DYNAMICALLY RECONFIGURABLE ARCHITECTURE FOR THIRD GENERATION MOBILE SYSTEMS

A Dissertation Presented to The Faculty of the
Fritz J. and Dolores H. Russ
College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

by
Ahmad M. Alsolaim

August, 2002

THIS DISSERTATION ENTITLED

“DYNAMICALLY RECONFIGURABLE ARCHITECTURE FOR THIRD GENERATION MOBILE SYSTEMS”

by Ahmad Alsolaim

has been approved

for the School of Electrical Engineering and Computer Science
and the Russ College of Engineering and Technology

Janusz Starzyk, Professor
School of Electrical Engineering and Computer Science

Dennis Irwin, Dean
Fritz J. and Dolores H. Russ
College of Engineering and Technology

Acknowledgement

In the Name of Allah, the Most Beneficent, the Most Merciful. All praise and thanks to Allah, lord of the universe and all that exists. Prayers and peace be upon His prophet Mohammed, the last messenger for all humankind.

First, I thank Allah for His guidance and the completion of this work. I am deeply thankful to my mother. I will never forget her patience and dedication to my brothers, my sister, and me.

I want to express my gratitude to my committee chair, Prof. Janusz Starzyk, for his personal and academic guidance. Dr. Starzyk was more than an academic advisor. He was a friend during the years I have been at Ohio University.

Thank you to my other committee members. Prof. Dennis Irwin, Dr. Jeffrey Dill, Dr. Michael Braasch, Dr. Mehemet Celenk, and Prof. Larry Snyder.

A special thank you to my wife, Thuriya, for her patience especially for the long years of waiting for the completion of this work. My daughters, Shatha and Leen, also receive my heart-felt thanks, without them life has no joy.

I must thank my brothers and sister for their love and support. A special appreciation to my oldest brother, Abdulrahman (Abu Mo'taz). He was and still is filling the place of my father. He believed in me and always expected more of me. His guidance and support in every step of my live is well appreciated.

Thank you to my friends Suliman Altwaijre, Abdulmalik Alhogail, Ahmad Alshumrani, and Bassam Alkharashi in Cleveland, OH when I earned an MS at

Case Western Reserve University. I am especially grateful to Suliman for the help and support during the hard days when I was in Cleveland.

I am particularly thankful for my long-time friend, Abdulqadir Alaqeeli. During the years I have known him, he always helped and supported me. Without him this work will never be completed. I want to thank Saleh Alotaiwe. Saleh has been my big brother. His words and actions are so valuable and appreciated. A special thanks to Mohammed Altamimi and Mazyad Almohailb for their valuable friendship. A special thanks to two friends Abdulrahman Alsebail, and Rashed Alsedran who helped me in many ways. I extend my thanks to Mingwei Ding for tireless hours of discussions about the design of the architecture.

Thanks go to Professor Manfred Glesner who is the head of the Institute of Microelectronics System Designs, Darmstadt, Germany, and to Dr. Ing. Jurgen Becker for the opportunity to join the Institute and establish my research. I am indebted to Dr. Becker for showing me how to look forward and think without limitations. I also learned how to write scholarly papers in a professional manner.

My family and I are so grateful to Hamad Al-brathin and his lovely family for being our extended family. Hamad is help and generosity is well appreciated.

I wish to express my deep gratitude to Dr. Zahiah Bin Za'roor, my academic advisor at the Saudi Cultural Mission in Washington, DC. Her support and great personality were major helps factors in the completion of this work.

Many people contributed to this work, either directly or indirectly. Thanking every one by name would take many pages. Therefore, for the people I did not mentioned in this acknowledgment, from my heart THANK YOU.

Table of Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

3GPP:	Third Generation Partner Project
AFC:	Automatic Frequency Controller
AGC:	Automatic Gain Controller
ASIC:	Application Specific Integrated Circuits
BER:	Bit Error Rate
CCM:	Custom Computing Machines
cdma2000:	Wideband spread spectrum radio interface standard developed by the TIA of the USA
CLFSR:	Configurable Linear Feedback Shift Register
CMU:	Configuration Memory Unit
CSDP:	Configurable Spreading Data Path
CSU:	Communication Switching Unit
DIO:	Dedicated I/O unit
DRAW:	Dynamically Reconfigurable Architecture for future generation Wireless mobile systems
DRPU:	Dynamically Reconfigurable Processing Unit
DS-SS:	Direct Sequence-Spread Spectrum
DSP:	Digital Signal Processor
FDD:	Frequency Division Duplex
FPGA:	Field Programmable Gate Array
GCU:	Global Communication Unit

GPS:	Global Positioning System
HDL:	Hardware Description Language
IIR:	Infinite Impulse Response
IP:	Intellectual Propriety
IS-95:	Second generation CDMA standard developed by TIA
ITU:	International Telecommunications Union
OVSF:	Orthogonal Variable Spreading Factor
PE:	Processing Element
PSCH:	Primary Synchronization Channel
QoS:	Quality of Service
QPSK:	Quadrature Phase Shift Keying
rDPU:	reconfigurable Data Processing Units
RRC:	Root Raised Cosine
RSC:	Recursive Systematic Convolutional
SF:	Spreading Factor
SNR:	Signal to Noise Ratio
SoC:	System on a Chip
SSCH:	Secondary Synchronization Channel
SWB:	Switching Box
TIA:	Telecommunications Industry Association
UMTS:	Universal Mobile Telecommunications System
UTRA:	UMTS Terrestrial Radio Access
WCDMA:	Wideband Code Division Multiple Accesses

Chapter 1

Introduction

1.1 Motivations

The evolving of broadband access techniques into the wireless domain introduces difficult and interesting challenges to the system architecture design. The system designers are faced with a challenging set of problems that stem from access mechanisms, energy conservation, a required low error rate, transmission speed characteristics of the wireless links, and mobility aspects such as small size, light weight, long battery life, and low cost. Currently, most of the hardware solutions for mobile terminal implementation are a combination of application-specific integrated circuits (ASICs) and digital signal processor (DSP) devices.

Future generations of mobile terminals will necessitate integrating reconfigurable architectures with the so-called System-on-a-Chip (SoC) solution. This requirement stems from the increasing variety and Quality of Services (QoS) that must be supported by the mobile terminal with high reliability and strict limitations for power consumption and size.

Reconfigurable architectures, e.g., those based on the Field Programmable Gate Arrays (FPGAs) are being recognized as an alternative solution for DSP processing. “*Wireless is in many cases the driver for DSP processing on reconfigurable logic*.” said Will Strauss, who is the president of Forward Concepts and one of the leading analysts on the DSP marketplace and technology. “*The processing power needed for the wireless infrastructure has been increasing, and in many instances is greater than past and most current DSPs could supply. Thus, designers have turned to FPGAs to do the on-the-fly, front-end processing*” [1]. The current method of handling complex processing problems is to use more and faster