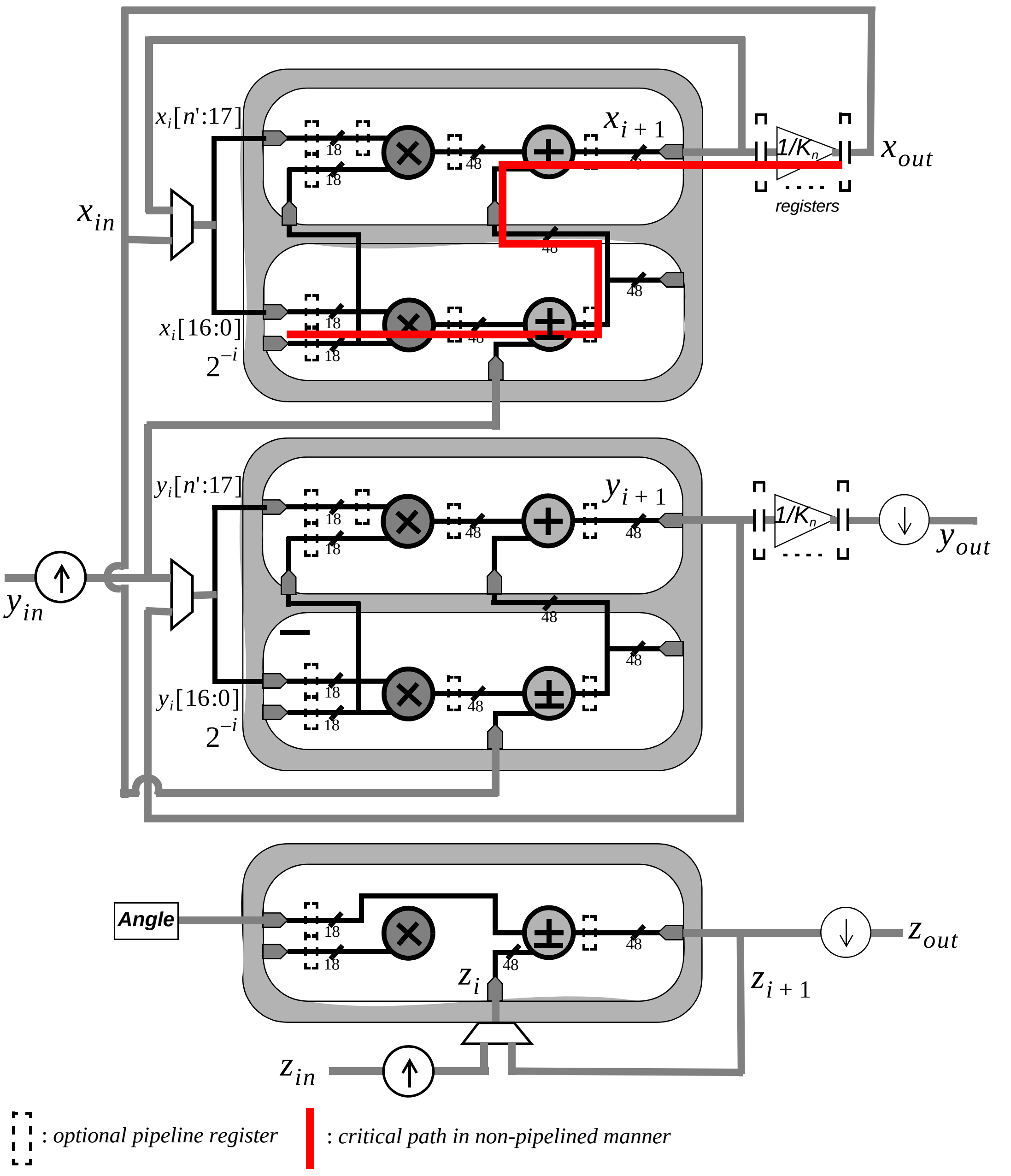


Figure: 4.12: Using DSP48 in Serial CORDIC (18-bit < (x, y width $n' \leq$ 36-bit))

critical path should be the sum of the propagation delays of one multiplier, two adders and one scaling compensator.

Similar to the serial CORDIC based QR PE in Figure 4.7, the sampling rate of pipelined DSP48 based ones in Figure 4.11 and Figure 4.12 are still '*CLK rate/*

$(n \times s + f)$ '. But the full pipelined DSP48 slices can be driven at a fast 500 MHz clock rate, so the DSP48 solution can improved the throughput of serial CORDIC.

FPGA synthesis results suggest that, for less than 25-bit wordlength, an unfolded pipelined CORDIC based PE can be driven at 250MHz clock rate, with the sampling rate '*CLK rate*/ $(n + f)$ '. Despite the clock of pipelined DSP48 slices can go faster, there are $s > 2$ pipeline stages inside DSP48, as shown in Figure 4.11 and Figure 4.12. The maximum sampling rate of DSP48 based PE is $500\text{MHz CLK} / (n \times s + f)$, which is smaller than that of the pipelined parallel CORDIC — $250\text{MHz CLK} / (n + f)$. Hence both the DSP48 and LUT based serial CORDICs are not the proper solutions for high throughput QR-RLS application.

4.5 Other Existing High Speed CORDIC Solutions

The throughput of CORDIC based QR-RLS PEs is limited by the scaling compensation and the long critical path inside the recursive update loop, which can not be directly pipelined to avoid the change of algorithm's input-output behaviour. Many previous researches focused on developing an improved CORDIC in two fields : first, changing the CORDIC algorithm and architecture; second, based on the standard CORDIC arithmetic, but improving the Overall Quantisation Error (OQE) assessment method to produce the desired numerical performance with the minimum number of microrotations [58]. Both topics are demonstrated as follows:

4.5.1 Low Latency CORDIC Algorithms and Architectures

To reduce the inherent latency, [61] proposes a scaling free CORDIC algorithm. Compared with the traditional CORDIC, the scaling free version produces the lower latency at the expense of the double cost for each iteration cell. Another algorithm discussed in [62] and [63] applies the Taylor Series expansion into the CORDIC, consequently reduces the number of iterations, meanwhile avoids the scaling operation, but this algorithm leads the extreme large angle quantization error compared with that of the conventional CORDIC, and this method is more suitable for the rotation mode CORDIC.

The Angle Recording (AR) technique [64] is a useful approach to speed up the operation of CORDIC. Compared with the conventional CORDIC algorithm, AR technique reduces the number of CORDIC iterations and the angle quantization error significantly. However, this method is only suitable for the rotation mode CORDIC. In [65] a novel On-Line Mixed-Scaling-Rotation (MSR) - CORDIC is investigated. MSR-CORDIC executes the vectoring mode in Boundary Cell (BC) of QR-RLS systolic array. Moreover, the MSR-CORDIC, presented in [66], is only able to conduct the rotation mode in Internal Cell (IC). The iteration number of MSR-CORDIC can be minimized to result in an extremely low latency and also be able to avoid the overhead of scaling operation. However, the low latency of on-line MSR-CORDIC is achieved at the expense of 18 adder/subtractors per iteration cell [65]. Although the increased hardware cost can be parially compensated by the reduced iteration number, the overall hardware utilization is still unacceptable for the low cost applications. Furthermore the defference in BC and IC's architecture [65] makes them difficult to map together.

4.5.2 Assessing the Overall Quantisation Error

The investigation around the bit error is often used to find a CORDIC design that gives the desired level of accuracy to maintain the algorithm integrity. By choosing the desired accuracy d_{eff} and the number of fractional bits b, the CORDIC arithmetic is given a number of iterations n [51]. The research on trading the CORDIC's pipelining latency against precision could be a vital approach to minimize the execution time of CORDIC based system. In [51] [67], it is suggested that the CORDIC implementation obtains b-bit precision under $b + 1$ iterations, with $\log_2 b$ additional fractional bits are added onto the x and y path. While this law is only derived from the round-off error of fixed point system, without considering the angle quantisation error caused by CORDIC iterative approximation [51].

$$OQE = \varepsilon_a + \varepsilon_r \qquad \varepsilon_a = a_1(n-1)|v(0)|$$
$$d_{eff} = -\log_2 OQE \tag{4.4}$$

An algorithm for estimating the Overall Quantisation Error (OQE) was developed by Hu [53]. As described in Eq. 4.4 the OQE is made up of two distinct errors. The first part is the Approximation Error (AE), ε_a, which is due to the quantised representation of a CORDIC rotation angle by finite numbers of elementary angles. The second one is the Rounding Error (RE), ε_r, which originates from the finite precision arithmetic.

In Eq. 4.4, $a_1(n-1)$ represents the final quantised rotation angle, which is approximated as 2^{-n+1}. $|v(0)|$ is the maximum value that the magnitude can take. The research in Strathclyde university [68] has proved that Hu's AE estimation results in the underestimation of the number of effective fractional bits for small n. So only the large iteration number have to be employed, despite resulting the increase of CORDIC execution time. To overcome such a drawback, reference [68] presents an improved but vectoring mode only equation which is more accurate in predicting the Approximation Error ε_a.

$$\varepsilon_a = |v(0)| - |v(0)|\cos\delta \tag{4.5}$$

where $\delta = a_1(n-1) = tan^{-1}(2^{-n+1})$.

To see the difference between the improved algorithm and the original version proposed by Hu, both are plotted in Figure 4.13 for a series of n, b and $|v(0)|$. With the increase of $|v(0)|$, AE also grows. And RE reduces with the increase of bit-width b of x and y path. Figure 4.13 shows that Hu's AE estimation equation has a large error which consequently causes the underestination of effective fractional bits d_{eff}.

AE causes the bit-width increase of x and y path. A proper iteration number n need to be chosen when the RE is absolutely dominant than the AE, i.e OQE is only determined by the RE, otherwise the low iteration number will lead to the large hardware cost. In Figure 4.13, when n is set in the range 12 to 22, the assessed AE from

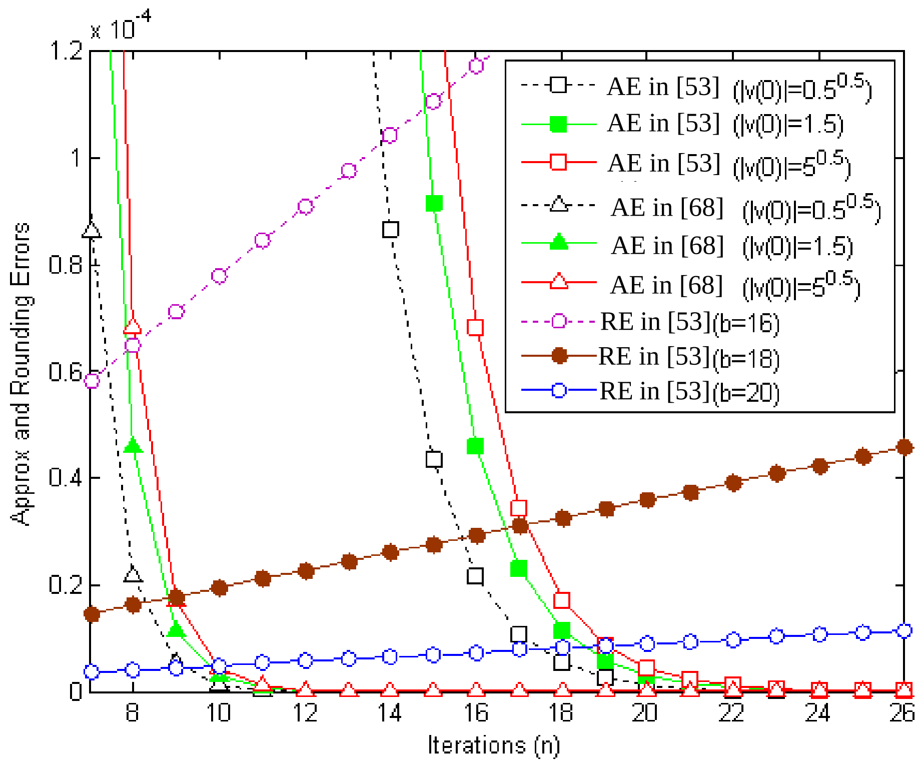


Figure: 4.13: Approximation Error vs Rounding Error

Hu's algorithm plays an important role in the OQE. In the same circumstance, the AE estimated by the improved algorithm is too small to influence the d_{eff}. However, the improved AE estimation algorithm in [68] is not compatible with the rotation mode CORDIC, which is required in the internal/downdating cells of QR-RLS systolic array.

4.6 Conclusion

In this chapter, the Processing Elements (PEs) of QR-RLS systolic array are realized by CORDICs rather than SGRs. Vectoring and rotation mode CORDICs can be easily accomplished with only a few shift-and-add operations. The RSG reduces the complexity of the CORDIC scaling compensation by the simple shift-add operations, which ultilises the minimum amount of slices. To fulfill the essential speed requirement of the next generation wireless communication, designing the low latency CORDIC based PEs becomes a vital challenge of this research.

This chapter has described the architectures of the three types of CORDICs : unpipelined parallel, pipelined parallel and serial structures. The corresponding ‘speed-area’ performances of all three types of CORDICs have been analyzed. Pipelined CORDIC results in the high clock rate, but the introduction of pipeline latency in the QR recursive loop will harm the throughput. A special Xilinx DSP48 slice based serial CORDIC is also presented and compared with the architectures basing on the LUTs. However, for the QR-RLS application, both the DSP48s and LUTs based serial CORDICs result in the lower throughput than those of the parallel CORDICs.

This chapter has also reviewed existing low latency CORDIC solutions, and of particular interest two potential ‘speed up’ schemes: first method is changing the CORDIC algorithms/architectures to reduce the iteration number or result in a novel scaling-free architecture; second, the assessment of Overall Quantisation Error (OQE) reduces the number of CORDIC microrotations. However, none of the existing low latency CORDIC schemes are suitable for QR-RLS system.

Chapter 5

Pipeline-Interleaving Coarse Angle CORDIC

5.1 Introduction

The CORDIC arithmetic has been widely implemented as part of the computational requirements of the well known QR-RLS algorithm. In this chapter, a new architecture of CORDIC is presented, this new CORDIC is easily pipelinable and can be used to implement both the Givens generations and Givens rotations associated with the QR update. This new architecture improves the throughput of processor-like QR-RLS architecture in Section 3.8.2.

5.2 Coarse Angle Rotation Mode CORDIC

According to the circuit implementation of conventional CORDIC arithmetic in Eq. 4.1, two subcircuits are required for the rotation of point (x_i, y_i) and a third to keep track of the corresponding angle z_i for an n iterations CORDIC (where i = 0, 1... $n-1$) [54]. As illustrated in Section 3, the vectoring mode CORDIC within the real valued BC generates the angle z, and the IC/DC uses rotation mode CORDIC to perform a Givens rotation of the same angle z. Since the x and y parts are always necessary for the Givens generation and rotation, a new CORDIC/QR architecture is presented next in which the z angle accumulation of Eq. 4.1 can in fact be discarded.

The decision factor d_i conveys the direction of angle rotation at each iteration (i.e. 1 or -1). In Figure 5.1, during the first micro-rotation of CORDIC, the input vector (x_0^{bc}, y_0^{bc}) of the real valued BC is rotated by $45°$ clockwise, and the d generated is -1. When d is conveyed to the internal cell, the corresponding vector rotates $45°$ angle correspondingly. After three CORDIC iterations (and therefore three d_i values are

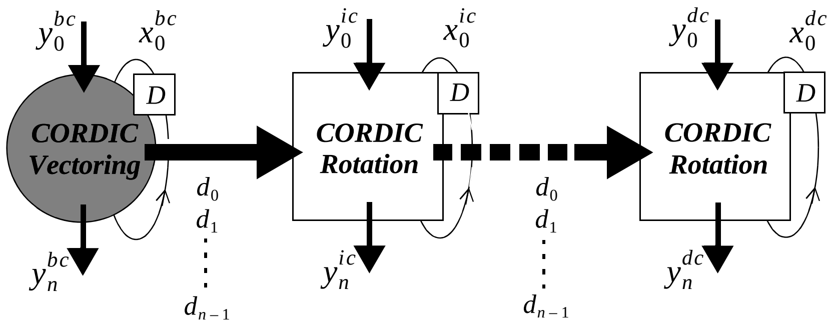


Figure: 5.2: Signal transfer between the real valued BC and IC/DC

transfered from BC to IC), the total angle generated by BC is $45° + 26.6° – 14° = 57.6°$, and for the IC, same angle is rotated. The angle derived in the BC is the same with those in the ICs and DCs, hence the d_i sequences are the same. Therefore it is possible to perform Givens rotations of the same angle in different types of PEs in a QR-RLS array without explicitly computing the angle, but rather convey the direction d of the various CORDIC coarse angle values.

The interconnection of the proposed coarse angle rotation CORDIC is illustrated in Figure 5.2. The link between both modes CORDIC is the *decision factor* d of each iteration. The removal of the z accumulator equation leads to a hardware resource reduction and produces a more regular array, and also maintains the identical numerical performance as the standard method, namely this hardware saving is independent of wordlength and number of CORDIC iterations is exactly the same [69]. In contrast to Eq. 4.1, now the iteration cells in either BCs or ICs/DCs principally requires two bit-shifters and two adder/subtractors to calculate the x and y, hence the coarse angle rotation mode PEs can be easily mapped onto a single processor-like PE. A further advantage of this coarse angle rotation mode CORDIC is that the PEs including BCs, ICs and DCs can all be mapped onto a single processor-like PE. This results in an area optimised architecture.

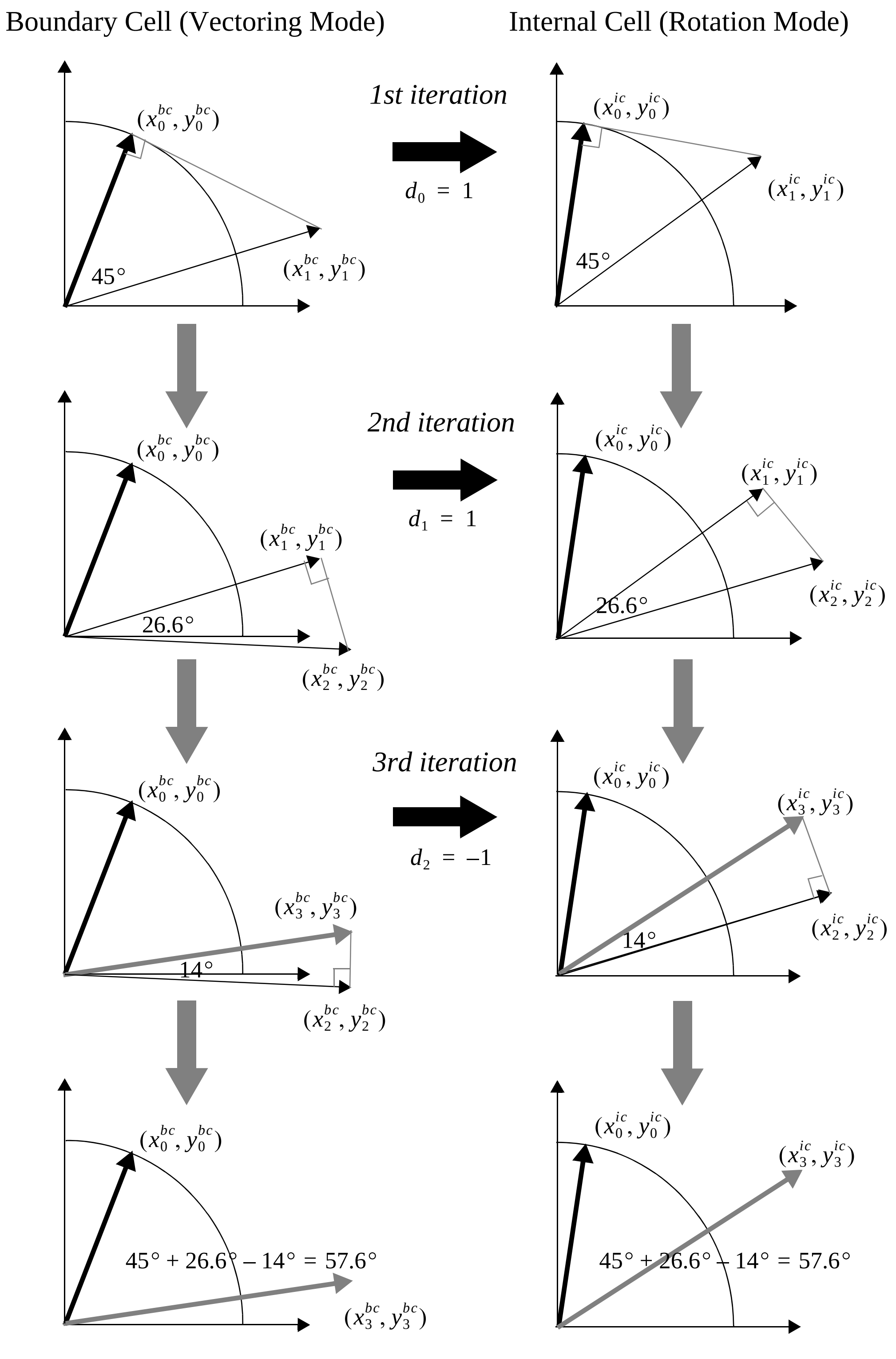


Figure: 5.1: CORDIC iteration in boundary and internal/downdating cells

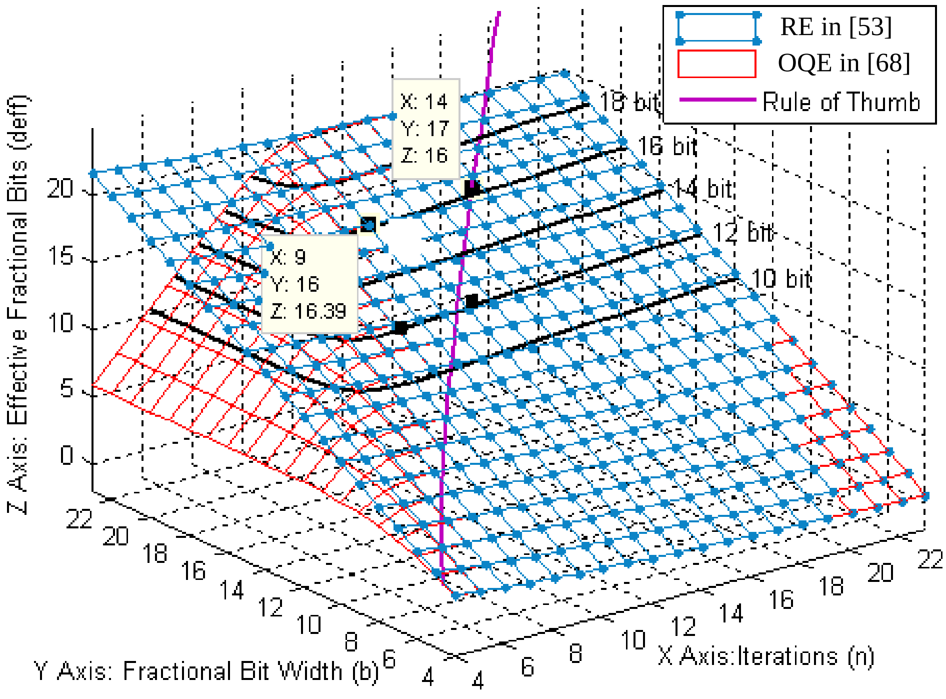


Figure: 5.3: Selection of iteration number under specific effective fractional bits

5.3 OQE Analysis of Coarse Angle Rotation CORDIC

According to Section 5.2, the BCs, ICs and DCs can all work on the newly proposed coarse angle rotation CORDIC mode. Hence the vectoring mode approximation error (AE) discussed in Section 4.5.2 can also feature the coarse angle rotation CORDIC.

Figure 5.3 compares the output precisions estimated by Walther's 'rule of thumb' [51], rounding error (RE) only and the OQE where the improved AE assessment in [68] is also considered. The 3D representation reveals the number of iterations n, the fractional bit-width b of the x and y path, and the minimum precision d_{eff} in terms of effective fractional bits. The contour lines allow an exchange between n and b. It is noticed that the RE only and OQE based meshes are overlapped when the value of n is set beyond their interface line. Hence by choosing n value close to the interface line, the lowest execution time of CORDIC can be achieved without hugely increasing b.

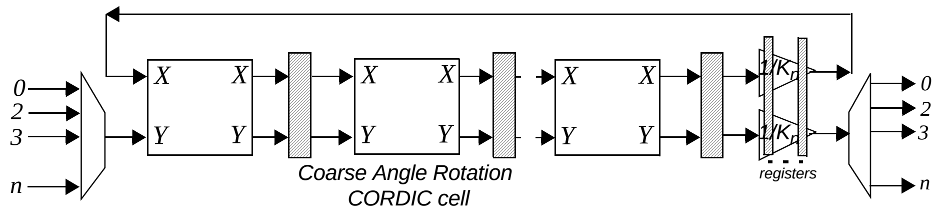


Figure: 5.4: Channel Interleaving

Consider the case that 16 effective fractional bits are required for a vector magnitude $|v(0)| = \sqrt{5}$. In reference [68], the improved OQE assessment predicts only 9 iterations with 16 fractional bits. However, the 'rule of thumb' estimates 14 iterations and 17 effective fractional bits are required. In this case, the modified OQE algorithm offers a significant saving of 21 percentage of iterations required by the 'rule of thumb'.

5.4 Pipeline-Interleaving CORDIC

According to Section 4.4.2, for pipelining the CORDIC based PEs, one solution to avoid changing the algorithm's input-output behaviour is using upsampling and downsampling technology on the input and output samples respectively. But this method can not improve the throughput. The resulted sampling rate is n new input/1 output per n clock cycles.

An alternative approach is to share the same CORDIC unit with more than one data channels, namely the pipeline-interleaving in Figure 5.4. When there are n pipeline stages inside the feedback loop, up to $n + 1$ independent input channels can share this hardware. As soon as the first sample enters the pipeline and clears the first pipeline stage, the second channel's samples can enter.

The pipeline-interleaving CORDIC in Figure 5.4 offers a low cost hardware solution, which allows the work of several CORDICs to be done by a single one. However, unless $n + 1$ input channels exist to fill n pipeline stages then there will still be some redundancy [41].

5.5 Mapping Procedure of QR-RLS Systolic Array

The QR-RLS BC and IC/DC can be mapped onto a single PE architecture by applying the pipeline-interleaving technique, as shown in Figure 5.5. According to Section 3.8, to transfer the double triangular QR-RLS array to a processor-like architecture, firstly, the extended QR-RLS structure in Figure 5.5 (*a*) needs to be mapped to a linear array architecture in Figure 5.5 (*b*).

For the double triangular (fully parallel) array in Figure 5.5 (*a*), assume each PE is unpipelined and operates on the same clock rate f_c with the input signal $x(k)$ and $d(k)$ and output signals $\xi(k)$, $\gamma(k)$ and $b(k)$. After all the N rows are mapped to a single one, as shown in Figure 5.5 (*b*), the resultant linear array still operates on the clock rate f_c, but the input and output signals ($x(k)$, $d(k)$, $\xi(k)$, $\gamma(k)$ and $b(k)$) have to be downsampled to f_c/N in order to facilitate a continuous flow of data in and out of the QR-RLS array. Now comparing with the fully paralleled design in Figure 5.5 (*a*), the latency of this architecture is N clock cycles (clock rate = f_c).

Then the next step is the space-time sharing of all the PEs of the linear array, by pipeline-interleave all the PEs on the same row to a 'process-like' single PE. In Figure 5.5 (*c*), P pipeline registers are introduced into the feedback loop. By applying the cut-set retiming technique, the long critical path existing inside the CORDIC arithmetic is reduced, the required clock rate of all the PEs in Figure 5.5 (*c*) increases to Pf_c. The latency of this resultant architecture has no change (NP clock cycles, but Pf_c clock rate). The data streams processed by the $N + 3$ columns of cells is considered as $N + 3$ separate channels, and therefore the pipeline-interleaving scheme could be employed here. This allows the row of $N + 3$ cells to be mapped onto a single, channel-interleaved CORDIC cell, as shown in Figure 5.5 (*d*).

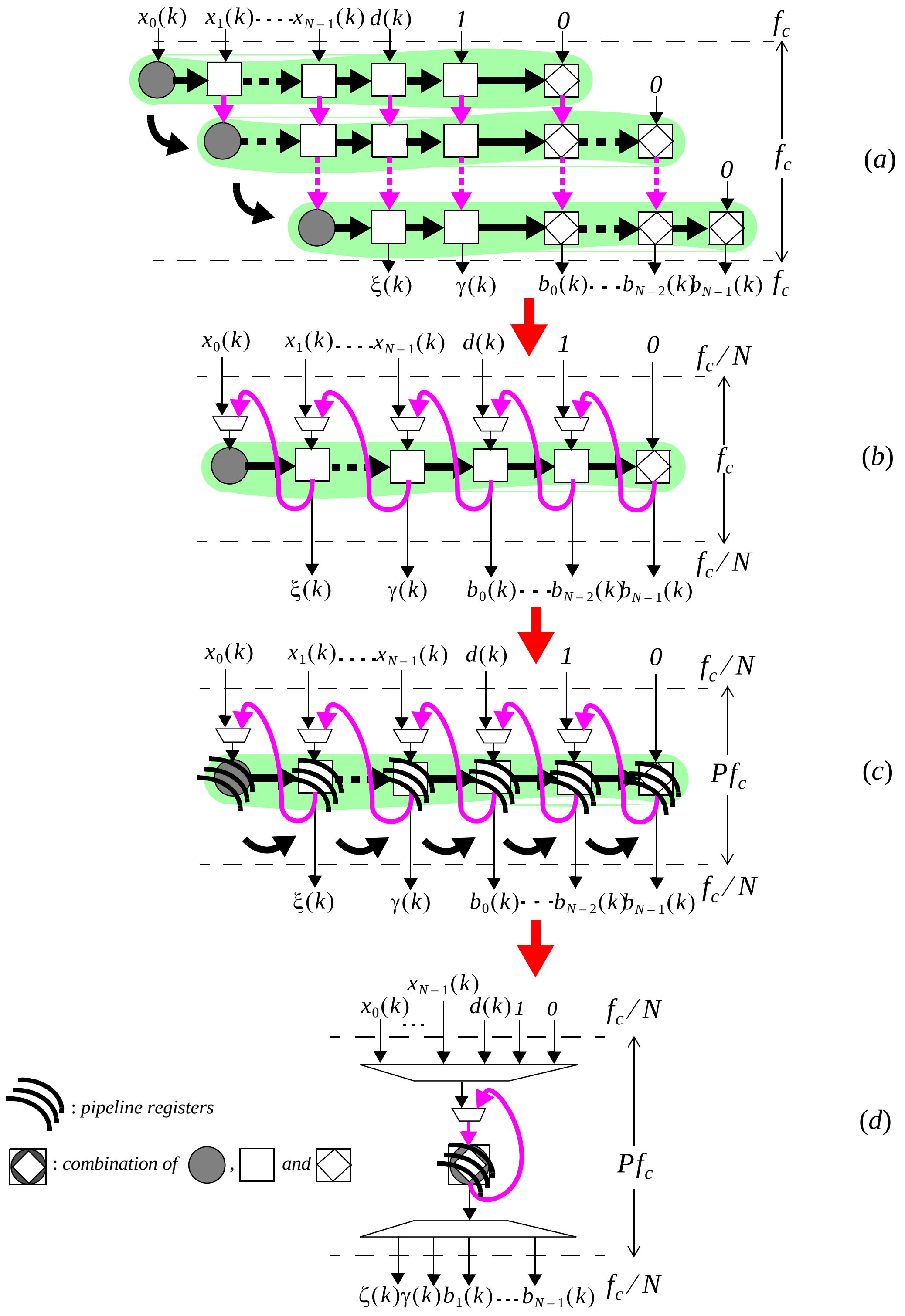


Figure: 5.5: QR-RLS array transformation

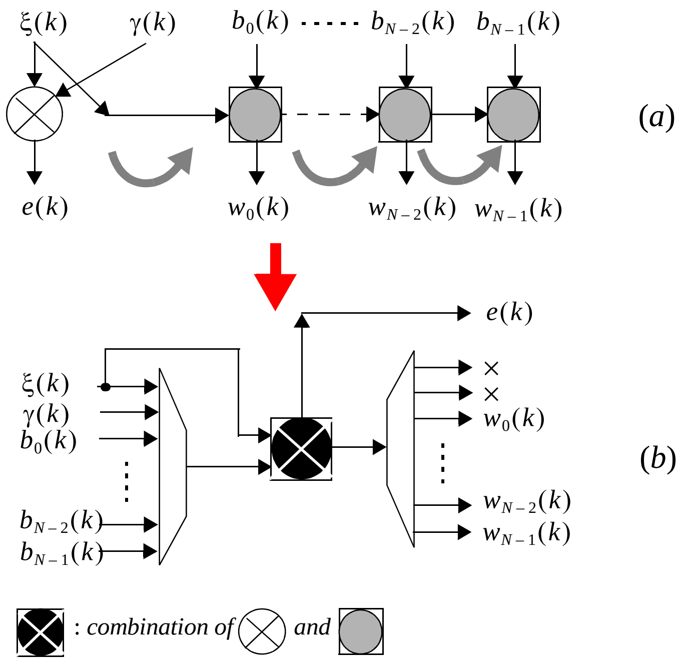


Figure: 5.6: Error and weights extraction (MAC) array

In contrast with the works in [24] and [70] which map the boundary and internal cells as the discrete processing blocks, this research focuses on generalising the architecture of the boundary and internal cells, such that they can be folded onto a single processing element. All the Extraction Cells (EC) in Figure 5.6 (*a*) can also be easily mapped to a single Multiply-ACcumulate (MAC) arithmetic component, as shown in Figure 5.6 (*b*). This approach results in a ‘single PE + MAC’ QR-RLS processor which is suitable for decomposition of a variable-size matrix.

The scalability of the resultant QR-RLS processor is shown in Figure 5.7. To process a QR decomposition of a 4×4 matrix, only the grey cells need to work, others are considered redundant. If the matrix size increases to 5×5, the white cells are also triggered to work. Whatever the size of QR decomposition changes, the hardware cost of the ‘Single PE + MAC’ QR-RLS processor remains almost the same, if the overheads of multiplexers and pipeline registers are not counted.

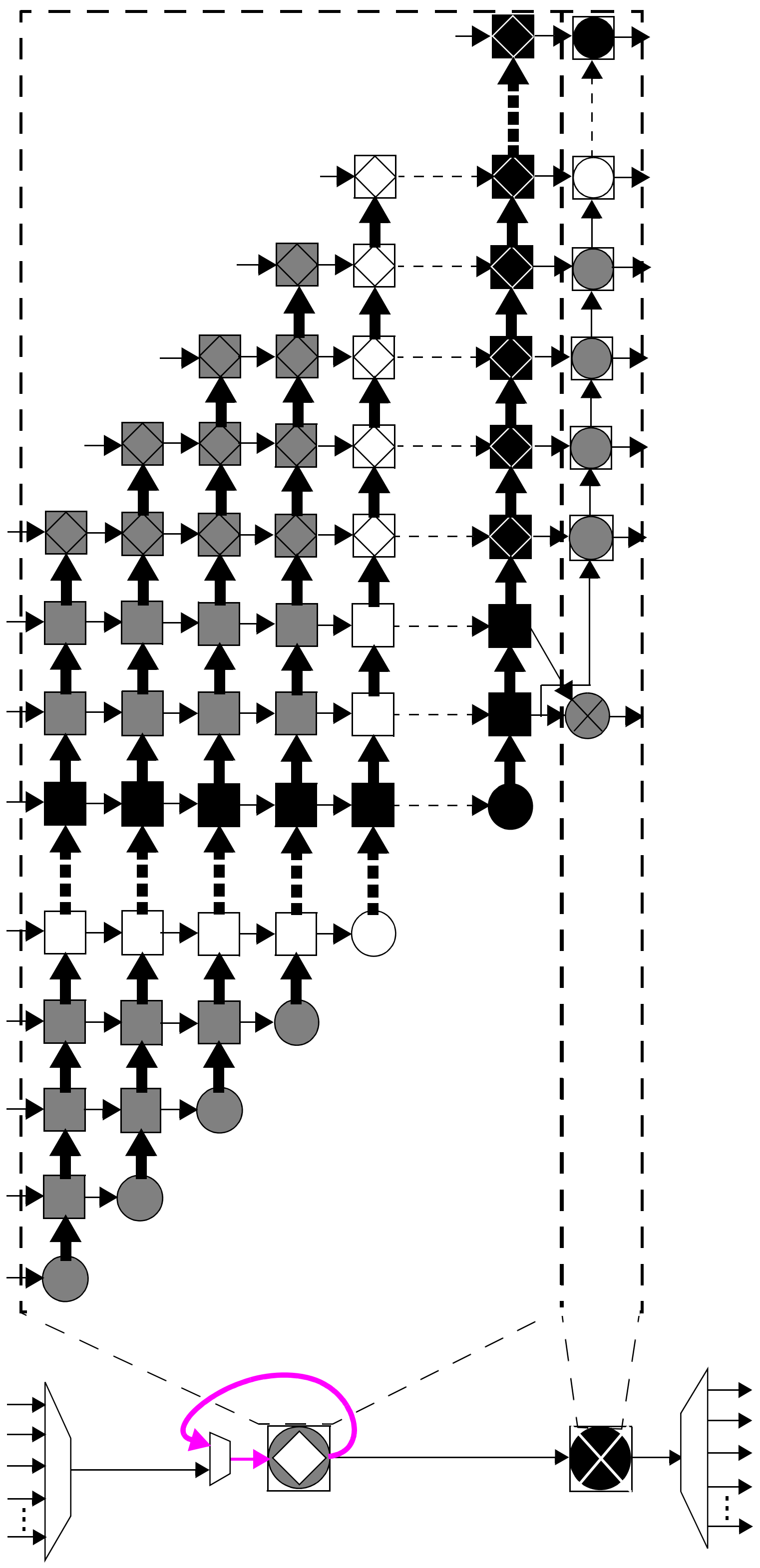


Figure: 5.7: Scalability of the ‘Single PE + MAC’ QR-RLS

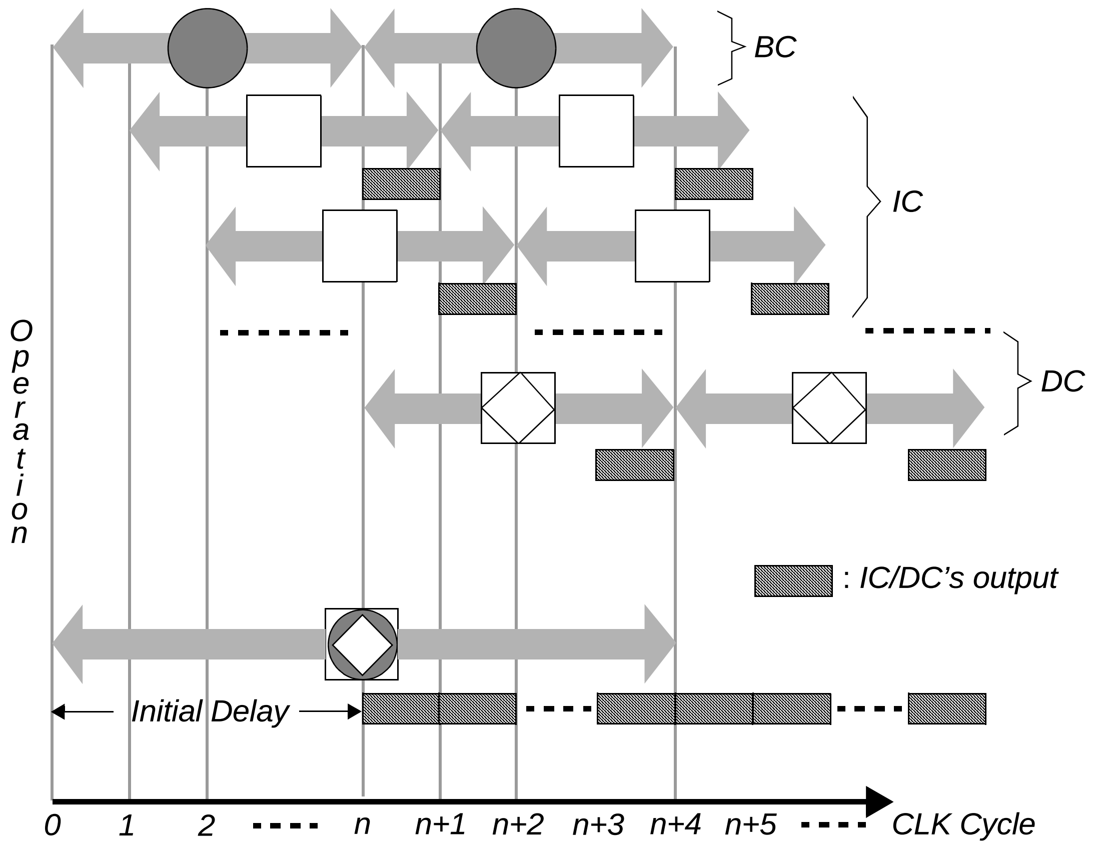
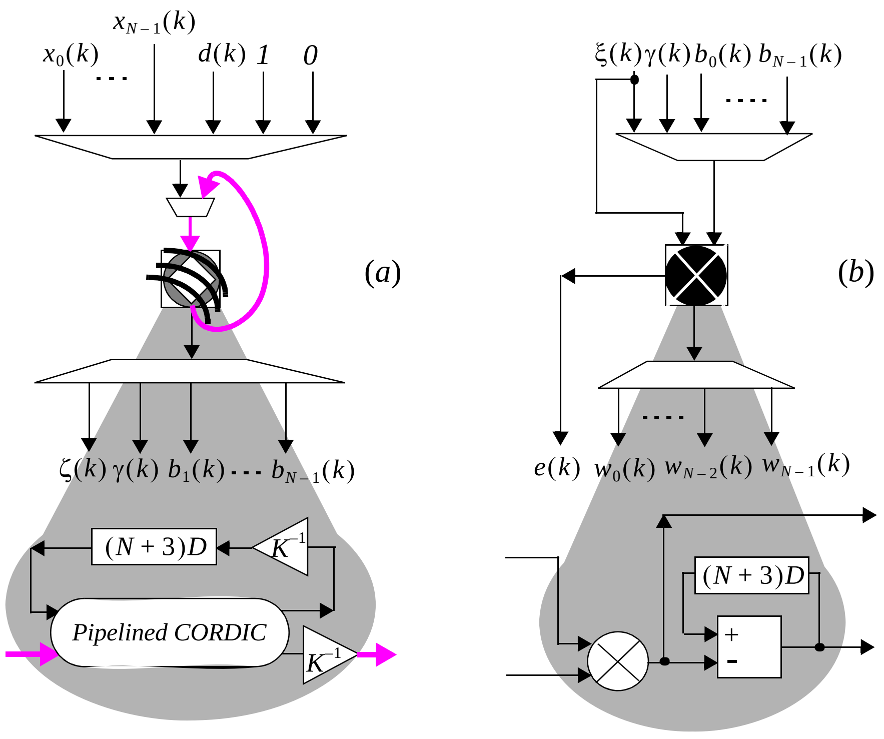


Figure: 5.8: Scheduling of the QR-RLS pipeline-interleaving

This proposed pipeline-interleaving CORDIC reduces the pipeline latency of conventional CORDIC for a target numerical accuracy [71]. As illustrated in Figure 5.8, after the initial latency of the circuit, the computation rate of the cell is one new input/output per clock cycle. Hence, the architecture is suitable for achieving the high speed performance, while avoiding using the additional hardware (Mult-Add units) for the vector rotation.

5.6 Comparision with the Classic CORDICs

This section compares the implementation results of the pipeline-interleaving coarse angle CORDIC (*Coarse Cordic Givens*) based real valued QR-RLS, as shown



$x_0(k)$
$x_{N-1}(k)$
$d(k)$
1
0
(a)
$\zeta(k)$ $\gamma(k)$ $b_1(k)$ ... $b_{N-1}(k)$
$(N+3)D$
K^{-1}
Pipelined CORDIC
K^{-1}
$\xi(k)$ $\gamma(k)$ $b_0(k)$ $b_{N-1}(k)$
(b)
$e(k)$ $w_0(k)$ $w_{N-2}(k)$ $w_{N-1}(k)$
$(N+3)D$
+
-

Figure: 5.9: (*a*) 'Single PE' part of real valued QR-RLS; (*b*) 'Single MAC' part of real valued QR-RLS

in Figure 5.9, with the corresponding BCs and ICs constructed by a standard CORDIC Givens [50] and standard CORDIC lookup [24]. 16 CORDIC iterations and 18-bit wordlengths were chosen for this comparison. The FPGA resource utilisations listed in Table 5.1 were generated using Xilinx ISE 10.1 to target a Xilinx Virtex-4 xc4vsx35-10ff668 device, and shown that the proposed coarse angle rotation mode CORDIC saves 32.5% hardware resources compared to the classic one in [50], for both the real valued BCs and ICs. This saving is largely due to the omission of angle z accumulation of the CORDIC subcircuit and hence is independent of wordlength and number of iterations (note the Xilinx FPGA's on-chip memory (i.e. BlockRAM) used by the standard CORDIC Givens is to compensate the CORDIC scaling factor K_n, nevertheless, the proposed pipeline-interleaving single PE only uses the BlockRAM to store the intermediate values in the feedback loop). Results in Table 5.1 also show that the novel coarse angle rotation mode CORDIC based BC uses nearly the same amount of resources as the IC, due to the similarity of their architectures. The relationship

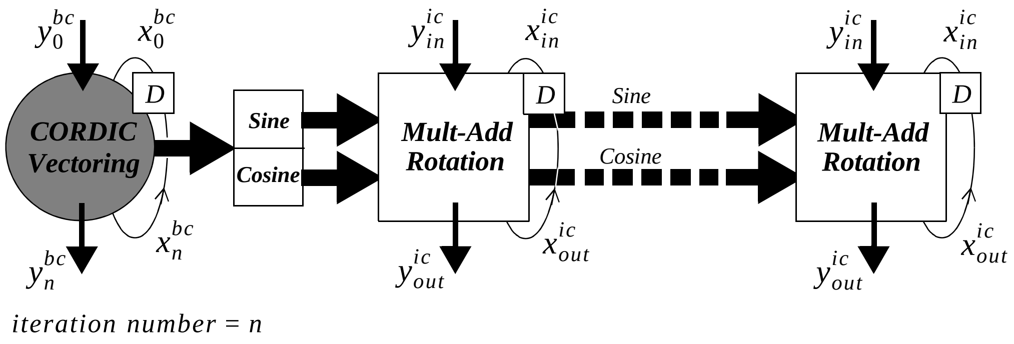


Figure: 5.10: Signal transfer between the BC and IC/DC in ref [24]

between the matrix dimension and the maximum sampling rate of the real valued QR-RLS processor is

$$Max\ Sampling\ Rate\ =\ \frac{Max\ CLK\ Rate \times (Dimension + 3)}{(CORDIC\ iterations + 2) \times Dimension} \tag{5.1}$$

An alternative architecture (*Standard Cordic Lookup*) in [24] is shown in Table 5.1. This reference design requires the distributed boundary and internal cells. The internal cell is implemented by the multiply-accumulate (MAC) functional units (DSP48 units for Xilinx). The boundary cells in the QR-RLS array in Figure 5.10 use the standard CORDIC architecture which involves the angle ‘*z*’ calculation component. One additional LookUp Table (LUT), implemented using a BlockRAM, converts each generated ‘*z*’ to its corresponding Sine and Cosine values. The ICs can perform the Givens rotations using only multiply and add operations. Here the wordlength and number of CORDIC iterations are still 18 bits and 16 respectively.

An architecture for real Given's rotations (based on that presented in [24] for complex Givens rotation) was constructed and compared with the architecture presented in this section. Its synthesised resource utilization is also shown in Table 5.1. The hardware cost of the boundary cell based of this structure is larger (56%) than that of our proposed boundary cell. This design also uses the Virtex-4's arithmetic elements (DSP48 slices).

Hence to finish the processing of linear array in Figure 5.5 (*b*), the reference design

Table 5.1: Synthesis results for 3 types of CORDIC based QR Givens

Type	PE	Slices	FFs	LUTs	DSP48	BRAM
Standard CORDIC Givens [50] (extra BS stage required)	Boundary	606	770	1150	0	1
	Internal	610	770	1147	0	1
	Total	1216	1540	2297	0	2
Standard CORDIC Lookup [24] (extra BS stage required)	Boundary	640	780	1173	0	1
	Internal	58	108	73	2	0
	Total	698	888	1246	2	1
Coarse Angle Cordic Givens	Boundary	409	611	789	0	1
	Internal	408	610	789	0	1
	Total	817	1221	1578	0	2
Pipeline-Interleaving Coarse Angle CORDIC	Boundary & Internal	578	795	995	0	1
	MAC	156	126	84	2	0

Note : BS = Back Substitution

needs $NP' = N(P + \text{cycles for IC})$ cycles, in contrast to NP cycles required by this work in Figure 5.5 (*d*). Hence based on the same clock rate, the total latency cycles required by the QR-RLS in [24] for decomposing an N by N matrix is $N^2(P + cycles\ \ for\ \ IC)$ (where $N(P + \text{cycles for IC})$ cycles for processing a single row $\{x_0(k)$ to $x_{N-1}(k)\}$ of the input matrix, and $N^2(P + \text{cycles for IC})$ cycles for all the N rows of this matrix), as shown in Figure 5.11. Hence the latency of reference design in [24] is larger than the N^2P clock cycles delay for the architecture shown in Figure 5.5 (*d*).

Comparing the resource utilisation of the standard CORDIC lookup design with that of the pipeline-interleaving CORDIC listed in Table 5.1, the cost of a single PE which supports the functionality of real valued BCs, ICs and DCs is 10% less than the utilisation of one BC only (i.e 578 slices verses 640). Another key point is that the proposed pipeline-interleaving coarse angle CORDIC includes all of the computation for weight extraction. This is not included in any of the other costs which require additional back-substitution computations. Furthermore the architectures of BC and IC in [24] are too different to be mapped together. Due to the high degree of scalability,

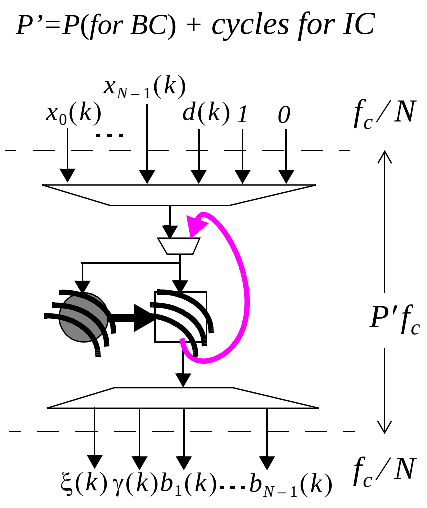

P'=P(for BC) + cycles for IC
$x_0(k)$
$x_{N-1}(k)$
$d(k)$
1
0
f_c/N
$P'f_c$
$\xi(k)\gamma(k)b_1(k)\cdots b_{N-1}(k)$
f_c/N

Figure: 5.11: QR-RLS architecture of reference design [24]

this pipeline-interleaved single PE architecture has the advantage of reduced cost compared to other existing techniques, especially for QR-RLS arrays of large size. Static Timing Analysis (STA) undertaken using Xilinx ISE shows that the circuits basing on our proposed architecture and [24] can be both clocked at 250 MHz.

5.7 Conclusion

This chapter proposed a novel structure which allows the same pipelined CORDIC unit to be efficiently time-shared by all the BC, IC and DC. Since the x and y parts are always necessary for the Givens rotation, a new CORDIC/QR architecture is presented in which the z angle accumulation of Eq. 4.1 can in fact be discarded. By comparing coarse angle rotation mode CORDIC with two conventional types of CORDICs based BCs and ICs, it was shown that the new CORDIC has the best speed-area performance, and directly yields the adaptive weight values, without the need of back-substitution. The improved OQE algorithm results in the low latency coarse angle rotatuion CORDIC.

This chapter applied the pipeline-interleaving scheme to multiplex all the PEs of QR-RLS systolic array together. By using the pipeline-interleaving CORDIC to multiplex more than one data channels together, the pipelined architecture can be filled up efficiently. This chapter presented a method to map the extended QR-RLS array onto a ‘Single PE + MAC’ architecture. The pipeline-interleaving coarse angle CORDIC offers a low cost time-multiplexed solution, which allows the processing work of the BC and IC to be undertaken using the same hardware. The pipeline-interleaving CORDIC based ‘Single PE + MAC’ QR-RLS processor combines the low hardware consumption with the benefit of high speed and scalability.

Chapter 6

Complex Valued Processor-Like QR-RLS Architecture

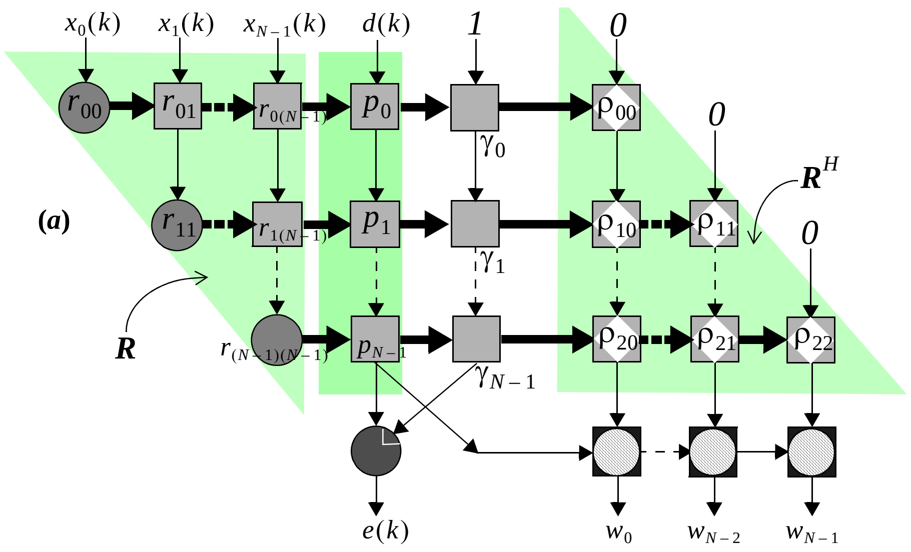
6.1 Introduction

In Chapter 5, the CORDIC arithmetic was implemented as part of the computational requirements of the real Givens rotation based QR-RLS algorithm. However, in modern digital communication systems, complex valued QR algorithm is widely used and implemented for applications such as MIMO [71], fast equalization [72] and beamforming [73]. Hence the Complex Givens Rotation (CGR) based PE is required to accomplish the complex valued QR-RLS array.

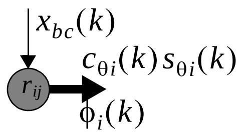
This chapter reviews the complex valued QR-RLS downdating array, and thereafter designs a CORDIC based complex valued QR-RLS processor, which features array multi-data stream sharing, pipelining, and can be used for solving problems for QR matrices of different dimensions. This chapter will also demonstrate how this architecture can be efficiently implemented on an FPGA (or considered for ASIC implementation), and run at suitable high sampling rates for high value DSP and communications applications.

6.2 Complex Valued QR-RLS Systolic Array For Weight Extraction

According to Table 3.3, the complex valued extended QR-RLS recursive algorithm is summarized as:

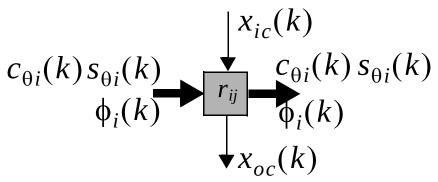


(*b*) *Boundary Cell* (*BC*)



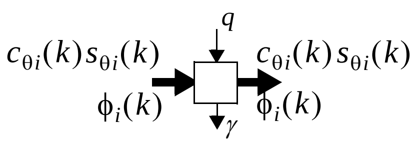
$$\text{if } x_{bc}(k) = 0, \text{ then}$$
$$c_i(k) = 1; s_i(k) = 0$$
$$\text{else}$$
$$\phi_i(k) = \operatorname{atan}\frac{\operatorname{imag}(x_{bc}(k))}{\operatorname{real}(x_{bc}(k))}$$
$$c_{\theta i}(k) = |r_{ij}(k)|(|r_{ij}(k)|^2 + |x_{bc}(k)|^2)^{-1/2}$$
$$s_{\theta i}(k) = |x_{bc}(k)|(|r_{ij}(k)|^2 + |x_{bc}(k)|^2)^{-1/2}$$
$$|r_{ij}(k+1)| = \lambda^{1/2}(|r_{ij}(k)|^2 + |x_{bc}(k)|^2)^{-1/2}$$

(*c*) *Internal Cell* (*IC*)


$x_{ic}(k)$
$c_{\theta i}(k)\,s_{\theta i}(k)$
$\phi_i(k)$
r_{ij}
$c_{\theta i}(k)\,s_{\theta i}(k)$
$\phi_i(k)$
$x_{oc}(k)$

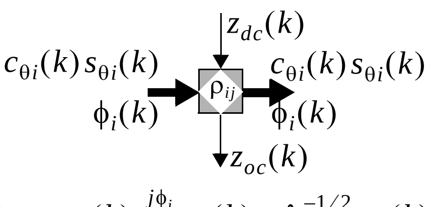
$$r_{ij}(k) = s_{\theta i}(k)e^{j\phi_i}x_{ic}(k) + \lambda^{1/2}c_{\theta i}(k)r_{ij}(k)$$
$$x_{oc}(k) = c_{\theta i}(k)e^{j\phi_i}x_{ic}(k) - \lambda^{1/2}s_{\theta i}(k)r_{ij}(k)$$

(*d*)

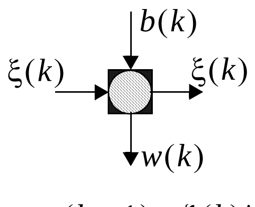

q
$c_{\theta i}(k)s_{\theta i}(k)$
$\phi_i(k)$
$c_{\theta i}(k)s_{\theta i}(k)$
$\phi_i(k)$
γ

$$\gamma = qc_i(k)$$

(*e*) *Downdating Cell* (*DC*)


$z_{dc}(k)$
$c_{\theta i}(k)\,s_{\theta i}(k)$
$\phi_i(k)$
ρ_{ij}
$c_{\theta i}(k)\,s_{\theta i}(k)$
$\phi_i(k)$
$z_{oc}(k)$

(*f*) *Extraction Cell* (*EC*)



$$\rho_{ij}(k) = s_{\theta i}(k)e^{j\phi_i}z_{dc}(k) + \lambda^{-1/2}c_{\theta i}(k)\rho_{ij}(k)$$
$$z_{oc}(k) = c_{\theta i}(k)e^{j\phi_i}z_{dc}(k) - \lambda^{-1/2}s_{\theta i}(k)\rho_{ij}(k)$$

$$w(k) = w(k-1) - \xi(k)^* \cdot b(k)$$

Figure: 6.1: Systolic array of complex valued extended QR-RLS

$$\boldsymbol{Q}(k)\begin{bmatrix}\lambda^{1/2}\boldsymbol{R}(k-1) & \lambda^{1/2}\boldsymbol{p}(k-1) & \lambda^{-1/2}\boldsymbol{R}^{-H}(k-1)\\ \boldsymbol{x}^{T}(k) & d(k) & \boldsymbol{0}\end{bmatrix} = \begin{bmatrix}\boldsymbol{R}(k) & \boldsymbol{p}(k) & \boldsymbol{R}^{-H}(k)\\ \boldsymbol{0} & \xi(k) & \boldsymbol{b}(k)\end{bmatrix} \tag{6.1}$$

Where $\boldsymbol{R}^H$ represents the Hermitian transpose of matrix $\boldsymbol{R}$.

Figure 6.1 (*a*) illustrates the SFG of the QR-RLS array with downdating for a simple $N = 3$ weight example. This array uses the Givens transform by first calculating the rotation angles, via Givens Generation (GG) in the Boundary Cells (BCs) and Givens Rotation (GR) in the Internal Cells (ICs) and Downdating Cells (DCs). This iteratively converts the input data matrix formed by successive rows of the input data vector $\boldsymbol{x}^H(k)$, into an upper triangular matrix $\boldsymbol{R}$ as shown recursively in Eq. 6.1. Each r_{ij} value that is stored and updated inside of the array is one element of this upper triangular matrix, where the subscript *ij* represents the location of the element in the $\boldsymbol{R}$ matrix. The definitions of the BCs, ICs, DCs and Extraction Cells (ECs) of QR-RLS array are shown in Figure 6.1 (*b*), (*c*), (*e*) and (*f*) respectively. The s and c (or sine and cosine) elements are used to express the values calculated within a boundary cell which is performing the Givens generation. The column of processing cells (Figure 6.1 (*d*)) on the right hand side of the u_i elements simply generates the product of the cosines $c_1(k)c_2(k)\ldots c_N(k)$ and and hence the least squares error $e(k)$ residual can be found from the output of the final complex multiplier.

6.3 Complex Valued QR-RLS Processing Elements

In this section, the implementation of PEs of the complex valued QR-RLS will be discussed following the work in [74]. The Givens generation/rotation technique is widely applied for the QRD to reduce the input matrix to a triangular form by applying successive rotations to matrix elements below the main diagonal of the input matrix. The idea of complex Givens generation/rotation can be easily illustrated on the following length 2 column vector:

$$\begin{bmatrix} r_0 & r_1 \\ x_0 & x_1 \end{bmatrix} = \begin{bmatrix} |r_0|e^{j\theta_{r0}} & |r_1|e^{j\theta_{r1}} \\ |x_0|e^{j\theta_{x0}} & |x_1|e^{j\theta_{x1}} \end{bmatrix} \tag{6.2}$$

where $j = \sqrt{-1}$; $|r|$, $|x|$ represent the magnitudes and θ_r, θ_x are the corresponding phases of r and x. The goal of Givens generation is to null the vector x_0.

The Complex Givens Rotation (CGR) is described by two rotation angles θ_1, θ_2 through the following matrix transformation [76]:

$$\begin{bmatrix} \cos\theta_1 & e^{j\theta_2}\sin\theta_1 \\ -e^{-j\theta_2}\sin\theta_1 & \cos\theta_1 \end{bmatrix} \begin{bmatrix} |r_0|e^{j\theta_{r0}} & |r_1|e^{j\theta_{r1}} \\ |x_0|e^{j\theta_{x0}} & |x_1|e^{j\theta_{x1}} \end{bmatrix} = \begin{bmatrix} |\bar{r}_0|e^{j\theta_{\bar{r}0}} & |\bar{r}_1|e^{j\theta_{\bar{r}1}} \\ 0 & |\bar{x}_1|e^{j\theta_{\bar{x}1}} \end{bmatrix} = \begin{bmatrix} \bar{r}_0 & \bar{r}_1 \\ 0 & \bar{x}_1 \end{bmatrix} \tag{6.3}$$

where angles θ_1, θ_2 are chosen to zero the matrix element below the main diagonal, and determined by:

$$\begin{aligned} \theta_1 &= \operatorname{atan}(|r_0| / |x_0|) \\ \theta_{r0} &= \theta_{\bar{r}0} = \operatorname{atan}(\operatorname{imag}(r_0) / \operatorname{real}(r_0)) \\ \theta_{x0} &= \operatorname{atan}(\operatorname{imag}(x_0) / \operatorname{real}(x_0)) \\ \theta_2 &= \theta_{x0} - \theta_{r0} \end{aligned} \tag{6.4}$$

Figure 6.2 and 6.3 graphically show the data flows of CGR based BCs and ICs respectively. As demonstrated in Chapter 4, the Boundary (BC), Internal (IC) and Downdating Cells (DC) which form the QR-RLS array are implementable via either the vectoring or rotation modes of the classic CORDIC algorithm. As shown in Figure 6.2, each BC ultilises 3 vectoring mode CORDICs, 1 rotation mode CORDIC, together with 1 adder. The implementation of each IC requires 6 CORDICs based on rotation mode shown in Figure 6.3. Obviously, both architectures result in large costs and systolic operation times. Since the architectures of Givens generation and Givens rotation are very similar, letter 'G' is used to represent both of them when appropriate.

An alternative approach to simplify the operation stated in Eq. 6.3 can be realized through a unitary matrix transformation, and is described by the Three Angle Complex

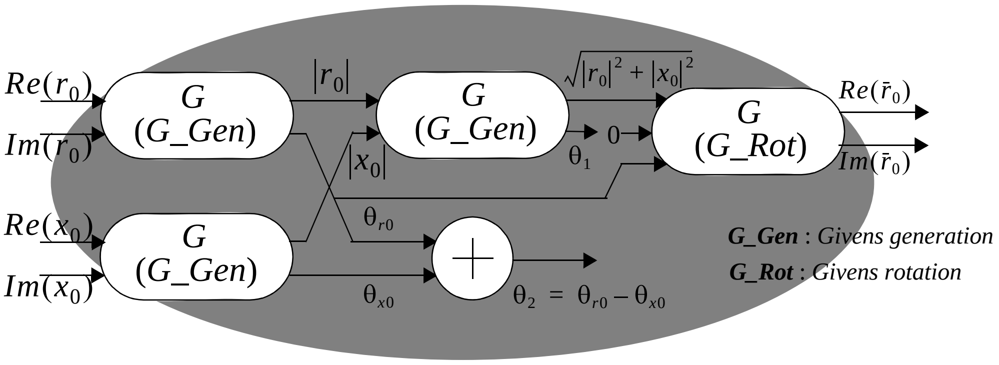


Figure: 6.2: Signal Flow of CGR Based BC

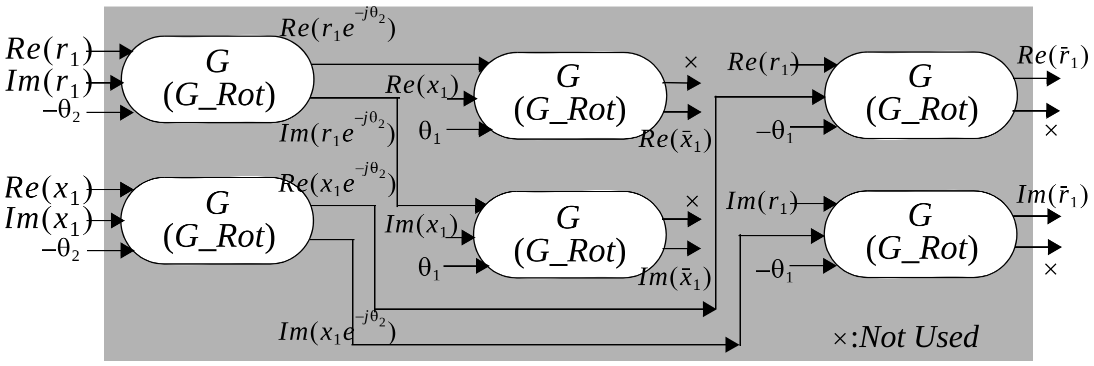


Figure: 6.3: Signal Flow of CGR Based IC

Rotation (TACR) equation in Eq. 6.5:

$$\begin{bmatrix} e^{-j\theta_{r0}} & 0 \\ 0 & e^{-j\theta_{x0}} \end{bmatrix} \begin{bmatrix} \cos\theta_1 & e^{j\theta_2}\sin\theta_1 \\ -e^{-j\theta_2}\sin\theta_1 & \cos\theta_1 \end{bmatrix} \begin{bmatrix} |r_0|e^{j\theta_{r0}} & |r_1|e^{j\theta_{r1}} \\ |x_0|e^{j\theta_{x0}} & |x_1|e^{j\theta_{x1}} \end{bmatrix} = \begin{bmatrix} e^{-j\theta_{r0}} & 0 \\ 0 & e^{-j\theta_{x0}} \end{bmatrix} \begin{bmatrix} |\bar{r}_0|e^{j\theta_{\bar{r}0}} & |\bar{r}_1|e^{j\theta_{\bar{r}1}} \\ 0 & |\bar{x}_1|e^{j\theta_{\bar{x}1}} \end{bmatrix}$$

$$= \begin{bmatrix} |\bar{r}_0| & |\bar{r}_1|e^{j(\theta_{\bar{r}1} - \theta_{r0})} \\ 0 & |\bar{x}_1|e^{j(\theta_{\bar{x}1} - \theta_{x0})} \end{bmatrix} = \begin{bmatrix} |\bar{r}_0| & \tilde{r}_1 \\ 0 & \tilde{x}_1 \end{bmatrix} \tag{6.5}$$

Note that the TACR matrix transformation introduces the real element $|\bar{r}_0|$ on the matrix diagonal. Application of the TACR approach for an $N \times N$ square matrix will lead to the appearance of real elements on the matrix diagonal. For Eq. 6.5, one further transformation may be adopted by:

$$\begin{bmatrix} e^{-j\theta_{r0}} & 0 \\ 0 & e^{-j\theta_{x0}} \end{bmatrix} \begin{bmatrix} \cos\theta_1 & e^{j\theta_2}\sin\theta_1 \\ -e^{-j\theta_2}\sin\theta_1 & \cos\theta_1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1 e^{j\theta_3} & \sin\theta_1 e^{j\theta_4} \\ -\sin\theta_1 e^{j\theta_3} & \cos\theta_1 e^{j\theta_4} \end{bmatrix} \tag{6.6}$$

To obtain an upper trianglar matrix the presented unitary transformation requires three angles θ_1, θ_3 and θ_4 which are defined by:

$$\begin{aligned} \theta_1 &= \operatorname{atan}(|r_0|/|x_0|) \\ \theta_3 &= -\theta_{r0} \\ \theta_4 &= -\theta_{x0} \end{aligned}$$

Hence the complex samples r_1 and x_1 are transformed to complex samples $\tilde{r}_1$ and $\tilde{x}_1$ according to:

$$\begin{bmatrix} |\bar{r}_0| & \tilde{r}_1 \\ 0 & \tilde{x}_1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1 e^{j\theta_3} & \sin\theta_1 e^{j\theta_4} \\ -\sin\theta_1 e^{j\theta_3} & \cos\theta_1 e^{j\theta_4} \end{bmatrix} \begin{bmatrix} r_0 & r_1 \\ x_0 & x_1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & \sin\theta_1 \\ -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \begin{bmatrix} e^{-j\theta_{r0}} & 0 \\ 0 & e^{-j\theta_{x0}} \end{bmatrix} \begin{bmatrix} r_0 & r_1 \\ x_0 & x_1 \end{bmatrix} \tag{6.7}$$

Eq. 6.7 consists of a phase compensation term and a real valued Givens rotation. By mapping this algorithm to the CORDIC implementation, a new CORDIC cell is constructed for the complex Givens rotations. Figure 6.4 shows how the modified BC determines the phases θ_1, θ_{r0} and θ_{x0}. This BC then applies these phases to the other ICs in the same row of the QR-RLS systolic array.

The new proposed architecture of CORDIC rotation mode based IC is shown in Figure 6.5. To perform the transformation described by Eq. 6.7, four CORDIC operations are sufficient. While considering the QR-RLS systolic array described in Figure 6.1, the $|\bar{r}_0|$ in Figure 6.4 and $Re(\tilde{r}_1)$ and $Im(\tilde{r}_1)$ in Figure 6.5 all need to be fed back as the next iteration's inputs $Re(r_0)$ in Figure 6.4 and $Re(r_1)$ and $Im(r_1)$ in Figure 6.5.

Hence for both the BC and IC, the upper-left CORDIC cells could be omitted, which results in the architectures of Double Angle Complex Rotation (DACR) in Figure 6.6 and Figure 6.7. The DACR approach [74] [75] [76] is described by two rotation angles ϕ and θ, where ϕ equals to the phase information θ_{x0} of the complex value x_0, i.e.

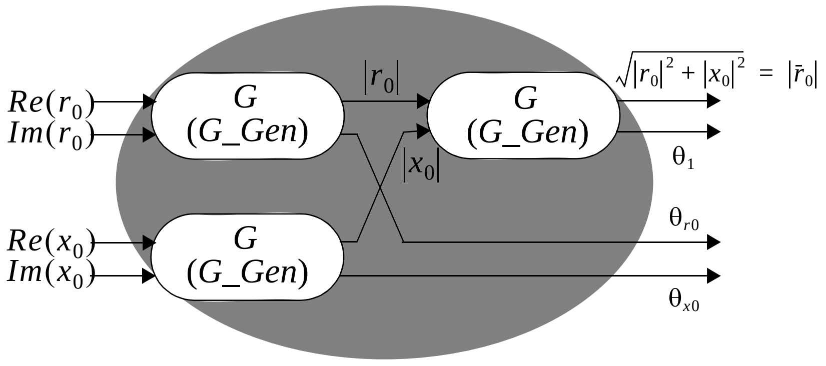


Figure: 6.4: Signal Flow of TACR Based BC

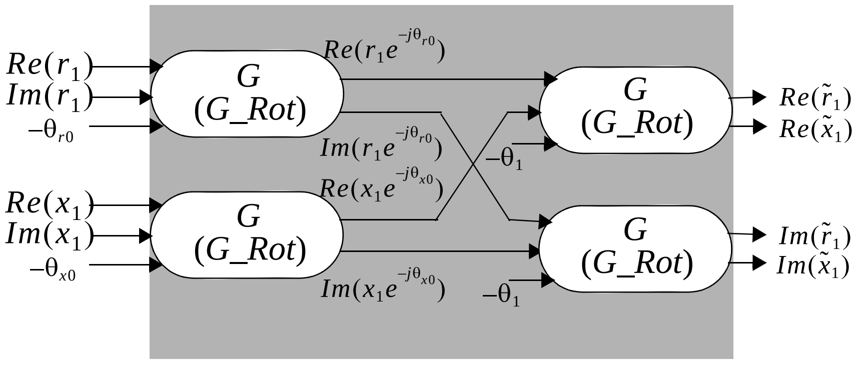


Figure: 6.5: Signal Flow of TACR Based IC

$$\phi = \theta_{x0} = \operatorname{atan}\left(\frac{\operatorname{imag}(x_0)}{\operatorname{real}(x_0)}\right)$$

and $\theta = \operatorname{atan}(|r_0|/|x_0|)$ **.** The DACR based Givens generation formula can be summarized as Eq. 6.8 [74]. Figure 6.6 and 6.7 graphically show the data flows of DACR based BC and IC.

The BC uses the double CORDICs architecture which involves the angle 'ϕ' and 'θ' calculation components. The IC/DC employs three CORDIC engines, to perform a complex Givens transform with these angles as the inputs, where the block *D* represents one data sample delay. The DACR based formula can be summarized as [75]:

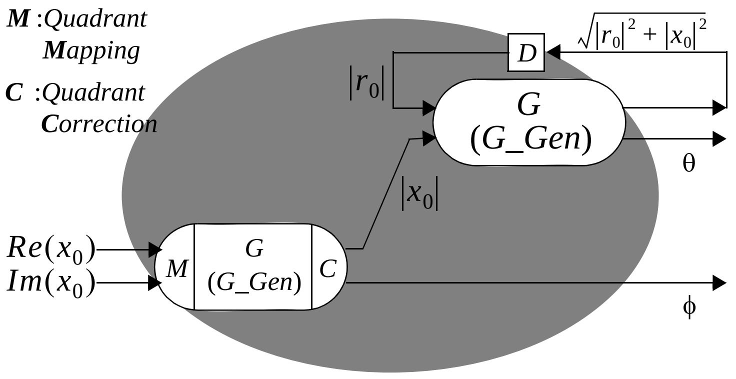


Figure: 6.6: Signal Flow of Simplified DACR Based BC

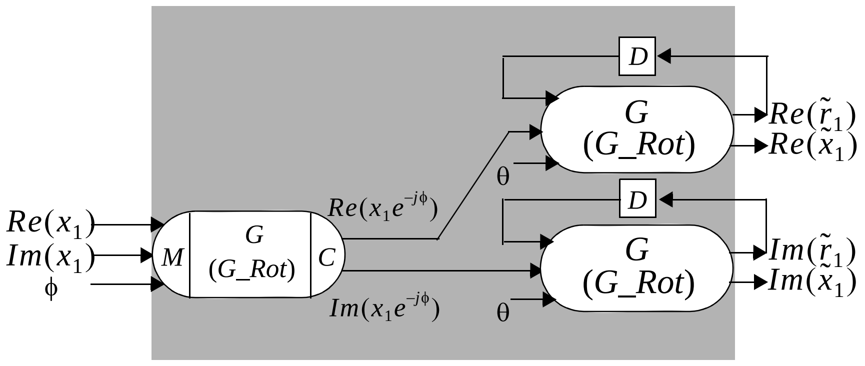


Figure: 6.7: Signal Flow of Simplified DACR Based IC

$$\begin{bmatrix} |\bar{r}_0| & \tilde{r}_1 \\ 0 & \tilde{x}_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & e^{-j\phi} \end{bmatrix} \begin{bmatrix} r_0 & r_1 \\ x_0 & x_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta e^{j\phi} \\ -\sin\theta & \cos\theta e^{j\phi} \end{bmatrix} \begin{bmatrix} r_0 & r_1 \\ x_0 & x_1 \end{bmatrix} \tag{6.8}$$

6.4 Quadrant-Correction-Free DACR

The pipeline-interleaving coarse angle rotation CORDIC in Section 5 requires very similar architecture between the BC and IC/DCs. Moreover, the implementations of both the quadrant mapping and correction also introduce an increased hardware cost. Hence it is important to investigate a low cost quadrant localisation methodology. The algorithm in Table 6.1 is consequently developed to offer a Quadrant-Correction-Free scheme [77], i.e only one simple quadrant mapping is included into the proposed

Table 6.1: Quadrant-Correction-Free Scheme

```
if $Re(x_{bc}) < 0$
    if $Im(x_{bc}) \geq 0$
    $Re(x'_{bc}) = Im(x_{bc})$ and $Re(x'_{ic}) = Im(x_{ic})$
    $Im(x'_{bc}) = -Re(x_{bc})$ and $Im(x'_{ic}) = -Re(x_{ic})$
    else
    $Re(x'_{bc}) = -Im(x_{bc})$ and $Re(x'_{ic}) = -Im(x_{ic})$
    $Im(x'_{bc}) = Re(x_{bc})$ and $Im(x'_{ic}) = Re(x_{ic})$
    end
else
    $Re(x'_{bc}) = Re(x_{bc})$ and $Re(x'_{ic}) = Re(x_{ic})$
    $Im(x'_{bc}) = Im(x_{bc})$ and $Im(x'_{ic}) = Im(x_{ic})$
end
```

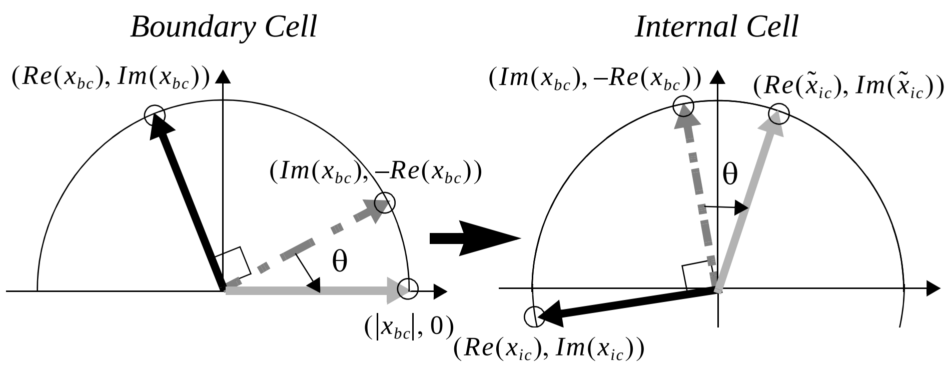

Boundary Cell
Internal Cell
$(Re(x_{bc}), Im(x_{bc}))$
$(Im(x_{bc}), -Re(x_{bc}))$
θ
$(|x_{bc}|, 0)$
$(Im(x_{bc}), -Re(x_{bc}))$
$(Re(\tilde{x}_{ic}), Im(\tilde{x}_{ic}))$
θ
$(Re(x_{ic}), Im(x_{ic}))$

Figure: 6.8: CORDIC Quadrant-Correction-Free Scheme

CORDIC based DACR.

Referring to Figure 6.8, firstly the dark input vectors of BC and IC will be rotated synchronously by $90°$ (to the dash vector) as the quandrant mapping. Then the CORDIC will take charge of the remaining rotation of angle θ in a clockwise direction. The output vectors of BC and IC are shown as the light colored vectors. Figure 6.9 shows the criteria of pipeline-interleaving coarse angle rotation CORDIC

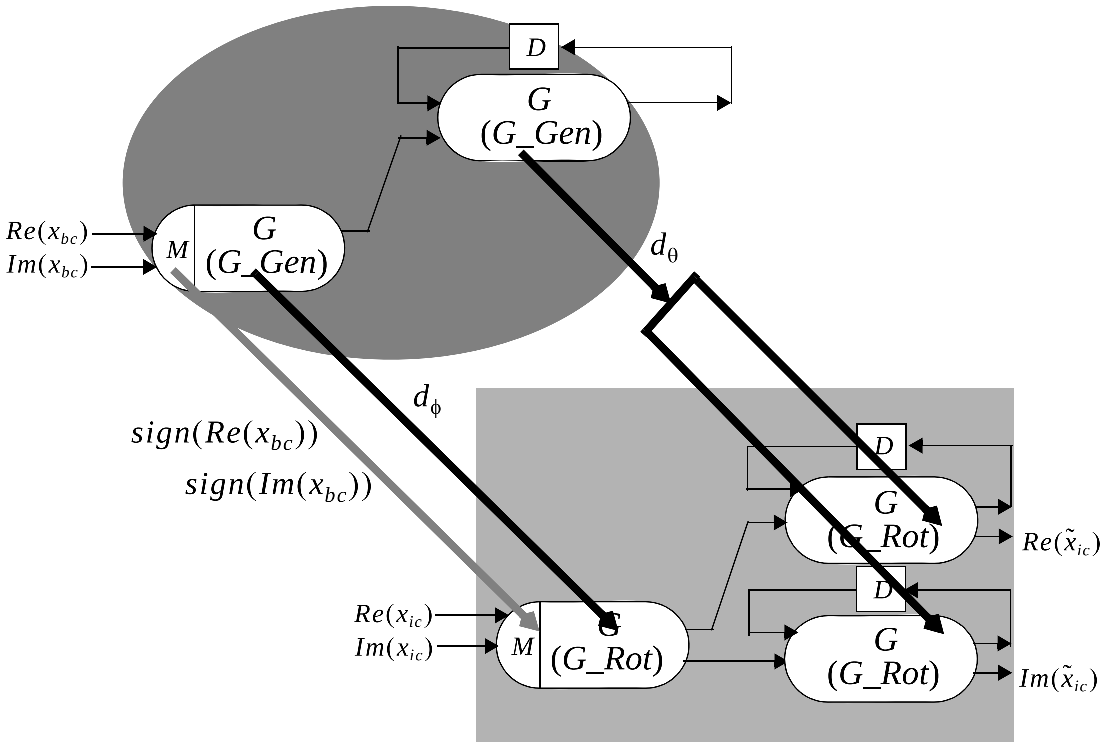
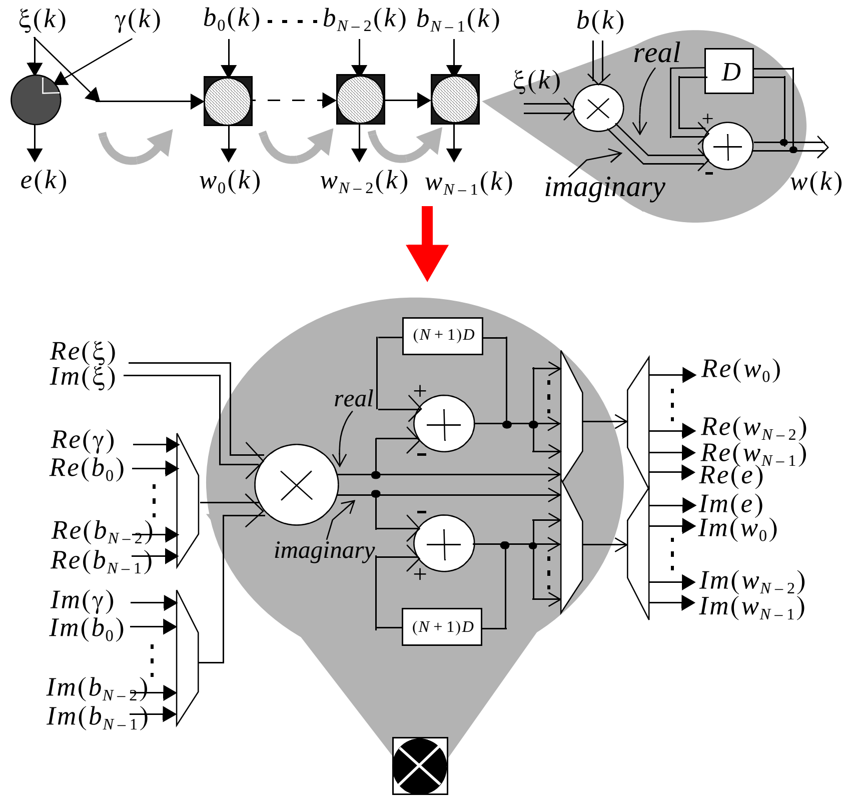


Figure: 6.9: Pipeline-interleaving coarse angle CORDIC based DACR

based DACR The grey arrow in Figure 6.9 represents the transfer of the sign of $Re(x_{bc})$ and $Im(x_{bc})$ between quadrant mapping part of BC and IC, this executes the 90° vector rotation in Figure 6.8. The dark arrows represent the transfer of decision factors d_{ϕ} and d_{θ} from BC to corresponding CORDICs in IC.

6.5 Mapping Procedure of QR-RLS Systolic Array

Similar to the pipelined interleaving mapping scheme of real valued QR-RLS array in Figure 5.5 and Figure 5.6, the Extraction Cells (ECs) on the lower right in Figure 6.1 (a) can be easily mapped in a linear array to a time-shared single complex multiply-accumulate (MAC) component. Figure 6.10 shows a single complex MAC operating on the values of a linear EC array over consecutive cycles [77]. Note that this shared unit has inputs selected from an $N + 1$ to 1 multiplexer where the first time step



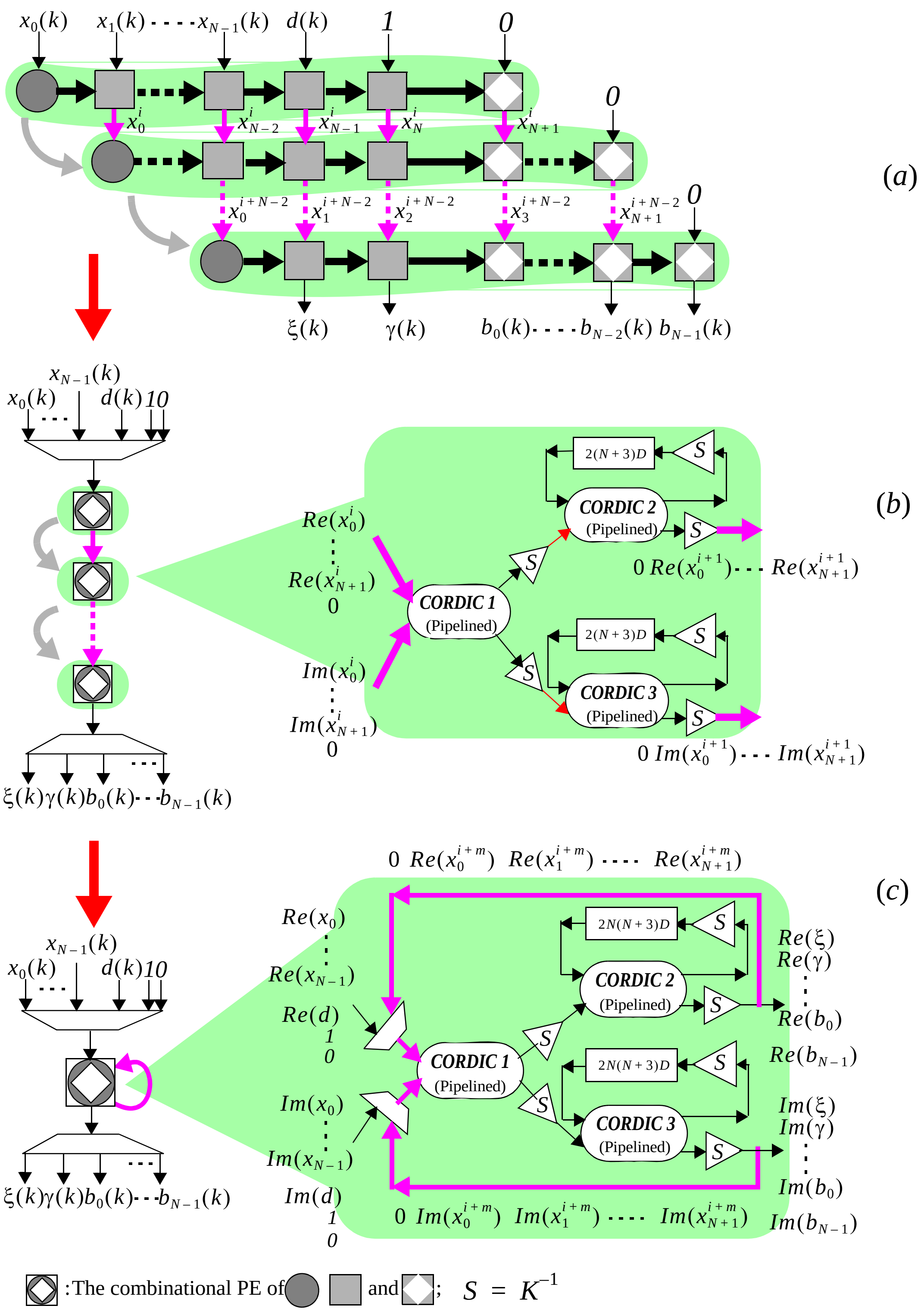
⊠: The combined PE of ● and ▨

Figure: 6.10: Error & weights extraction array

performs the single complex multiply to calculate $e(k)$ and the next N time steps (i.e. $N + 1$ time steps in total) to calculate the weight values.

6.5.1 Pipelined Interleaving

For the PEs in Figure 6.6 and Figure 6.7 which are used to implement the main array of Figure 6.1 (*a*), all 3 types of PE (BC, IC & DC) have two cascaded CORDIC stages (the first denoted CORDIC1 and the second stage as CORDIC2 and CORDIC3) to perform the complex Givens rotation. Hence the double triangular QR-RLS array with downdating could be mapped onto a single PE by applying the pipeline-interleaving technique presented in Chapter 5.2 [77]. The first step is multiplexing all the PEs on each row of the array in Figure 6.11 (*a*) to a single cell, resulting in the linear array in



$x_0(k)$ $x_1(k)$ $x_{N-1}(k)$ $d(k)$ 1 0
x_0^i x_{N-2}^i x_{N-1}^i x_N^i x_{N+1}^i
x_0^{i+N-2} x_1^{i+N-2} x_2^{i+N-2} x_3^{i+N-2} x_{N+1}^{i+N-2}
$\xi(k)$ $\gamma(k)$ $b_0(k)$ $b_{N-2}(k)$ $b_{N-1}(k)$
(a)
(b)
CORDIC 1 (Pipelined)
CORDIC 2 (Pipelined)
CORDIC 3 (Pipelined)
$2(N+3)D$
S
$Re(x_0^i)$ $Re(x_{N+1}^i)$ 0
$Im(x_0^i)$ $Im(x_{N+1}^i)$ 0
0 $Re(x_0^{i+1})$ $Re(x_{N+1}^{i+1})$
0 $Im(x_0^{i+1})$ $Im(x_{N+1}^{i+1})$
(c)
0 $Re(x_0^{i+m})$ $Re(x_1^{i+m})$ $Re(x_{N+1}^{i+m})$
0 $Im(x_0^{i+m})$ $Im(x_1^{i+m})$ $Im(x_{N+1}^{i+m})$
$2N(N+3)D$
$Re(x_0)$ $Re(x_{N-1})$ $Re(d)$ 1 0
$Im(x_0)$ $Im(x_{N-1})$ $Im(d)$ 1 0
$Re(\xi)$ $Re(\gamma)$ $Re(b_0)$ $Re(b_{N-1})$
$Im(\xi)$ $Im(\gamma)$ $Im(b_0)$ $Im(b_{N-1})$
$x_0(k)$ $x_{N-1}(k)$ $d(k)$ 1 0
$\xi(k)\gamma(k)b_0(k)$ $b_{N-1}(k)$
:The combinational PE of and ; $S = K^{-1}$

Figure: 6.11: Triangularization array transformation (*a*) rhomboid array (*b*) linear array (*c*) single PE

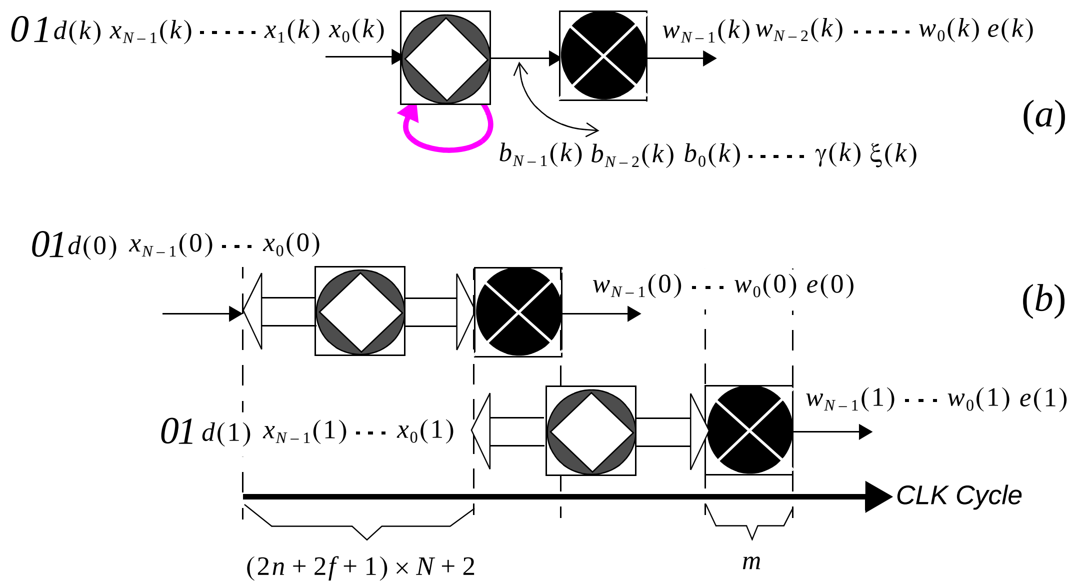
Figure 6.11 (*b*) to carry the work of triangularisation. Since each BC, IC (or DC) in Figure 6.6 and Figure 6.7 requires one cycle delay unit in the feedback loop, to store the intermediate value of each element of the updated matrix $\boldsymbol{R}$ (or $\boldsymbol{R}^{-H}$), a delay chain is employed here to store all the delayed intermediate values. According to Section 2.5.2, this delay chain can be targetted onto the on-slice shift register SRL16 on Xilinx FPGAs [13].

Then the next mapping procedure combines all the distributed PEs on the linear array in Figure 6.11 (*b*) to a final single processor-like PE (also includes an 'intermediate value' stored memory) in Figure 6.11 (*c*). Inside this processor-like PE, the effect outputs are only generated cycle by cycle from the end of 2nd CORDIC stage (note here the 'outputs' refers to the $\{0, Re(x_0^{i+k})\ldots Re(x_{N+2}^{i+k})\}$ output by each distributed rolled PE on the linear array in Figure 6.11 (*b*), where $k \in [0, N-2]$).

6.5.2 'Single PE + MAC' Architecture

Above mapping approach results in a double PE (triangularization and MAC) based QR-RLS processor (as shown in Figure 6.12 (*a*)) which is suitable for a variable size matrix decomposition (as is the case with the scalability of real valued QR-RLS presented in Chapter 5.5, up to a maximum of $(n-2)\times(n-2)$ matrices where the number of CORDIC iterations = n), costing fewer PEs than the linear array presented in [70], which requires one BC and several ICs. Note that the number of iterations of the CORDIC are now linked with the $\boldsymbol{R}$ matrix and the weight vector dimension, i.e. for a $n = 16$ iterations, the complex valued single PE array will process up to a 14×14 matrix. This architecture has the same computational complexity $O(n^2)$ as the fully parallel one in Figure 6.1.

In order to facilitate a continuous flow of data into and out of the QRD-RLS core in Figure 6.12 (*b*) (namely, assuming the size of QRD is $N \times N$, after seeding the first input group $\{x_0(0), x_1(0)\ldots x_{N-1}(0)\}$, the QR-RLS can receive the new second input group $\{x_0(1), x_1(1)\ldots x_{N-1}(1)\}$ immediately), the clock rate used within the PEs



$0\ 1\ d(k)\ x_{N-1}(k) \cdots x_1(k)\ x_0(k)$
$w_{N-1}(k)\ w_{N-2}(k) \cdots w_0(k)\ e(k)$
$b_{N-1}(k)\ b_{N-2}(k)\ b_0(k) \cdots \gamma(k)\ \xi(k)$
(a)
$0\ 1\ d(0)\ x_{N-1}(0) \cdots x_0(0)$
$w_{N-1}(0) \cdots w_0(0)\ e(0)$
(b)
$0\ 1\ d(1)\ x_{N-1}(1) \cdots x_0(1)$
$w_{N-1}(1) \cdots w_0(1)\ e(1)$
CLK Cycle
$(2n+2f+1) \times N + 2$
m

Figure: 6.12: Double PE Architecture for QR-RLS

must be higher than the input sampling rate. It is possible to reduce or remove the requirement for a higher clock rate by introducing multiple PEs. Considering the low cost goal of this research, this architecture is implemented in the ‘upsampled’ structure, hence it does not require the additional hardware resourses. As shown in Figure 6.12 (*b*), when the MAC PE is triggered to process (assuming this process takes m clock cycles), new group of $\{x_0(k), x_1(k) \ldots x_{N-1}(k), d(k), 1, 0\}$ will be able loaded to the input port of the triangularization core. The QR-RLS core in Figure 6.12 is clocked at $(n+f) \times 2N$ times the input clock rate (where N is the number of weights, n stands for the total CORDIC iterations and f is the number of clock cycle delays resulted by pipelined CORDIC scaling compensation), namely $(n+f) \times 2N$ times upsampling.

As a result, for an $N \times N$ matrix

$$\begin{bmatrix} x_0(k) & x_1(k) & \ldots & x_{N-1}(k) \\ x_0(k+1) & x_1(k+1) & \ldots & x_{N-1}(k+1) \\ \vdots & \vdots & & \vdots \\ x_0(k+N-1) & x_1(k+N-1) & \ldots & x_{N-1}(k+N-1) \end{bmatrix}$$

the number of clock cycles for processing the first row of $N+3$ inputs

$\{x_0(k), x_1(k)\dots x_{N-1}(k), d(k), 1, 0\}$ is

$$(n+f)\times 2N + (N+2) = (2n+2f+1)\times N + 2$$

Consequently, the QR decomposition of the whole matrix requires $(2n+2f+1)\times N^2 + 2N$ clock cycles.

The pipeline-interleaving based coarse angle CORDIC offers a low cost time-multiplexed solution, which allows the processing work of the BC, IC and DC to be undertaken using the same hardware. This method leads to significantly shorter execution time for the single cycle operation of QR decomposition. The reference design in [24] consists of a single boundary, internal and back-substitution cells. As with the structure in Figure 5.11, the BC in [24] uses the double CORDIC architecture which involves the angle 'ϕ' and 'θ' calculation components. Additional LUTs are applied to convert each generated 'ϕ' and 'θ' to its corresponding Sine and Cosine values. With these trigonometric functions as the inputs, the IC/DC can perform a Givens rotation using only multiply-add functional units (DSP48 units for Xilinx FPGAs). To process a group of $N+3$ inputs $\{x_0(k), x_1(k)\dots x_{N-1}(k), d(k), 1, 0\}$, this work offers $2\cdot(n+f)\cdot N + N + 2 = (2n+2f+1)\times N + 2$ cycles, while reference [24] needs $(2n+2f+1)\times N + 2$ + 'cycles to process ICs'.

The 'Single PE+MAC' QR-RLS processor in Figure 6.12 is clocked at $(n+f)\times 2N$ times the input clock rate, hence in toal $(n+f)\times 2N$ clock cycles required to complete updating one row of matrix $\boldsymbol{R}$, which results $N+3$ effective outputs (as shown in Figure 6.11 (*a*)), hence the maximum sampling rates of this section's modified CORDIC based QR-RLS is computed in Eq. 6.9, where CLK represents the maximum clock rate.

$$Max\ \ Sampling\ \ Rate = \frac{CLK\times N}{(2n+2f+1)\times N + 2} \tag{6.9}$$

As a result, the total execution time including the QR decomposition of an $N\times N$ matrix and the filter coefficients calculation is derived in Eq. 6.10, where m represents the number of pipelining levels inside the MAC element in Figure 6.12.

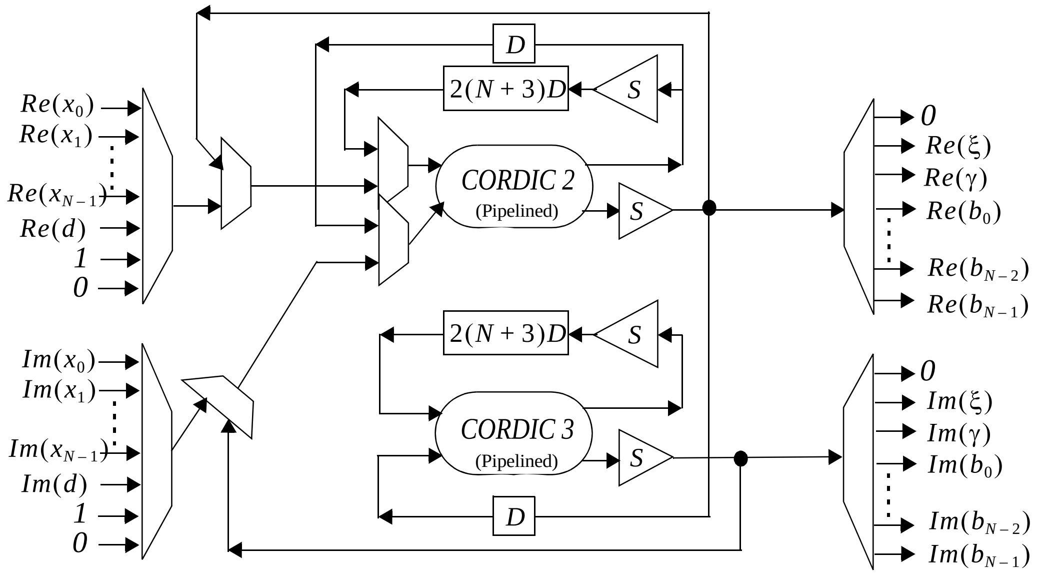


Figure: 6.13: Double CORDICs based QRD-RLS triangularization

$$Execution\ \ Time\ =\ \frac{(2n+2f+1)\times N^2+2N+m}{CLK} \tag{6.10}$$

The aforementioned three CORDICs based QR-RLS triangularization part in Figure 6.11 (c) can be further transfered to a two cell based architecture, by time-sharing the 1st and 2nd CORDIC elements, as shown in Figure 6.13 [77].

6.6 Adaptive Equalisation Verification

To demonstrate and verify the correct numerical and algorithmic performance, this section presents the simulation of adaptive equalisation using both the floating and fixed point QR-RLS based on the architecture we proposed in Chapter 6.8. Figure 6.14 shows the block diagram of the simulation environment. The random signal applied to the channel input consists of a Bernoulli sequence. The impulse response of the channel is described by the product of raised cosine and complex Gaussian in Eq. 6.11 (the values of all taps are listed in Table 6.2).

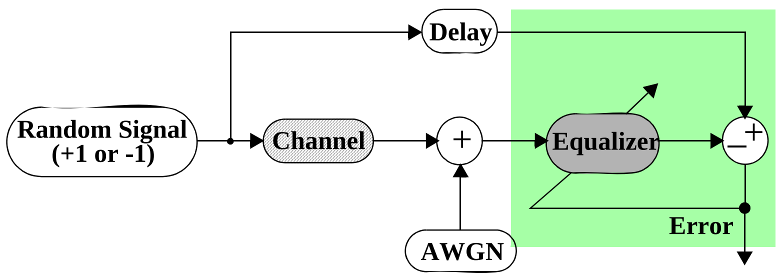


Figure: 6.14: Adaptive equalization for computer experiment

Table 6.2: Channel Impulse Response

Weight	0	1	2
Real	0.48748	-1.59373	-0.56016
Imaginary	0.51943	-0.71432	-0.63114

$$h_n = \begin{cases} \frac{1}{2}\left[1 + \cos\left(\frac{2\pi}{W}(n-2)\right)\right] \cdot [a(n) + b(n)j], & n = 1, 2, 3 \\ 0, & otherwise \end{cases} \quad (6.11)$$

where the parameter W controls the eigenvalue spread of the correlation matrix of the tap inputs in the equalizer. $a(n)$ and $b(n)$ are both Gaussian with zero mean and variance $\sigma^2 = 1$. The AWGN block also produces white Gaussian noise with zero mean and $\sigma^2 = 0.001$. The equalizer is a QRD-RLS adaptive filter containing a systolic array. In the following simulations, the tap-number of the equaliser is set to 11.

The convergence speed of QR-RLS algorithm depend on the initial value of the matrix $\boldsymbol{R}$. Generally the initial $\boldsymbol{R}$ is chosen as a scalar matrix. The smaller the size of the matrix, the faster the convergence. To demonstrate the convergence property of the algorithm, the initial value of the scalar matrix was chosen as 0.5.

The eigenvalue spread is set to $W = 3.5$. An approximation to the ensemble-average learning curve of the adaptive equaliser is obtained by averaging the instantaneous-squared-error curve over 30 independent iterations of the computer

Table 6.3: Coefficients of the Fixed Point QR-RLS based

	Floating point		Fixed point	
Weight	Real	Imaginary	Real	Imaginary
0	-0.0012	-0.00129	-0.00142	-0.00099
1	-0.00636	-0.00394	-0.00685	-0.00376
2	-0.02046	-0.00621	-0.02077	-0.00608
3	-0.05805	0.00118	-0.05844	0.00128
4	-0.1524	0.04827	-0.1528	0.04828
5	-0.3691	0.2531	-0.3696	0.2532
6	0.1833	-0.05087	0.183	-0.0509
7	-0.08088	-0.00404	-0.08116	-0.00411
8	0.03224	0.01298	0.03201	0.01287
9	-0.01074	-0.00999	-0.01102	-0.01012
10	0.00191	0.00362	0.00167	0.00350

simulation. From Figure 6.15 to Figure 6.18, each figure plots the ensemble-average learning curves of both the floating and fixed point QR-RLS based adaptive equalizer. The fixed point wordlength varies from signed 23-bit down to 17-bit, with 6-bit representing the integer part to prevent overflow. As can be seen, the convergence rates of the proposed fixed point QR-RLS are very close to the floating point references in Figure 6.15, 6.16 and 6.17. However, the signed 17-bit or less wordlength is not favoured in this scenario to produce the satisfied numerical stability. As can be seen in Figure 6.18, the round-off noise reduces the accuracy of QR-RLS implementation, due to its large growth with the increase of QRD iteration time. Table 6.3 also lists the resulting filter coefficients of both the floating point and 19-bit fixed point QR-RLS designs, both designs converge to a very similar solution with only fixed point round off noise being the differences.

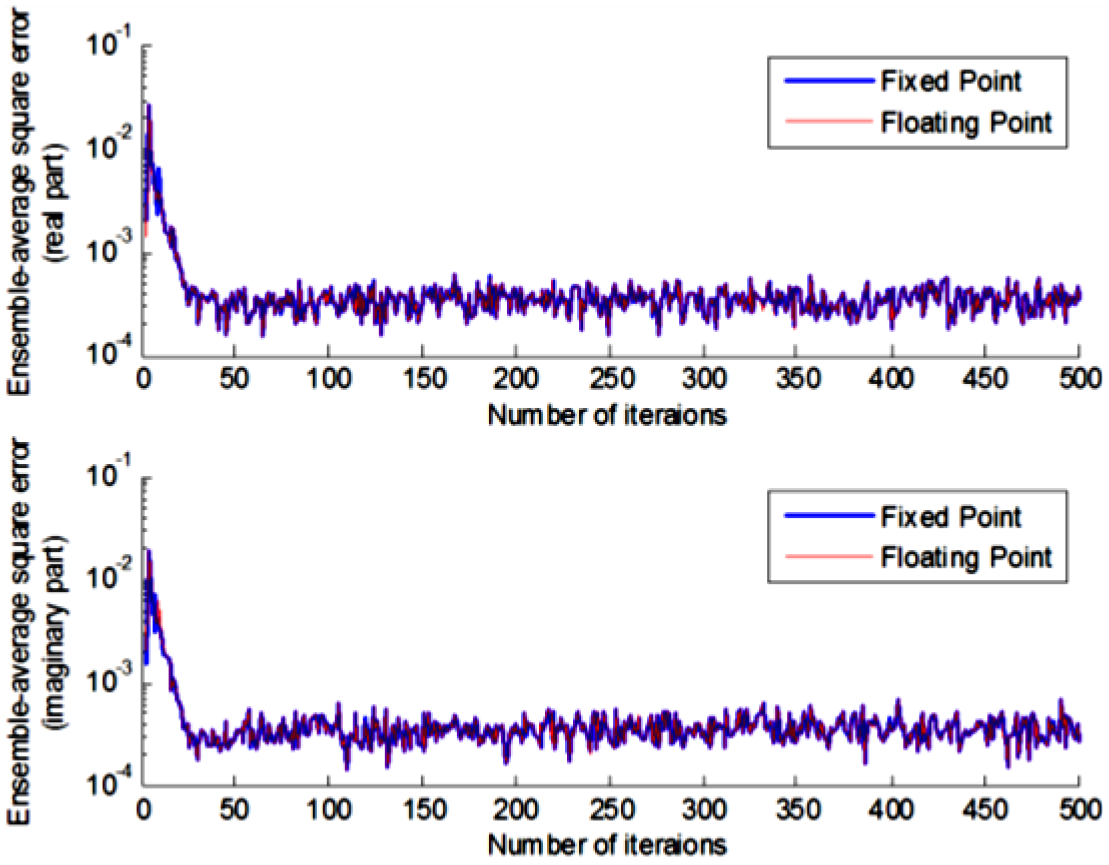


Figure: 6.15: Learning curves of the floating vs 23-bit filxed point QR-RLS

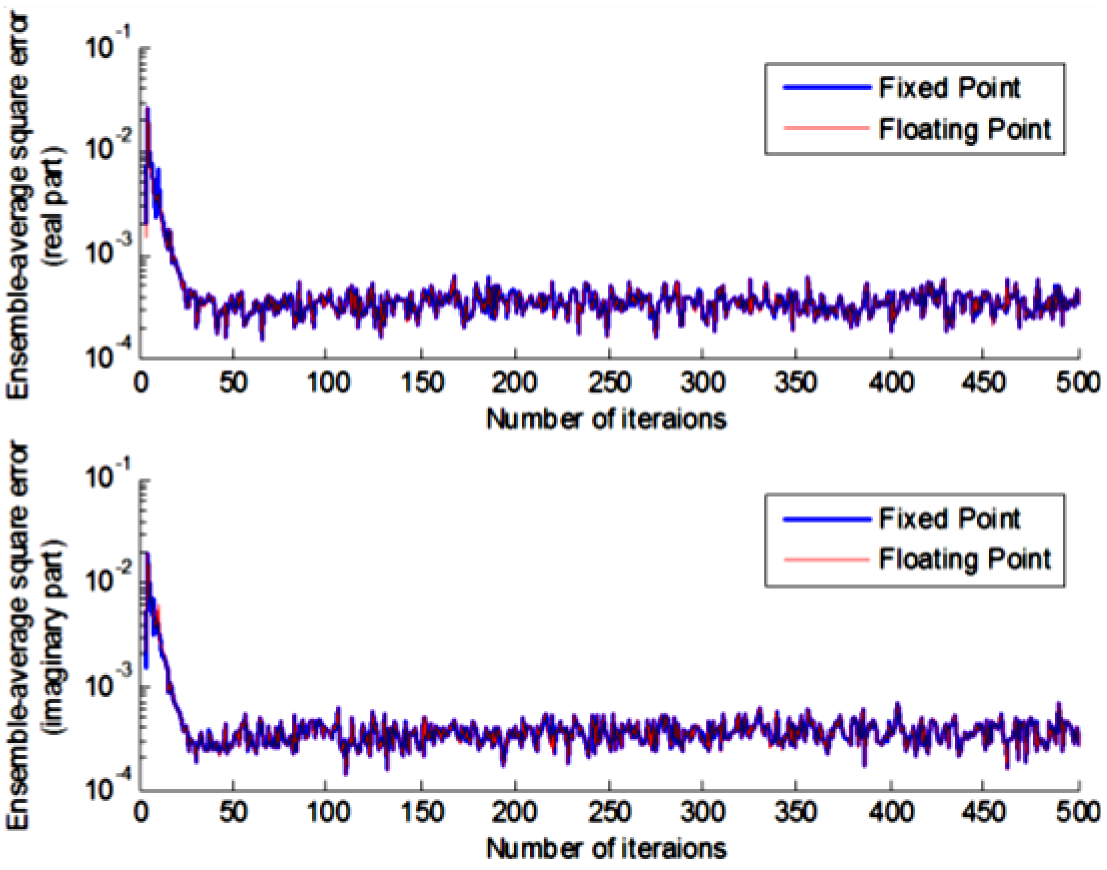


Figure: 6.16: Learning curves of the floating vs 21-bit filxed point QR-RLS

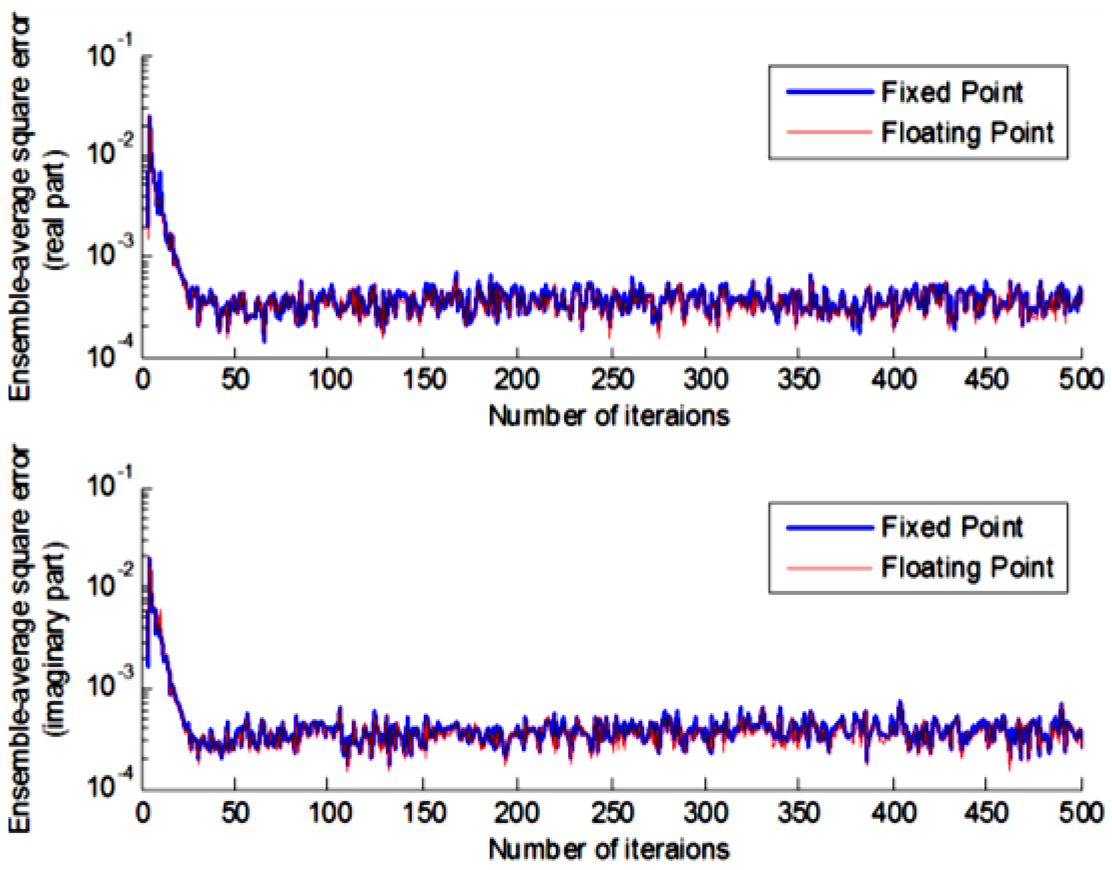


Figure: 6.17: Learning curves of the floating vs 19-bit filxed point QR-RLS

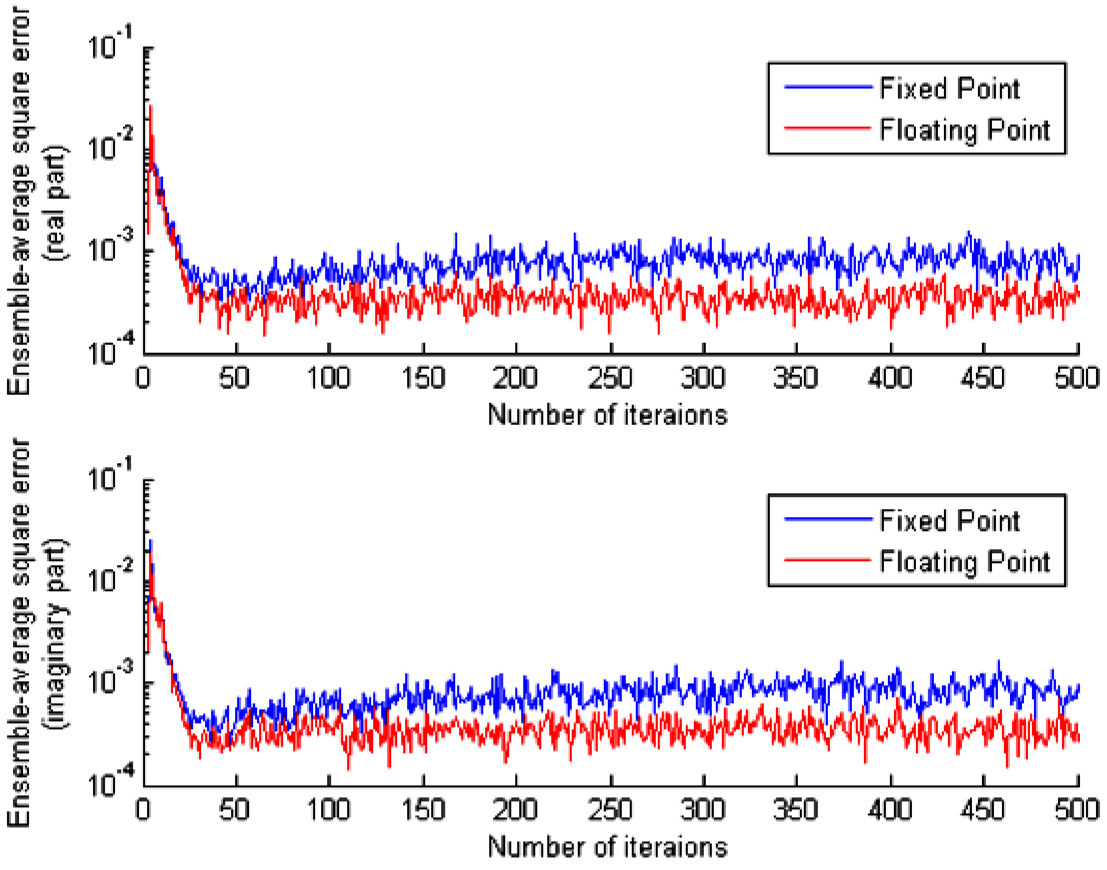


Figure: 6.18: Learning curves of the floating vs 17-bit filxed point QR-RLS

6.7 Performance

Table 6.4 compares the FPGA ultilisation of the 16-iteration (i.e. $n = 16$) and 18-bit wordlength DACR based 'Single PE + MAC' 4×4 QR-RLS, with the work in [24]

Table 6.4: Synthesis results for CORDIC QR Givens (Size $\mathbf{4 \times 4}$)

Type	Part	Slice	FF	LUT	DSP48	BRAM
Standard CORDIC Lookup [24]	Triangularization	1598	2549	2630	9	5
	BS	1932	2862	3286	4	1
	Total	3530	5411	5916	13	6
Pipeline-Interleaving DACR	Triangularization	1252	1953	1815	0	4
	MAC	463	392	631	8	0
	Total	1715	2345	2446	8	4

Note : BS = back-substitution

Table 6.5: Resource utilization of the $\mathbf{4 \times 4}$ QR-RLS for the various wordlengths

Parts	Bits	Slices	LUTs	FFs	DSP48s	BRAMs
Triangular-ization	20	1576	2473	2105	0	5
	22	1744	2817	2284	0	5
	24	2147	3275	2555	0	5
	26	2319	3701	2778	0	5
MAC	20	502	476	732	8	0
	22	546	476	732	8	0
	24	558	512	776	8	0
	26	578	554	820	8	0
QR-RLS Total	20	2078	2949	2837	8	5
	22	2290	3293	3016	8	5
	24	2705	3787	3331	8	5
	26	2897	4255	3598	8	5

which uses the classic back-substitution for weight extraction. Xilinx ISE 10.1 was used to target a Virtex-4 xc4vsx35-10ff668 device. The total estimated power consumption (using Xilinx Xpower utilities [78]) is 1064mW (the quiescent power is 459mW, and the dynamic power is 604mW). Table 6.5 shows the FPGA ultilisation of the 16-iteration DACR based ‘Single PE + MAC’ QR-RLS processor for the various wordlengths.

Based on Eq. 6.10, Table 6.6 provides the timing information for several configurations of the input data set under the triangularisation and MAC parts (back-substitution part for [24]) of the QR-RLS processor presented in this thesis. This table

Table 6.6: Execution Time Of The QR-RLS for Matrix Sizes

Type	R Size	Cycles for Triangular'tn	Cycles for BS/MAC	Total Cycles	Time (μs) for 250 MHz Clock
Ref [24]	5 x 5	2540	255	2795	11.18
	7 x 7	5656	371	6027	24.11
	9 x 9	10476	495	10971	43.88
Pipeline-Interleaving DACR	5 x 5	985	9	994	3.976
	7 x 7	1925	9	1934	7.736
	9 x 9	3177	9	3186	12.744

Table 6.7: Cost and Speed Performances of the 9×9 QR Decomposition RLS Implementations

Type	Triangularization cost (Slice)	BS/MAC cost (Slice)	Total cost (Slice)	Triangularization delay (μs)	BS/MAC delay (μs)	Total delay (μs)
Ref [23]	2600 LEs ≈ 1300 Slices	N/A (Nios)	1300+	198.11	120	318.11
Pipeline-Interleaving DACR	967	443	1410	21.18	0.06	21.24

shows that the newly proposed architecture can significantly reduce the time for weights extraction, and also save the hardware implementation cost.

The proposed QR-RLS design also compares favourably with a 9-element, 16-bit reference design described in [23]. As shown in Table 6.7, the reference design, running from a 150MHz clock, can sustain an update delay of 198.11μs, whereas the proposed design can sustain a latency of approximately 21.18μs at the same clock rate. This equates to an approximate ten-fold performance increase. The Altera design also uses in excess of 2600 logic elements (approximately 1300 Xilinx slices [79]), and also requires an additional embedded soft processor.

6.8 CORDIC Approximate-Scaling-Compensation

As mentioned in Section 4.5, the throughput of a fully pipelined CORDIC based QR-RLS system is limited by the scaling compensation. Furthermore previous research on Scaling Free CORDIC algorithm is not applicable to QR-RLS application. In this section, a novel architecture is presented to approximately compensate the scaling factor inside the QRD recursive loop without changing the CORDIC algorithm. This results in fewer pipeline stages inside the loop, so the throughput of the QR-RLS system can further improve.

According to Table 3.2 and Eq. 6.8, during the QR recursive operation, the new $\boldsymbol{x}$ and d values are added respectively to the upper triangular matrix $\boldsymbol{R}$ and the column $\boldsymbol{p}$. The decompositon in Eq. 6.12 then converts the augmented matrix $[\boldsymbol{R}|\boldsymbol{x}]$ to a new upper triangular matrix by using Given's rotations to zero the elements below the main diagonal. Eq. 6.12 can be further transformed to Eq. 6.13. According to Chapter 4.4.1, each CORDIC output must be scaled by a constant $S = K^{-1} \approx 0.6072$ to give a true circular rotation. This constant scaling can be realised by Reduced Slice Graph (RSG) with more than 1 pipeline stages (the number of pipeline stage depends on the precision required), as shown in Figure 6.19 (*a*). In Figure 6.19 (*b*), this S scaling is circumvented by applying a $S_1 = 0.5 + 0.125 = 0.625$ approximate scaling. This can be easily achieved by the combination of a 1-bit right shift and a 3-bit right shift, with only 1 pipeline stage, and the number of pipeline stage is independent with the precision.

$$\begin{bmatrix} (\cos\theta_1)_{jj} & 0 & & (\sin\theta_1 e^{j\theta_4})_{ji} \\ 0 & 1 & & 0 \\ & & & 0 \\ (-\sin\theta_1)_{ij} & 0 & 0 & (\cos\theta_1 e^{j\theta_4})_{ii} \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{0(N-1)} & p_0 \\ 0 & r_{11} & r_{1(N-1)} & p_1 \\ & & & \\ 0 & 0 & r_{(N-1)(N-1)} & p_{(N-1)} \\ x_0 & x_1 & x_{(N-1)} & d \end{bmatrix}$$

$\boldsymbol{Q}^T$

$$= \begin{bmatrix} \overline{r_{00}} & \overline{r_{01}} & \overline{r_{0(N-1)}} & \overline{p_0} \\ 0 & \overline{r_{11}} & \overline{r_{1(N-1)}} & \overline{p_1} \\ & & & \\ 0 & 0 & \overline{r_{(N-1)(N-1)}} & \overline{p_{(N-1)}} \\ 0 & 0 & 0 & \xi \end{bmatrix} \tag{6.12}$$

$$\begin{bmatrix} (\cos\theta_1)_{jj} & 0 & (\sin\theta_1 e^{j\theta_4})_{ji} \\ 0 & 1 & 0 \\ & & \\ \left(\dfrac{-\sin\theta_1}{(S/S_1)^2}\right)_{ij} & 0 & \left(\dfrac{\cos\theta_1 e^{j\theta_4}}{(S/S_1)^2}\right)_{ii} \end{bmatrix} \begin{bmatrix} r_{00} & r_{01} & r_{0(N-1)} & p_0 \\ 0 & \dfrac{r_{11}}{(S/S_1)^2} & \dfrac{r_{1(N-1)}}{(S/S_1)^2} & \dfrac{p_1}{(S/S_1)^2} \\ & & & \\ 0 & 0 & \dfrac{r_{(N-1)(N-1)}}{(S/S_1)^{(2N-2)}} & \dfrac{p_{(N-1)}}{(S/S_1)^{(2N-2)}} \\ x_0 & x_1 & x_{(N-1)} & d \end{bmatrix}$$

$\overline{\boldsymbol{Q}}^T$

$$= \begin{bmatrix} \overline{r_{00}} & \overline{r_{01}} & \overline{r_{0(N-1)}} & \overline{p_0} \\ 0 & \dfrac{\overline{r_{11}}}{(S/S_1)^2} & \dfrac{\overline{r_{1(N-1)}}}{(S/S_1)^2} & \dfrac{\overline{p_1}}{(S/S_1)^2} \\ & & & \\ 0 & 0 & \dfrac{\overline{r_{(N-1)(N-1)}}}{(S/S_1)^{(2N-2)}} & \dfrac{\overline{p_{(N-1)}}}{(S/S_1)^{(2N-2)}} \\ 0 & 0 & 0 & \dfrac{\xi}{(S/S_1)^{2N}} \end{bmatrix} \tag{6.13}$$

$S = K^{-1} \approx 0.6072$

$S_1 = 0.625$

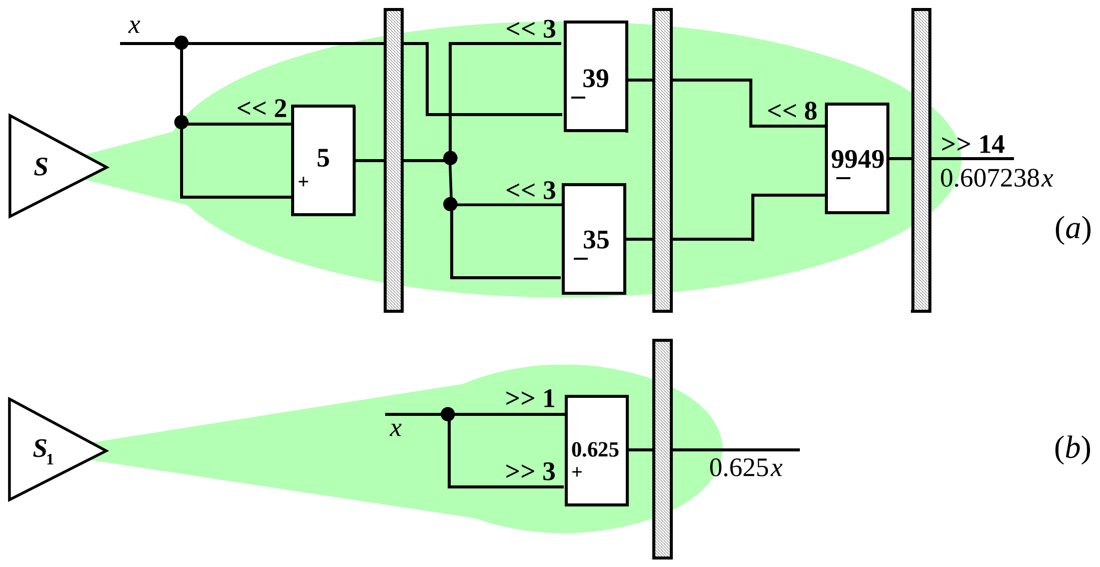


Figure: 6.19: (*a*) Scaling Compensation; (*b*) Approximate-Scaling-Compensation

This new CORDIC can result in the modified matrix $\overline{\boldsymbol{Q}}^T$ in Eq. 6.13. $\overline{\boldsymbol{Q}}^T$ scales the initial values of the r and p elements in $[\boldsymbol{R}|\boldsymbol{x}]$ and $[\boldsymbol{p}|d]$ by an additional factor $(S/0.625)^{-2}$ on each complex Givens updating iteration, namely $(S/0.625)^{-(2N-2)}$ on the N-th iteration. The initial values of all the r need to be pre-computed and stored in a memory. A final correction operation — scale by $S_2 = (S/0.625)^{2N}$ — is required on the output of the channel-interleaving CORDIC, as described in Eq. 6.14 and Figure 6.20.

$$\frac{\xi}{(S/S_1)^{2N}} \times S_2 = \frac{\xi}{(S/0.625)^{2N}} \times (S/0.625)^{2N} = \xi \qquad (6.14)$$

Since $S_2 = (0.6072/0.625)^{2N}$ is always smaller than 1, this Approximate-Scaling-Compensation scheme causes the bit growth of the QRD recursive loop, namely the S_2 scaling corrects such a bit growth which might damage the numerical precision of the design. Table 6.8 lists the S_2 values under different configurations. When QR decomposing an 11×11 matrix, the corresponding S_2 scaling equals to $\times$ 0.5303, i.e. this correction requires approximately a 1-bit right shift, hence the numerical property of the Approximate-Scaling-Compensation architecture can be maintained by increasing 1-bit accuracy to the original wordlength of the recursive loop. When the

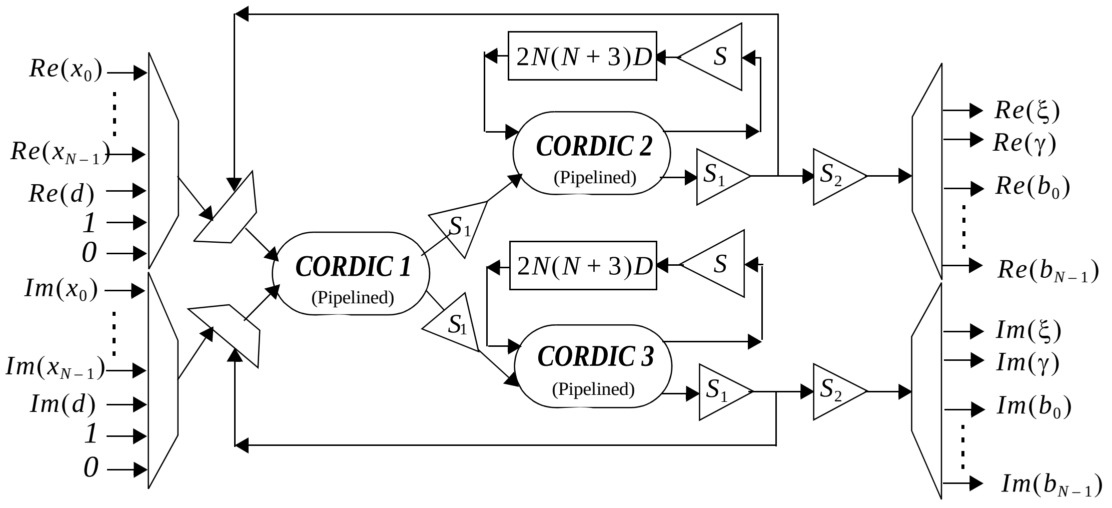


Figure: 6.20: CORDIC Approximate-Scaling-Compensation QR-RLS

Table 6.8: S_2 values under different configurations

	S_2 value
$N = 4$	0.7940
$N = 11$	0.5303
$N = 20$	0.3156
$N = 30$	0.1773
$N = 40$	0.0996

matrix size further increases to 40 by 40, only 5-bit extension (to represent $\times 0.0996$) is required by the original wordlength. Hence, the newly proposed Approximate-Scaling-Compensation architecture does not require the much higher bit-width than that of the standard CORDIC Scaling-Compensation QR-RLS, particularly for the small matrice size.

Same with Figure 6.13, the three Approximate-Scaling-Compensation CORDICs based triangularization in Figure 6.20 can also be further transfered to a two units based architecture, as shown in Figure 6.21. Table 6.9 lists the FPGA ultilisation of the 16-iteration (i.e. $n = 16$) and 18-bit wordlength 'Approximate-Scaling-Compensation' DACR based 'Single PE + MAC' 4×4 QR-RLS.

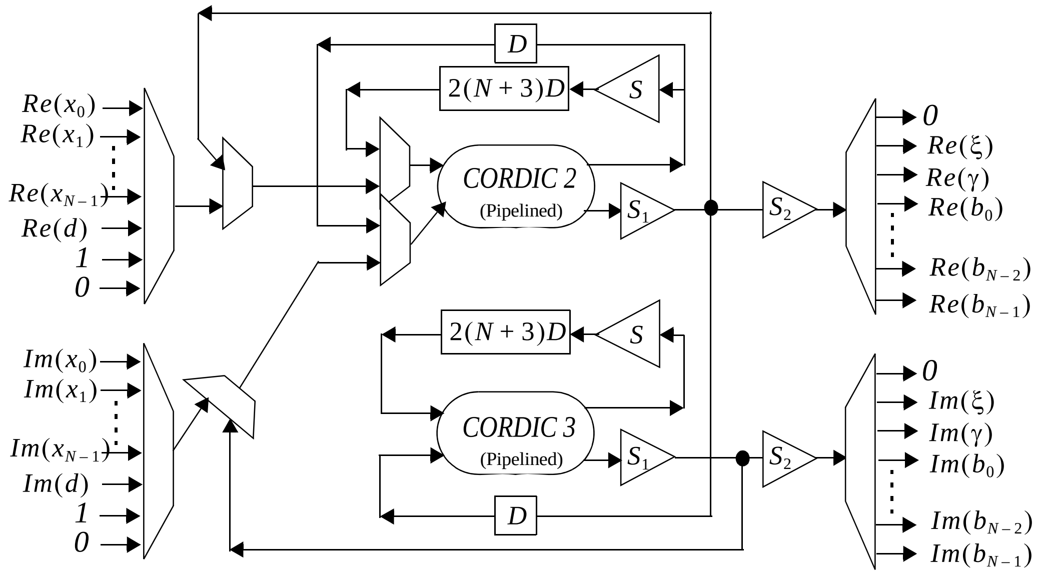


Figure: 6.21: Double Approximate-Scaling-Compensation CORDICs based QRD-RLS triangularization

Table 6.9: Synthesis results for CORDIC QR Givens (Size $\mathbf{4 \times 4}$)

Type	Part	Slice	FF	LUT	DSP48	BRAM
Approximate-Scaling-Compensation	Triangularization	1285	2042	1846	0	4
	MAC	463	392	631	8	0
	Total	1748	2434	2477	8	4

Table 6.10 compares the execution time (for one iteration of QR decomposition) of this section's modified CORDIC based QR-RLS and the original 'Scaling-Compensation' design in Chapter 6.5. CLK represents the maximum clock rate, N is the number of weights, n stands for the total CORDIC iterations and d is the number of clock cycle delays introduced by the CORDIC scaling operations (assuming both the 'Scaling-Compensation' and 'Approximate-Scaling-Compensation' designs request the same number of pipelining levels). Last but not least, the m represents the clock cycle delays introduced by the MAC part and the new 'Approximate-Scaling-Compensation' architecture to accelerate the QR-RLS processor.

Based on Table 6.10, Table 6.11 lists the timing information for several configurations of the input data set under the triangularisation and MAC parts of the

Table 6.10: Execution time of the single PE QR-RLS

Design	Execution time
Scaling-Compensation	$\frac{(2n+2d+1)\times N^2+2N+m}{CLK}$
Approximate-Scaling-Compensation	$\frac{(2n+3)\times N^2+(2+d)\times N+m}{CLK}$

QR-RLS processor in this section. Comparing with the information in Table 6.6, the newly proposed 'Approximate-Scaling-Compensation' scheme can further reduce the execution time of QR-RLS processor.

Table 6.11: Execution Time Of The QR-RLS for Matrix Sizes

Type	*R* Size	Cycles for Triangularization	Cycles for BS/MAC	Total Cycles	Time (μs) for 250 MHz Clock
Approximate-Scaling-Compensation	5 x 5	900	9	909	3.636
	7 x 7	1750	9	1759	7.036
	9 x 9	2880	9	2889	11.556

6.9 Conclusion

This chapter has described the design and implementation of the complex valued 'Single PE + MAC' QRD-RLS processor. The pipeline-interleaving coarse angle CORDIC based quadrant correction free DACR is employed to combine the low hardware consumption with the benefit of scalability. The Scaling-Partial-Compensation algorithm plays essentially the role of increasing the sampling rate of our proposed QR-RLS processor.

The principal contribution of this chapter is applying a pipeline-interleaving scheme to multiplex all the PEs of complex value QR-RLS systolic array together. The pipeline-interleaving CORDIC based single PE QR-RLS downdating processor can be created which combines low hardware consumption with the benefit of speed and scalability. By comparing with the reference design in [23] [24] which implements the IC/DCs by mult-add function units, it is shown that the new pipeline-interleaving coarse angle CORDIC based complex value QR-RLS is a low cost implementation, and directly yields the adaptive weight values without the need for explicit back-substitution.

Chapter 7

Annihilation-Reordering Look-ahead Technique for Complex Givens Rotation based QR-RLS Filter

7.1 Introduction

According to Section 3.10, in recent years, various Look-ahead technologies have been used in the architecture and VLSI implementation of the communication physical layer design to improve the clocking speed of adaptive DSP algorithms. To increase the throughput of QR-RLS adaptive filters, Annihilation-Reordering look-ahead Transformed (ART) [41] was proposed to achieve more fine-grain pipelining of real valued QR-RLS implementation by retiming.

The novel approach proposed in this chapter pipelines a Complex value Annihilation-Reordering look-ahead Transformed (C-ART) updated QRD-RLS array at the fine-grained level. The architecture critical path can be reduced through retiming transformation. Thereafter a folding transformation is used to reduce the number of the pipelined CORDIC stages inside each processing element (PE).

The rest of this chapter is organized as follows. Chapter 3.10 briefly reviews the existing adaptive DSP applications using the Look-ahead transformation. In Chapter 3.10.1, the real valued ART technique for speeding up the QR-RLS transformation is reviewed. Chapter 7.2 proposes the architecture of C-ART updated QR-RLS. Chapter 7.3 presents the simulation of an adaptive equalization and beamforming system to demonstrate that the new technology achieves the same ensemble-average learning curve and beampattern with the classic sequential updated QR-RLS algorithm. Finally, the methodology of fine-grain level mapping is presented in Chapter 7.4, and the synthesis implementation results on Xilinx FPGA demonstrates that the proposed structure results in a higher throughput.

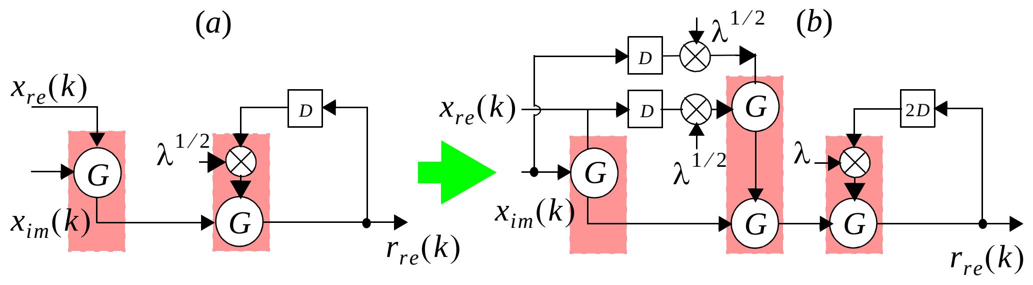


Figure: 7.1: (*a*) Sequential transformed BC; (*b*) C-ART updated BC

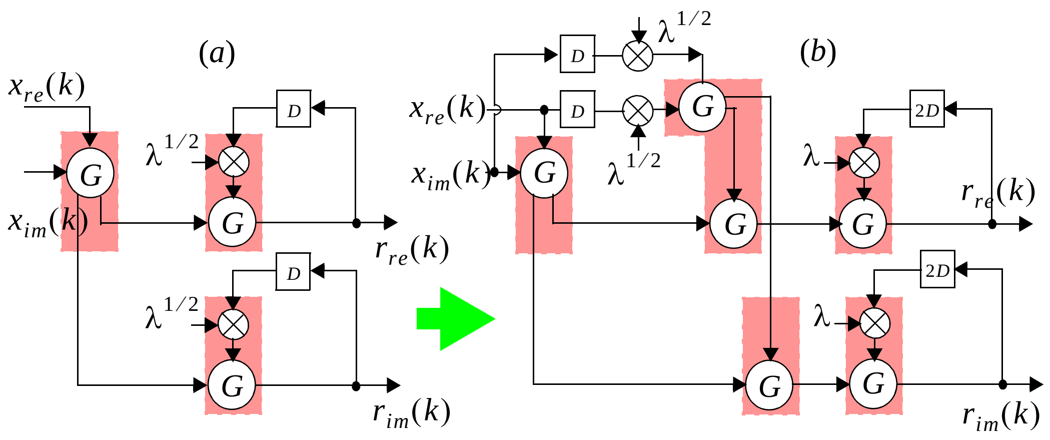


Figure: 7.2: (*a*) Sequential transformed IC/DC; (*b*) C-ART updated IC/BC

7.2 ART Complex valued QR-RLS

The annihilation-reordering look-ahead transformation of DACR can be derived from the real valued ART in Figure 3.18. After applying the ART transformation with pipelining level 2 to the two Givens rotations based BC in Figure 7.1 (*a*) and the three Givens rotations based IC/DC in Figure 7.2 (*a*), the resulting SFGs are given in Figure 7.1 (*b*) and Figure 7.2 (*b*) respectively. Similar to Figure 3.16 (*b*), the design sample rate can also be increased by a factor of two after redistributing the two delay elements in the feedback loop.

In contrast with the real valued annihilation-reordering look-ahead updated QR-RLS in Eq. 3.30, Eq. 7.1 presents the ART update of the complex valued QRD-RLS algorithm with the block size M, where the hermitian transpose “H” is used in the downdating matrix:

$$\boldsymbol{Q}(k)\begin{bmatrix}\lambda^{M/2}\boldsymbol{R}(k-M) & \lambda^{M/2}\boldsymbol{p}(k-M) & \lambda^{-M/2}\boldsymbol{R}^{-H}(k-M)\\ \boldsymbol{x}_M^T(k-1) & d_M(k-1) & 0_M\end{bmatrix}$$
$$= \begin{bmatrix}\boldsymbol{R}(k) & \boldsymbol{p}(k) & \boldsymbol{R}^{-H}(k)\\ 0 & \xi(k) & b(k)\end{bmatrix} \tag{7.1}$$

The architecture of the Complex value Annihilation-Reordering look-ahead Transformed (C-ART) QRD-RLS systolic array (with $M = 2$ demonstrated next) is shown in Figure 7.3. The fine-grain levels of all PEs are also provided in Figure 7.4. Note that the EC elements consist several arithmetic units labeled by ‘E’, this kind of units can be implemented by the linear CORDIC arithmetic [47]. From an implementation point of view, the number of ultilized CORDIC elements inside each complex valued PE is still proportional to the pipelining level M. When the pipelining level equals M, each complex Givens rotation based BC contains $2M$ circular CORDICs, $2 \times (M+1)$ circular CORDICs required by every IC/DC, $4 \times M$ and $4 \times M + 2$ linear CORDICs are involved in the error calculator and each EC seperately. The total complexity is around $O(2MN^2)$, which is double the complexity of the real valued system in [47]. When the block updating size increases to three, the amount of ultilised CORDICs in BC are six, and eight CORDICs required by every IC/DC, as shown in Figure 7.5. Then in Figure 7.6, if the look-ahead pipelining level further rises to four, correspondingly eight CORDICs are employed in each BC, and ten CORDICs required by every IC/DC.

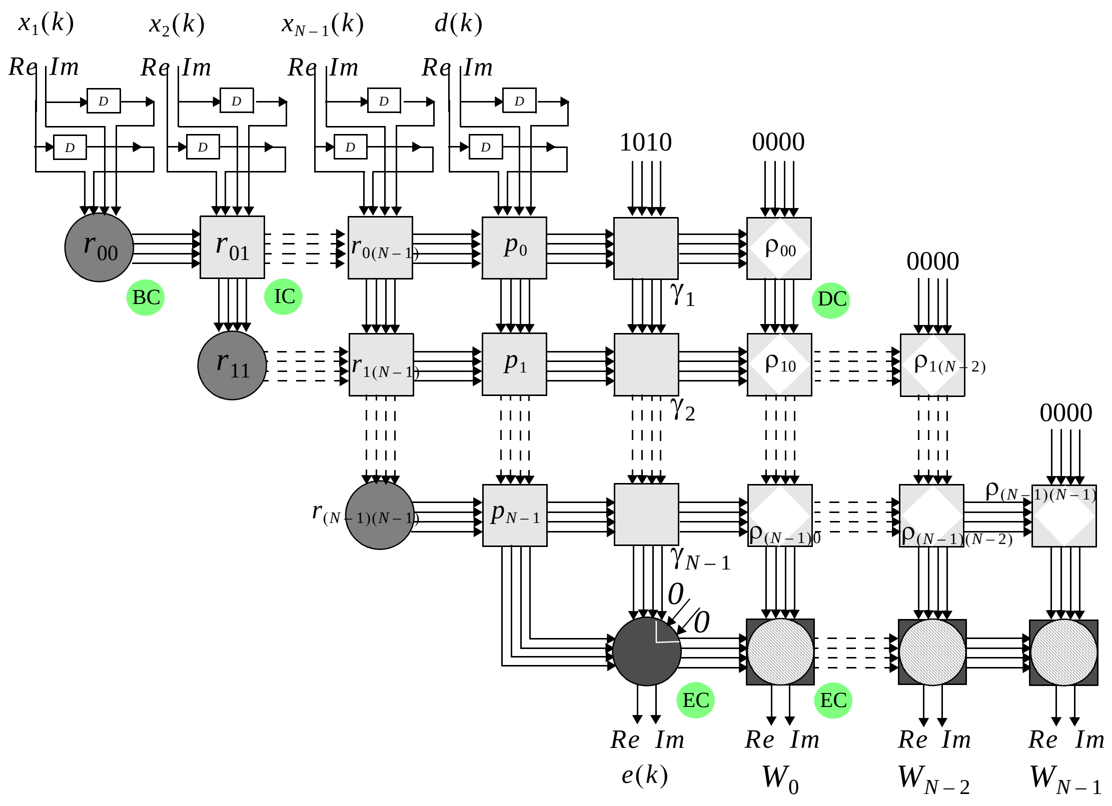


Figure: 7.3: C-ART updated QR-RLS ($M = 2$)

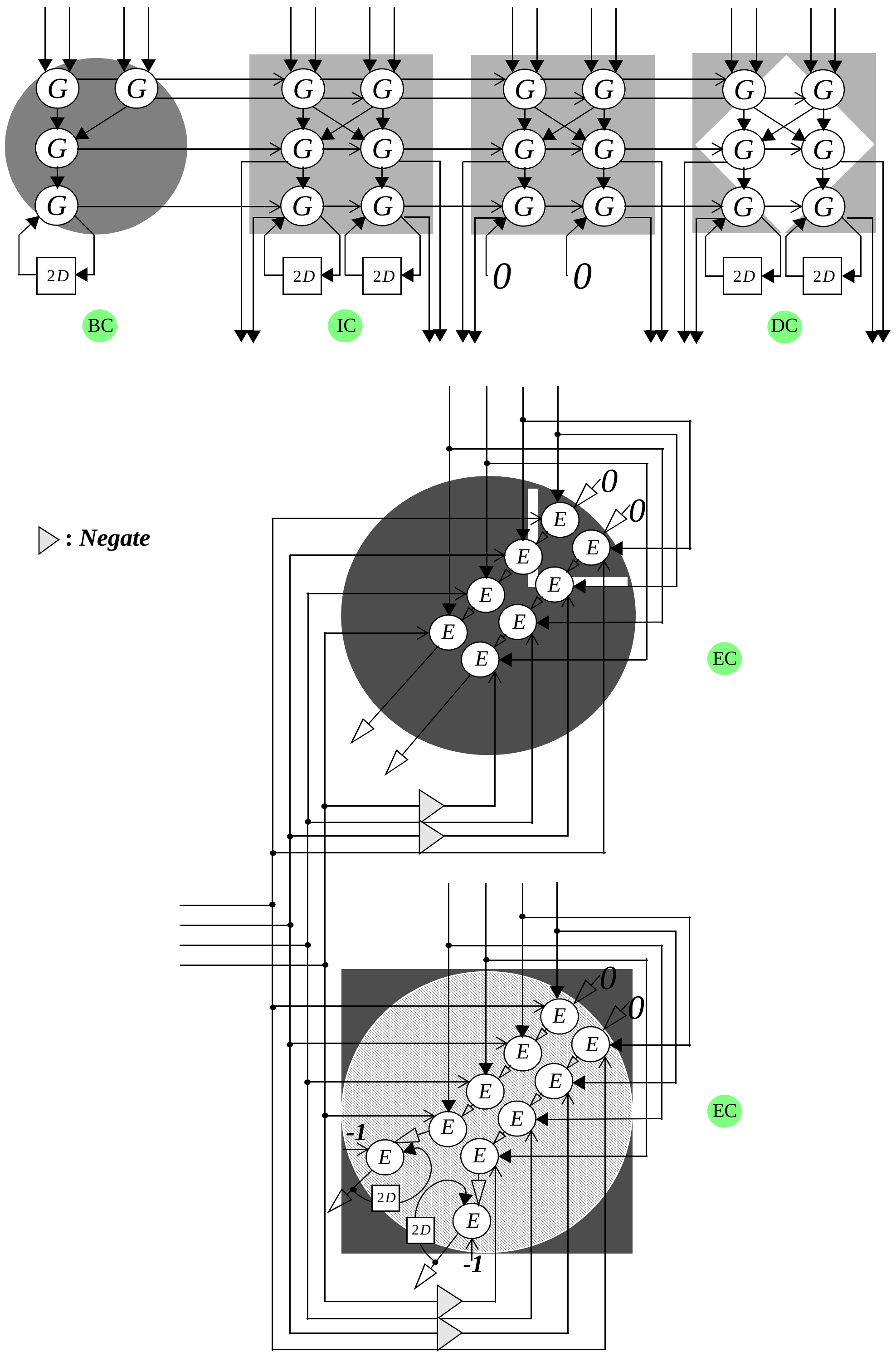


Figure: 7.4: C-ART updated QR-RLS ($M = 2$) *cont'd*

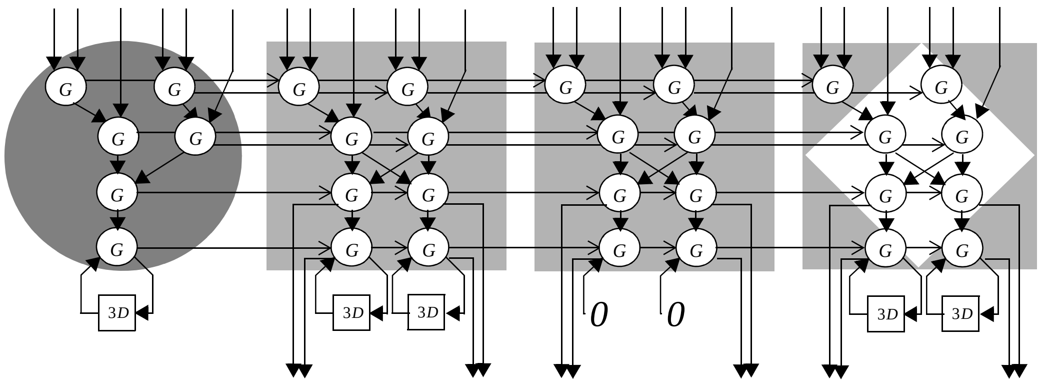


Figure: 7.5: C-ART updated QR-RLS ($M = 3$)

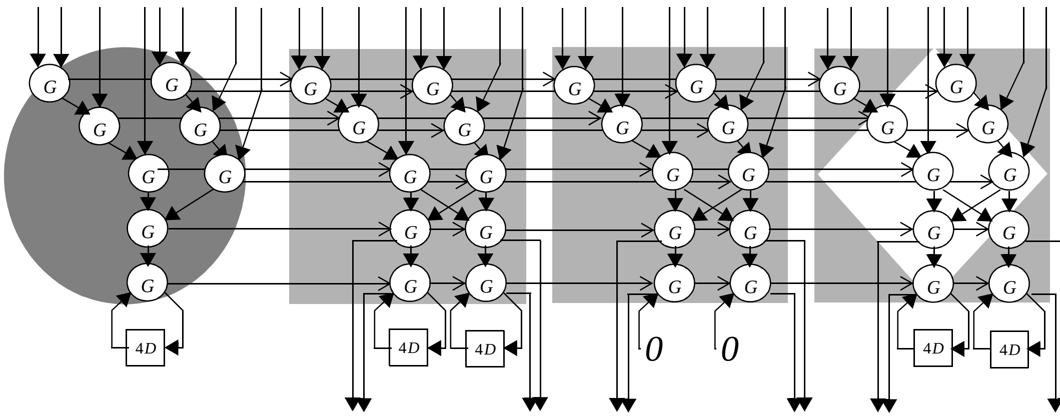


Figure: 7.6: C-ART updated QR-RLS ($M = 4$)

7.2.1 Pipeline C-ART QR-RLS PEs Through Retiming

Retiming is an effective transformation technique used to change the location of delay elements in a circuit without affecting the input/output characteristics of the circuit [12] [16] [41]. Through retiming, the maximum clocking rate of a circuit is increased by ensuring the critical path is minimised. Figure 7.7 shows how the boundary and internal cells in Figure 7.4 are pipelined by retiming the extra delay elements, highlighted as the additional cut-set registers in Figure 7.7, added at their inputs (in general, an M-level pipelined complex Givens rotation based PE can be obtained by retiming the $2M$-level lumped delay elements at its inputs). After this retiming, the propogation delays of critical paths in BC, IC and DC are reduced by 1/

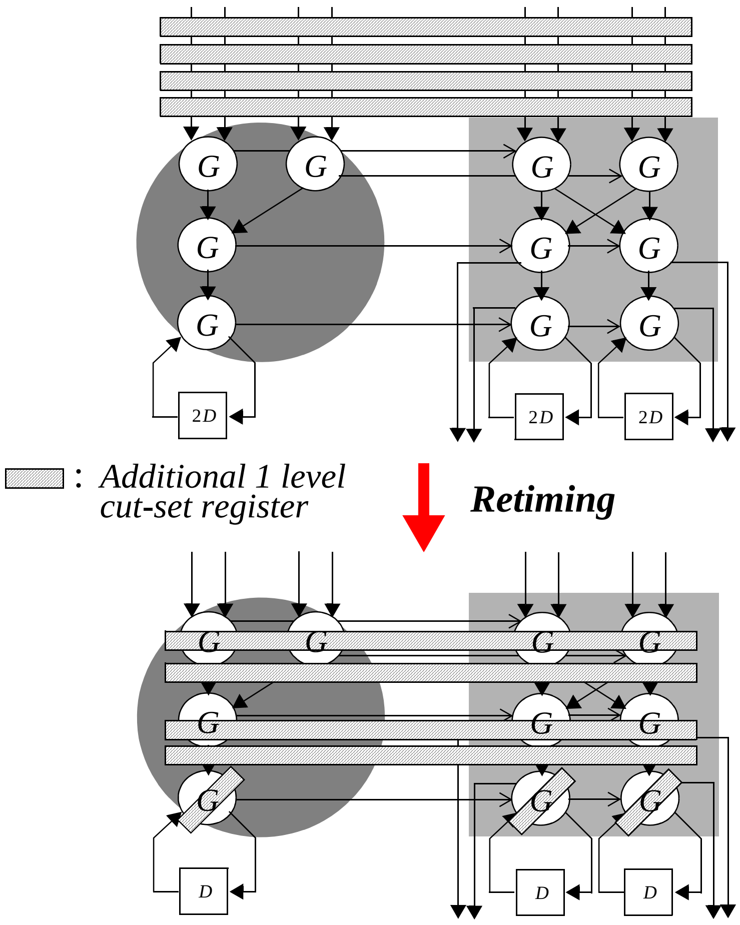


Figure: 7.7: Retiming the processing elements of C-ART updated QR-RLS ($M = 2$)

2 ($1/M$ in general), i.e the new proposed technology could speed up the QR-RLS algorithm to work on 2 (M in general) times of its original maximum clock/sampling rate.

7.3 Simulation Verification

This part presents the simulation verification of both the adaptive equalization and beamforming systems to prove that the new C-ART QR-RLS technology can achieve the same ensemble-average learning curve and beampattern with the classic sequential

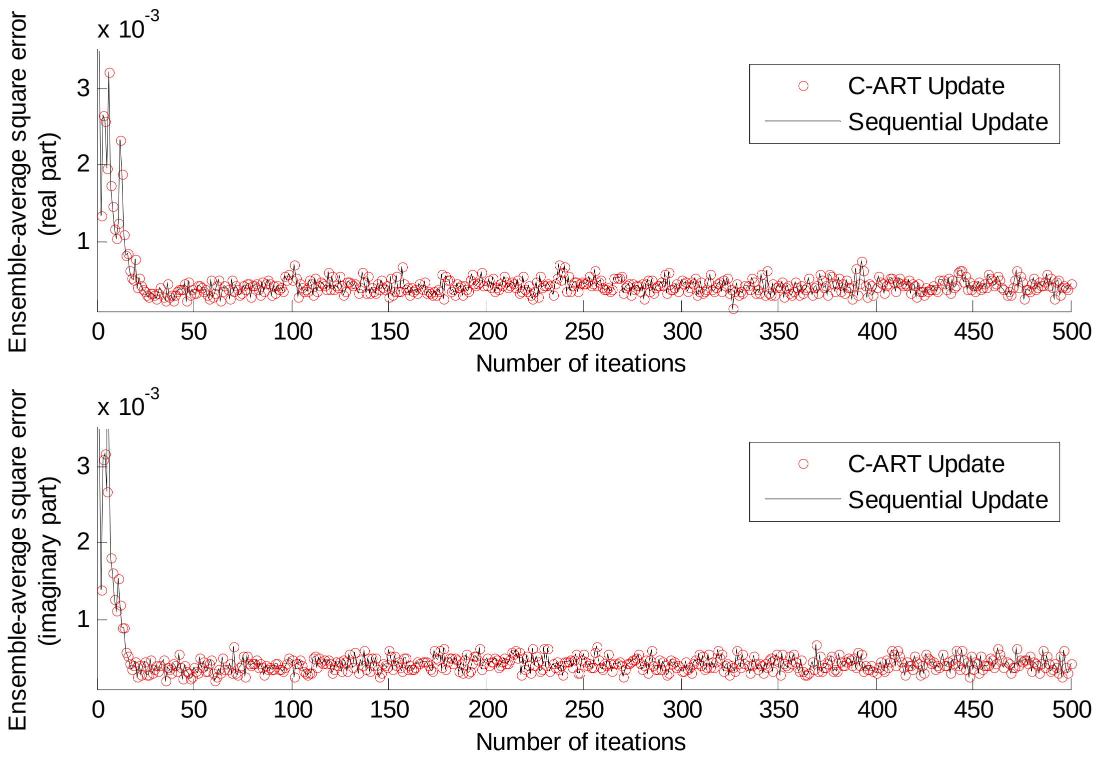


Figure: 7.8: Comparision between the learning curves of the sequential and $M = 2$ C-ART updated QR-RLS algorithms

updated algorithm.

7.3.1 Adaptive Equalization

This section presents the simulation of adaptive equalization using both the sequential and Look-ahead updating QR-RLS algorithms. The simulation system settings are the same with those in Section 6.6. Figure 7.8 shows the real and imaginary part of the learning curves of floating point based ‘classic’ QR-RLS algorithm with the sequential update. For $W = 3.5$, an approximation to the ensemble-average learning curve of the adaptive equalizer is obtained by averaging the instantaneous-squared-error curve over 100 independent trials of the computer experiment. When we replace the sequential QR update based equalizer with the proposed annihilation reordering Look-ahead transformed array, the real and imaginary parts of the ensemble-average

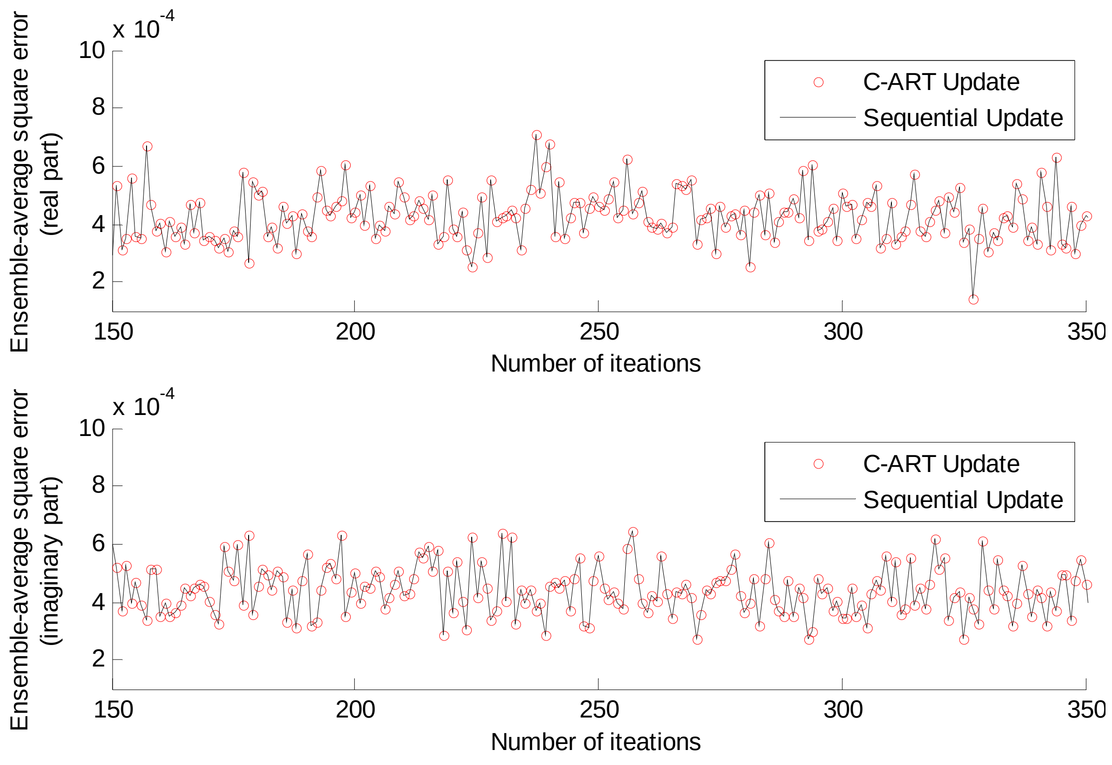


Figure: 7.9: Comparision between the learning curves of the sequential and $M = 2$ C-ART updated QR-RLS algorithms (zoomed in)

learning curve of the adaptive equalizer are shown in Figure 7.8. As can be seen in Figure 7.9 which is the zoomed in version of Figure 7.8, the convergence rate of the learning curves of the proposed C-ART updated QR-RLS are “exactly” the same with that of the classic sequential updated one. Hence with simulation we verify that the $M = 2$ C-ART is numerically identified to the standard complex QR algorithm.

7.3.2 Generalized Sidelobe Canceller (GSC) beamforming

Beamforming is a spatio-temporal technique used to enhance a target signal while reducing the amount of noise and interference using an array of sensors. The linearly constrained minimum variance (LCMV) beamformer is a specific type of adaptive beamformer that uses the direction of arrival to differentiate between the target signal and interference. It attempts to completely null out the interference by minimising the

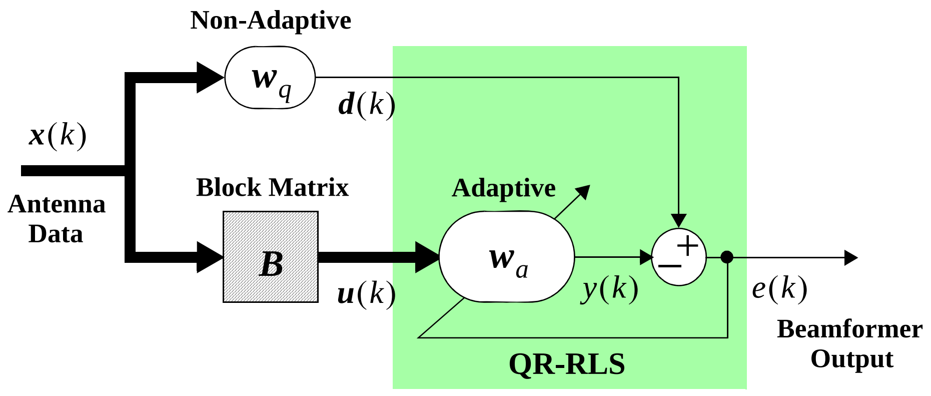


Figure: 7.10: GSC beamforming for computer experiment

power of the output signal $e(k)$ while maintaining the integrity of the target signal. A generalised sidelobe canceller (GSC) is a specific implementation of the LCMV which uses a blocking matrix to create a set of signals that are free of the target signals and thus are suitable for noise cancelling [80] [81].

The operation of the GSC is highlighted in Figure 7.10. Assuming the two spatially separated sources emitting signals — the target and interference signal, respectively — are in the narrowband form. Considering an N-element antenna array with sensor signals $x_n(k)$, $n = 0 \ldots N-1$. The input signal of the adaptive beamformer is [15]

$$\boldsymbol{x}(k) = [x_0(k) \;\; x_1(k) \;\; \ldots \;\; x_{N-1}(k)]^T \tag{7.2}$$

For each element of the array, the input signal is

$$x_n(k) = A_0 \exp(jn\theta_0) + A_1 \exp(jn\theta_1 + j\psi) + v_n(k) \tag{7.3}$$

where A_0 is the amplitude of the target signal and A_1 is the amplitude of the interfering signal. $v_n(k)$ is uncorrelated additive Gaussian noise with zero mean and unit variance. In this paper, the target-to-noise ratio (TNR) is held constant at 10dB; the interference-to-noise rato (INR) is assumed as 20dB. Measured in radians, the angles of target signal is $\theta_0 = -0.2\pi$, and $\theta_1 = 0$ is for the interference.

Based on $\boldsymbol{x}(k)$, the array response is steered by forming linear combinations of the beamformer outputs

$$e(k) = \sum_{n=0}^{M-1} w*_n \cdot x_n(k) = \boldsymbol{w}^H \boldsymbol{x}(k) \tag{7.4}$$

The LCMV problem for narrowband beamforming can be formulated as

$$min\{|e(k)|^2\}, \text{ subject to } \boldsymbol{Cw} = \boldsymbol{g}$$

where $\boldsymbol{w}$ contains the beamformer coefficients $[w_0 \ w_1 \ \dots \ w_{N-1}]$, $\boldsymbol{C}$ is the constraint matrix which specifies the spatio-temporal characteristics of the impinging target signal, and $\boldsymbol{g}$ specifies its gain response as it is passes through the beamformer.

The GSC decomposes the beamforming vector $\boldsymbol{w}$ into a fixed vector $\boldsymbol{w}_q = \boldsymbol{C}^H(\boldsymbol{CC}^H)^{-1}\boldsymbol{g}$ which operates as a delay-sum beamformer but with a specified frequency response, and a second branch containing a blocking matrix $\boldsymbol{B}$ and a coefficient vector $\boldsymbol{w}_a$,

$$\boldsymbol{w} = \boldsymbol{w}_q - \boldsymbol{B}^H \boldsymbol{w}_a \tag{7.5}$$

Using the definition of Eq. 7.5 in Eq. 7.4, we may express $e(k)$ as

$$e(k) = d(k) - y(k) = \boldsymbol{w}_q^H \boldsymbol{x}(k) - \boldsymbol{w}_a^H \boldsymbol{Bx}(k) \tag{7.6}$$

The fixed vector $\boldsymbol{w}_q$ passes the signal of interest plus interference (represented by $\boldsymbol{d}(k)$ in Figure 7.10), while the blocking matrix output $\boldsymbol{u}(k) = \boldsymbol{Bx}(k)$ contains interference components only. Therefore, the coefficient vector can be optimised, using standard adaptive filters in a noise cancellation setup. In this thesis, we select $\boldsymbol{g} = 1$, this is the well known minimum-variance distortionless response (MVDR) beamformer.

The spatial response, or beampattern, is defined by $10\log_{10}|\boldsymbol{w}^H(n)\boldsymbol{s}(\theta)|^2$, where

$$\boldsymbol{s}(\theta) = [1, e^{-j\theta}, e^{-j2\theta}, \dots, e^{-j(N-1)\theta}]^T \tag{7.7}$$

is the steering vector. The electrical angle θ, measured in radians, is related to the angle of incidence ϕ by $\theta = \pi \sin\phi$ ($\phi = \sin^{-1}(-0.2)$ is selected in this chapter).

Figure 7.11 shows the adapted spatial response of the classic sequential updated QR-RLS based MVDR beamformer after 200 snapshots (k counts from 0 to 199).

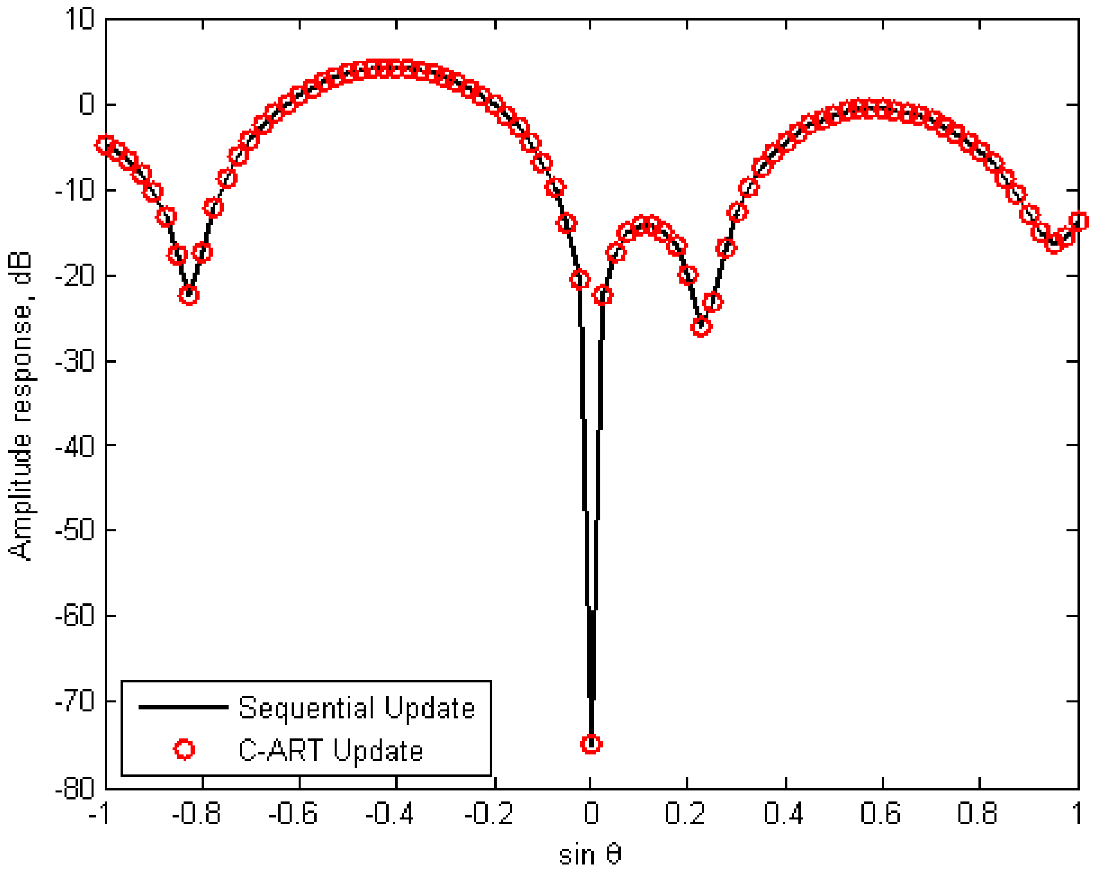


Figure: 7.11: Spatial response of the sequential updated QR-RLS based MVDR beamformer

When we replace the sequential QR update based beamformer with the proposed annihilation reordering look-ahead transformed array, the beampattern of the MVDR beamformer is also plotted. the parameters of the QR-RLS here have the forgetting factor $\lambda = 0.99$ and the initial value of $\boldsymbol{R}$ equals the identity matrix. By comparing both figures, the responses of two beamformers along the target angle -0.2π are both held at 0 dB, hence employing both QR-RLS algorithms in adaptive beamforming produces the same spatial response.

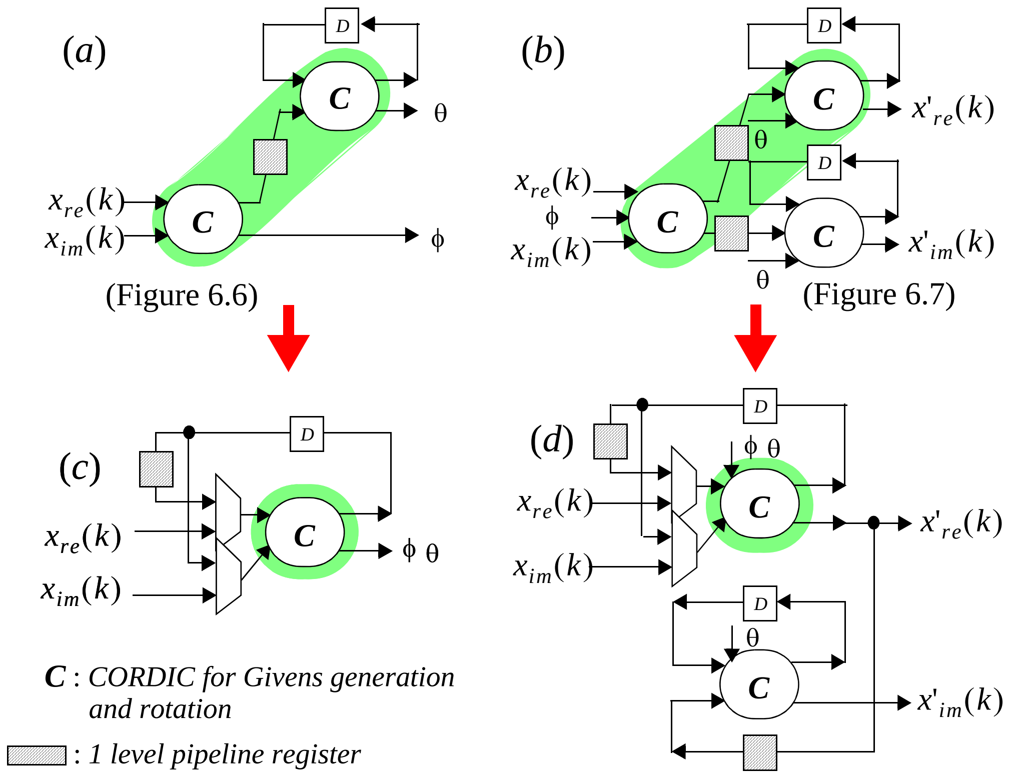


Figure: 7.12: Mapping of sequential updated BC and IC/DC

7.4 Fine-grain Level Mapping

Basing on Section 7.2, retiming an M-level pipelined complex Givens rotation based PE can speed up the QR-RLS algorithm to work at M times of its original maximum clock/sampling rate. However, for an equalization or beamforming application, the hardware ultilisation is too large to distributedly map all the BCs and ICs of QR-RLS systolic array. By incorporating the pipeline-interleaving scheme [69] and merging the BC in Figure 7.12 (*a*) and IC in Figure 7.12 (*b*), a 3-CORDIC structure is created which has similar cost to a single IC. Similarly, a 6-CORDIC architecture is resulted by pipeline-interleaving the C-ART updatecd BC in Figure 7.13 (*a*) and IC in Figure 7.14 (*a*). Hence for comparison purposes, we only look at the internal cells.

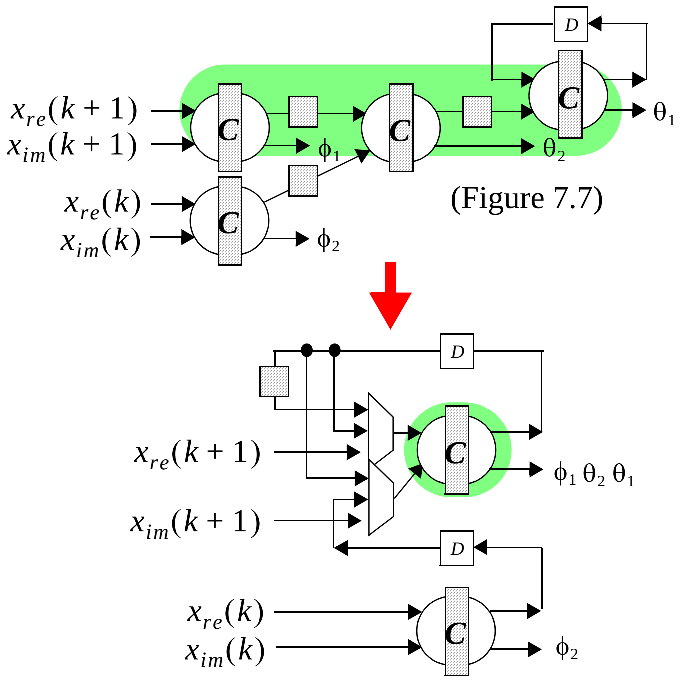


Figure: 7.13: Mapping of C-ART updated BC

Table 7.1 compares the FPGA synthesis results of the 15-iteration 18-bit precision unrolled IC of the sequential updated QR-RLS in Figure 7.12 (*b*), with that of the C-ART QR-RLS in Figure 7.14 (*a*). Xilinx ISE 10.1 is used to target a Virtex-4 xc4vsx35-10ff668 device. From the results in Table 7.1, it is clear that the C-ART architecture can improve the throughput, and almost achieves the theoretical 2 times speed.

Table 7.1: Synthesis results of a sequential and C-ART updated unrolled IC (M =2)

Type	Slice	FF	LUT	Critical Path	Max Clock Rate	Max Sampling Rate
Sequential update	1249	148	2437	38.924ns	23.691MHz	23.691MHz
C-ART update	2862	613	5374	22.968ns	43.538MHz	43.538MHz

As presented in Section 7.2, since the number of CORDIC elements required is linearly proportional to the look-ahead transformation factor M, the cost is often too large to map the fully parallel BCs and ICs of QR-RLS systolic array on hardware, due

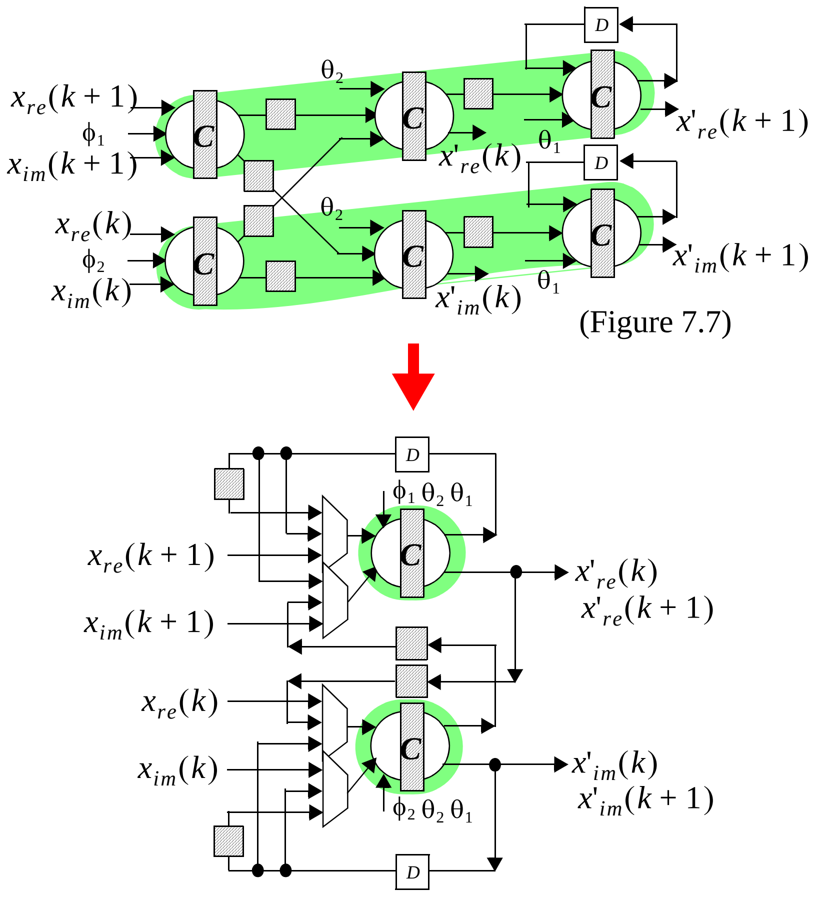


Figure: 7.14: Mapping of C-ART updated IC/DC

to the large number of CORDIC processors. Hence, the folding transformation is necessary to reduce the amount of the CORDICs inside each PE in Figure 7.7, by time-sharing one CORDIC unit.

The aforementioned 2-CORDIC sequential updated BC in Figure 7.12 (*a*) can be folded to a 1-CORDIC architecture in Figure 7.12 (*c*), the 3-CORDIC sequential updated IC in Figure 7.12 (*b*), together with the newly presented C-ART PEs in Figure 7.7, can all be further mapped to a 2-CORDIC architecture, as shown in Figure 7.12 (*d*), Figure 7.13 (*b*) and Figure 7.14 (*b*). The system throughput of sequential updated PEs has been reduced to its $1/2$, and that of the C-ART updated PEs is down to $1/3$. However by using retiming transformation, it can be verified that the folded architectures in Figure 7.13 (*b*) and Figure 7.14 (*b*) can be pipelined at 2 levels

Table 7.2: Synthesis results of a sequential and C-ART updated rolled IC ($M=2$)

Type	Slice	FF	LUT	Critical Path	Max Clock Rate	Max Sampling Rate
Sequential update	919	123	1718	53.077ns	18.841MHz	9.42MHz
C-ART update	1000	252	1822	30.329ns	32.972MHz	10.99MHz

compared to the non-pipelined rolled architecture in Figure 7.12 (*c*) and (*d*), hence the folded C-ART based PEs have 2/3 sampling rate of the unfoled ones, if the small overhead of multiplexers, flip-flops etc in the folded architecture are reasonably considered minimal. Namely, the folded $M = 2$ C-ART updated PEs can still improve the system throughput by $2/3 - 1/2 = 1/6$

From the synthesis results listed in Table 7.2, the newly proposed architecture can almost improve the throughput by $1/6$, and maintain a low hardware cost.

7.5 Conclusion

In this chapter, a novel Complex valued Annihilation-Reordering Look-ahead Transformation (C-ART) was successfully developed to increase the throughput/ sampling rate of the double angle complex rotation (DACR) based QRD-RLS systolic array, without degrading the algorithm's convergence behavior in contrast to the relaxed annihilation-reordering Look-ahead in [46]. The DACR can be realized by the CORDIC arithmetic and is suitable for cut-set pipelining to increase the throughput. Computer verification in the adaptive equalization and beamforming environments prove that the new technology could achieve the same ensemble-average learning curve and beampattern with the classic sequential QR updated RLS algorithm. Finally, the methodology of fine-grain level mapping is presented, and the FPGA implementation of a single processing element (PE) confirms the higher throughput property of the C-ART structure. And also demonstrates the insignificant change on its hardware cost point of view. Compared to the classic sequential updated QR-RLS, this architecture leads to almost twice the throughput with the Look-ahead factor of 2.

Chapter 8

Conclusions & Future Work

8.1 Conclusions

The conclusions formed from the analysis and experimental results presented are summarised below.

Both the unpipelined and pipelined barrel shifter based serial CORDIC architectures can not result in the high throughput for QR-RLS implementations. DSP48 based serial CORDIC does not produce the satisfactory speed either. The introduction of pipeline registers into the recursive loop also limits the throughput of QR-RLS array implementation due to the iteration bound increase.

By comparing the proposed coarse angle rotation mode CORDIC with two types of conventional CORDICs, it is shown that the newly proposed CORDIC has the lowest cost. One principal contribution of this research is applying a pipeline-interleaving scheme to multiplex all the PEs of QR-RLS systolic array. By using the pipeline-interleaving CORDIC to multiplex more than one data channels together, the pipelined recursive loop can be filled up efficiently.

This thesis uses the pipeline-interleaving scheme to map the extended QR-RLS array onto a 'Single PE + MAC' architecture. The resulted pipeline-interleaving CORDIC based 'processor-like' QR-RLS combines the low hardware consumption with the benefit of high speed and scalability. By comparing with other types of CORDIC based BCs and ICs, the new 'processor-like' QR-RLS PE has the best speed-area performance, and can directly produce the adaptive filtering weights, without the need of back substitution.

The design and implementation of complex valued QRD-RLS processor on a 'Single PE + MAC' architecture has been discussed. Double Angle Complex Rotation

(DACR) is used to replace the conventional Complex Givens Rotation (CGR) in the QR PEs. New quadrant mapping/correction method has been developed. The proposed novel Approximate-Scaling-Compensation algorithm plays essentially the role of increasing the sampling rate of QR-RLS processor.

The novel Complex valued Annihilation-Reordering look-ahead Transformation (C-ART) is successfully developed to increase the throughput/sampling rate of the DACR based QRD-RLS systolic array, without degrading the algorithm's convergence behavior. The DACR could be realized by the CORDIC arithmetic and is suitable for cut-set pipelining to increase the throughput. Computer verifications in the adaptive equalization and beamforming environments prove that the new technology could achieve the same ensemble-average learning curve and beampattern with the classic sequential QR updated RLS algorithm.

8.2 Future Work

Potential avenues for extending the research presented in this thesis are given next.

Future research will be directed toward the scheduling and mapping of the distributed C-ART updated QR-RLS PEs in Chapter 7.4 onto a single processor-like hardware configurations, by employing the pipeline-interleaving scheme presented in Chapter 6. It will be interesting to see how different mapping strategies lead to different hardware ultilizations and throughputs. Low power consumption Design methodology will be challenging.

In Section 7.3, the flexibility and the scalability of the QRD-RLS core make it a perfect solution for the multi-antenna communication systems. An efficient adaptive MVDR beamforming technique will be investigated, and the system level floating and fixed point simulations will be performed. The MVDR beamforming algorithm using QR-RLS filtering will be finally implemented, analysed, and successfully tested on Xilinx FPGAs.

References

[1] Berkeley Design Technology, “Choosing a DSP Processor”, 2000, http://www.bdti.com/MyBDTI/pubs/choose_2000.pdf

[2] NVIDIA Corporation, “NVIDIA CUDA Compute Unified Device Architecture”, June 2008, http://developer.download.nvidia.com/compute/cuda/2_0/docs/CudaReferenceManual_2.0.pdf

[3] Dake Liu, “Embedded DSP Processor Design, Application Specific Instruction Set Processors”, *Mogen Kaufmann*, ISBN 9780123741233, June, 2008

[4] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* , Vol. 26, No. 2, pp 203-215, Feb. 2007

[5] Khronos OpenCL Working Group, “The OpenCL Specification”, August 2008, http://www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf

[6] V. Volkov and J. Demmel, “LU, QR and Cholesky Factorizations using Vector Capabilities of GPUs”, *Tech. Report UCB/EECS-2008-49*, EECS Department, University of California, Berkeley, May 2008

[7] V. Volkov and J. Demmel, “Benchmarking GPUs to Tune Dense Linear Algebra”, *ACM/IEEE Conference on Supercomputing (SC08)*, Austin, TX, November 15-21, 2008

[8] S. Tomov, R. Nath, H. Ltaief and J. Dongarra, "Dense Linear Algebra Solvers for Multicore with GPU Accelerators," *IEEE International Parallel & Distributed Processing Symposium*, Atlanta, GA, January 15, 2010

[9] Xilinx Inc, “UG011 - PowerPC Processor Reference Guide”, Jan, 2007, http://www.xilinx.com/support/documentation/user_guides/ug011.pdf

[10] Xilinx Inc, “UG081 - MicroBlaze Processor Reference Guide, Embedded Development Kit, EDK 10.1”, 2008, http://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[11] Xilinx Inc, “UG070 - Virtex-4 FPGA User Guide”, December 2008, http://www.xilinx.com/support/documentation/user_guides/ug070.pdf

[12] Uwe Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays,* Edition 3, Springer,. 2007, ISBN 3540726128

[13] Xilinx Inc, “XAPP465 - Using Look-Up Tables as Shift Registers (SRL16) in Spartan-3 Devices”, April 2003, http://www.xilinx.com/support/documentation/application_notes/xapp465.pdf

[14] Xilinx Inc, “UG073 - XtremeDSP for Virtex-4 FPGA”, May 2008, http://www.xilinx.com/support/documentation/user_guides/ug073.pdf

[15] S Haykin. *Adaptive Filter Theory*. Prentice Hall, 3rd edition, 1991, ISBN 0133979857

[16] R Woods, J McAllister, Y Yi, G Lightbody, *FPGA-based Implementation of Signal Processing Systems*, Wiley, 1999

[17] S. Haar, D. Daecke, R. Zukunft, and T. Magesacher, “Equalizer-Based Symbol-Rate Timing Recovery for Digital Subscriber Line Systems”, *Proc. Globecom 2002*, Taipei, Taiwan, Nov. 2002

[18] J A. Apolinario JR., *QRD-RLS Adaptive Filtering*, Springer, 1st edition, Feb 2009, ISBN 0387097333

[19] G H Golub, C F. Van Loan, *Matrix Computations*, Edition: 3, Johns Hopkins University Press,

1996, ISBN 0801854148

[20] J G McWhirter, “Systolic Array for Recursive Least-squares Minimisation”, *Electronics Letters*, vol. 19, Issue: 18, pp: 729-730, Sept 1. 1983

[21] W Givens, “Computation of Plane Unitary Rotations Transforming a General Matrix to Triangular Form”, *J. Soc. Indust. Appl. Math.*, vol. 6, No. 1, 1958

[22] L. Gao and K.K. Parhi, "Hierarchical Pipelining and Folding of QRD-RLS Adaptive Filters and Its Application to Digital Beamforming," *IEEE Trans. on Circuits and Systems Part-II: Analog and Digital Signal Processing*, vol. 47, no. 12, pp. 1503-1519, Dec 2000

[23] D. Boppana, K. Dhanoa and J. Kempa, “FPGA Based Embedded Processing Architecture for the QRD-RLS Algorithm”, *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 330 - 331, April. 20-23, 2004, Napa, CA

[24] C Dick, F Harris, M Pajic and D Vuletic,“Real-Time QRD-Based Beamforming on an FPGA Platform” Fortieth Asilomar Conference on Signals, Systems and Computers, pp. 1200 - 1204, Oct. 29 2006 - Nov. 1 2006

[25] G Lightbody, R Walke, R Woods and J McCanny, “Novel Mapping of a Linear QR Architecture”, *Proc. ICASSP*, pp. 1933 - 1936, 1999

[26] B Yang, J F. Bohme, "Rotation-based RLS algorithm unified derivation, numerical properties, and parallel implementation," *IEEE Trans. on Signal Processing*, vol. 40, no. 5, pp. 1151-1167, May 1992

[27] M Harteneck, R W Stewart, J G McWhirter, I K Proudler, “Algorithmic Engineering Applied to the QR-RLS Adaptive Algorithm”, *Proceedings of 4 th International Conference on Mathematics in Signal Processing*, 1996

[28] F. Edman and V. Owall, “A Scalable Pipelined Complex Valued Matrix Inversion Architecture”, *Proceedings of ICECS’05*, Kobe, Japan, May 2005

[29] S T. Alexander and A L. Ghirnikar, “A method for recursive least squares adaptive filtering based upon an inverse QR decomposition”, *IEEE Trans. on Signal Processing*. vol. 41 no. 1, pp. 20–30, Jan 1993

[30] S J. Chern and C Y. Chang, “Adaptive Linearly Constrained Inverse QRD-RLS Beamforming Algorithm for Moving Jammers Suppression”, *IEEE Trans. on Antennas and Propagation*, vol. 50, no. 8, pp. 1138-1150, Aug 2002

[31] J. M. Cioffi, “The Fast Adaptive ROTOR’s RLS algorithm”, *IEEE Trans. On Acoust, Speech, and Signal Processing*, vol. ASSP-38, no. 4, pp. 631-653, Apr. 1990

[32] J. A. Apolinario Jr., M. G. Siqueira and P. S. R. Diniz, “On fast QR algorithms based on backward prediction errors: New results and comparisons”, First Balkan Conference on Signal Processing, Communications, Circuits and Systems, Istanbul, Turkey, June 2000

[33] I. K. Proudler, “Computationlly efficient QR decomposition approach to least squares adaptive filtering,” in *Proc. Inst. Elect. Eng.*, vol. 183, pp. 341–353, 1983

[34] F. Ling, “Givens rotation based least squares lattice and related algori thms,” *IEEE Trans. on Signal Processing*, vol. 39, no. 7, pp. 1541–1551, July 1991

[35] J. G. McWhirter, “Algorithmic Engineering in Adaptive Signal Processing”, IEE Proceedings F, Radar and Signal Processing, vol. 139, no. 3, pp. 226-232, June 1992

[36] M. Moonen and I. K. Proudler, “Generating 'Fast QR' Algorithms Using Signal Flow Graph Techniques”, *Proc. 13th Asilomar Conf. on Signals, Systems and Computers,*. Pacific Grove, CA, USA, 3-6th November 1996

[37] M. Harteneck, R. W. Stewart, J. G. McWhirter and I. K. Proudler, “Algorithmic Engineering

Applied to the QR-RLS Adaptive Algorithm", *Proceedings of 4 th International Conference on Mathematics in Signal Processing*, 1996

[38] A. A. Rontogiannis and S. Theodoridis, "Multichannel Fast QRD-LS Adaptive Filtering: New Technique and Algorithms", IEEE Trans. Signal Process. vol: 46, no: 11, pp: 2862-2876, Nov 1998

[39] A. L. L. Ramos, J. A. Apolinário Jr., "A lattice version of the multichannel FQRD algorithm based on a posteriori backward errors", *Proceedings of the 11th Internacional Conference on Telecommunications, Lecture Notes in Computer Science*, vol. 1, pp. 488-497, Fortaleza, Brazil, 2004

[40] M. Harteneck, J. G. McWhirter, I. K. Proudler and R. W. Stewart, "Algorithmically Engineered Fast Multichannel Adaptive Filter Based QR-RLS", IEE Proceedings — Vision, Image and Signal Processing, vol. 146, no. 1, pp. 7-13, Feb. 1999

[41] K. K Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*, Wiley-Interscience, 1999

[42] C H. Lin and A Y. Wu, "Soft-Threshold-Based Multilayer Decision Feedback Equalizer (STM-DFE) Algorithm and VLSI Architecture", *IEEE Trans. on Signal Processing*, vol 53, No.8, pp. 3325 - 3336, Aug 2005

[43] J T. Lai, A Y. Wu and C H. Lee, "Joint AGC-Equalization Algorithm and VLSI Architecture for Wirelined Transceiver Designs", *IEEE Trans. on VLSI Systems*, vol 15, No. 2, pp: 236-240, Feb 2007

[44] W C. Kan and G E. Sobelman, "High Speed Look-ahead LMS Detector for MIMO Systems," *Proc. IEEE Workshop on Signal Processing Systems*, pp. 56-60, 2007

[45] Y L. Chen and A Y. Wu, "Generalized Pipelined Tomlinson–Harashima Precoder Design Methodology With Build-In Arbitrary Speed-Up Factors", *IEEE Trans. on Signal Processing*, vol 58, No.4, pp. 2375-2382, Apr 2010

[46] L Gao, K K. Parhi and J Ma, "Relaxed Annihilation-Reordering Look-ahead QRD-RLS Adaptive Filters", *Journal of VLSI Signal Processing*, vol 35, No. 2, pp. 119-135, 2003

[47] J Ma, K K. Parhi and Ed F. Deprettere, "Annihilation-Reordering Look-ahead Pipelined CORDIC-Based RLS Adaptive Filters and Their Application to Adaptive Beamforming", *IEEE Transactions on Signal Processing*, vol. 48, No. 8, Aug 2000

[48] J Ma, K K. Parhi and Ed F. Deprettere, "Pipelined CORDIC Based QRD-MVDR Adaptive Beamforming", *Proc. ICASSP*, pp. 3025–3028, Seattle, WA, May 1998

[49] A S. Householder, "Unitary Triangularization of a Nonsymmetric Matrix", Journal of the ACM (JACM), vol. 5, No.4, pp.339-342, Oct. 1958

[50] J E. Volder, "The CORDIC Trigonometric Computing Technique", *IRE Trans. Electronic Computers*, vol. 3, pp. 330-334, Sept. 1959

[51] J Walther, "A Unified Algorithm for Elementary Functions", *Joint Computer Conference Proceedings*, vol 38, pp. 379-385, Spring 1971

[52] J. Valls, M. Kuhlmann, and K K. Parhi, "Efficient mapping of CORDIC algorithms on FPGA", *SiPS 2000: IEEE Workshop on Signal Processing Systems*, pages 336–345, 2000

[53] H Y. Hu, "The Quantisation Effects of the CORDIC Algorithm", *IEEE Trans. Signal Process*. 40, 1992

[54] R Andraka,"A survey of CORDIC algorithms for FPGA based computers", *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field Programmable Gate Arrays*, pp. 191-200, Feb. 22-24, 1998

[55] P. K. Meher, J Valls, T. B. Juang, K Sridharan and K Maharatna, "50 Years of CORDIC:

Algorithms, Architectures and Applications”, *IEEE Trans on Circuits and Systems - I*, volume 56, issue 9, pp. 1893-1907, 2009

[56] R Dohler, “Squared Givens Rotation”, IMA Journal of Numerical Analysis, no. 11, pp. 1-5, 1991

[57] M. Karkooti, J R. Cavallaro and C. Dick, “FPGA Implementation of Matrix Inversion Using QRD-RLS Algorithm”, *39th Asilomar Conference on Signals, Systems, and Computers*, pp. 1625-1629, Pacific Grove, CA, Oct. 28. 2005 - Nov. 1. 2005

[58] M. Renfors and Y. Neuvo, “The maximum sampling rate of digital filters under hardware speed constraints,” *IEEE Trans. on Circuits and Systems*, vol. CAS-28, pp. 196–202, Mar. 1981

[59] A G. Dempster, M D. Macleod, “Constant Integer Multiplication Using Minimum Adders”, *IEE Proceedings on Circuits, Devices and Systems*, vol.141, Iss.5, pp. 407-413, Oct 1994

[60] K N. Macpherson, R W. Stewart, “Low FPGA Area Multiplier Blocks for Full Parallel FIR Filters”, *Field-Programmable Technology (FPT), IEEE International Conference on Proceedings*, pp. 247-254, Brisbane, Australia, Dec 6-8. 2004

[61] J Ma, K K. Parhi, G J. Hekstra and Ed F. Deprettere, “Efficient Implementations of Pipelined CORDIC Based IIR Digital Filters Using Fast Orthonormal -Rotations”, *IEEE Trans. on Signal Processing*, vol. 48, No. 9, Sept 2000

[62] K. Maharatna, A Troya, S Banerjee and E Grass, “New Virtually Scaling Free Adaptive CORDIC Rotator”, *IEE Proceedings on Computers and Digital Techniques*, vol. 151, Issue 6, pp. 448-456, Nov 2004

[63] R Q. Zhang, J H. Han, A T. Erdogan and T Arslan, "Low Power CORDIC IP Core Implementation", *ICASSP*, May 2006

[64] Y H. Hu, S. Naganathan, “An Angle Recoding Method for CORDIC Algorithm Implementation”, *IEEE Transactions on Computers*, vol. 42, No. 1, Jan 1993

[65] T H. Yu, C L. Yu, K Y Jheng and A Y. Wu, “On-Line MSR-CORDIC VLSI Architecture with Applications to Cost-Efficient Rotation-based Adaptive Filtering Systems,” *Proc. IEEE Workshop on Signal Processing Systems (SiPS-2006)*, Banff, Canada, pp. 426-431, Oct. 2006

[66] C H. Lin and A Y. Wu, "Mixed-scaling-rotation CORDIC (MSR-CORDIC) Algorithm and Architecture for High-performance Vector Rotational DSP Applications," *IEEE Trans. Circuits and Systems I*, vol 52, no. 11, pp. 2385-2396, Nov. 2005

[67] J. Valls, M. Kuhlmann, and K K. Parhi, “Efficient mapping of CORDIC algorithms on FPGA”, *SiPS 2000: IEEE Workshop on Signal Processing Systems*, pages 336–345, 2000

[68] S W. Alexander, E. Pfann and R W. Stewart, “An Improved Algorithm For Assessing The Overall Quantisation Error In FPGA Based CORDIC Systems Computing A Vector Magnitude”, *Microprocessors and Microsystems Special Issue on FPGA-based Reconfigurable Computing*, Jan 2007

[69] Q. Gao, R.W. Stewart, “Coarse Angle Rotation Mode CORDIC Based Single Processing Element QR-RLS Processor”, *17th European Signal Pricessing Conference (EUSIPCO)*, Glasgow, Scotland, Aug 24 - 28, 2009

[70] G Lightbody, R Woods and R Walke, “Design of a parameterizable Silicon intellectual property core for QR-based RLS filtering”, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, Issue 4, pp. 659-678, Aug. 2003

[71] C Dick, M Trajkovic, S Denic, D Vuletic, R Rao, F Harris and K Amiri, “FPGA Implementation of a Near-ML Sphere Detector for 802.16e Broadband Wirless Systems”, *SDR'09 Technical Conference and Product Exposition*, San Jose, CA, Dec 2009

[72] K. Hooli, M. Juntti, M. J. Heikkila, P. Komulainen, M. Latvaho, and J. Lilleberg, “Chip-Level

Channel Equalization in WCDMA Downlink," *EURASIP Journal on Applied Signal Processing*, vol. 2002, no. 8, pp. 757–770, Aug. 2002

[73] L. Gao and K.K. Parhi, "Hierarchical Pipelining and Folding of QRD-RLS Adaptive Filters and Its Application to Digital Beamforming," *IEEE Trans. on Circuits and Systems Part-II: Analog and Digital Signal Processing*, vol. 47, no. 12, pp. 1503-1519, Dec 2000

[74] B Haller, J Gotze, J R Cavallaro, "Efficient implementation of rotation operations for high performance QRD-RLS", *IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 162-174, 14-16 Jul 1997

[75] A Maltsev, V Pestretsov, R Maslennikov and A Khoryaev, "Triangular Systolic Array with Reduced Latency for QR-decomposition of Complex Matrices", *IEEE International Symposium on Circuits and Systems*, pp. 4, 2006

[76] J. R. Cavallaro and A C. Elster, "A CORDIC Processor Array for the SVD of a Complex Matrix," *Elsevier Science Publishers in SVD and Signal Processing II - Algorithms, Analysis and Applications*, pp. 227-239, Amsterdam, 1991

[77] Q. Gao and R. W. Stewart, "Improved Double Angle Complex Rotation QRD-RLS", *19th ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA'11)*, pp: 79-82, Monterey, CA, Feb 21-23, 2010

[78] Xilinx XPower Estimator User Guide, *http://www.xilinx.com/support/documentation/user_guides/ug440.pdf*

[79] Altera Inc, "Comparing Altera APEX 20KE & Xilinx Virtex-E Logic Densities", *http://www.altera.com/products/devices/apex/features/apxcompdensity.html*

[80] K. M. Buckley and L. J. Griffith, "An Adaptive Generalized Sidelobe Canceller with Derivative Constrains", *IEEE Transactions on Antennas and Propagation*, vol. 34, no. 3, pp. 311-319, March 1986

[81] L. J. Griffith and C. W. Jim, "An Alternative Approach to Linearly Constrained Adaptive Beamforming", *IEEE Transactions on Antennas and Propagation*, vol. 30, no. 1, pp. 27-34, Jan 1982

Associated Publications

During the course of this PhD research, the following papers were produced :

Q. Gao and R. W. Stewart, "Coarse Angle Rotation Mode CORDIC based Single Processing Element QR-RLS Processor", 17th EUSIPCO Conference, Glasgow, Aug 24 - 28, 2009

Q. Gao and R. W. Stewart, “Improved Double Angle Complex Rotation QRD-RLS”, 19th ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA’11), pp: 79-82, Monterey, CA, Feb 21-23, 2010

Q. Gao and R. W. Stewart, “Annihilation-Reordering Look-Ahead Technique for Complex Givens Rotation based QR-RLS Filter”, Submitted to IEEE Trans on Circuits and Systems

Q. Gao and R. W. Stewart, “Pipeline-Interleaving CORDIC Complex Arithmetic for FPGA based QR-RLS Array Processing”, Submitted to IEEE Trans on Computers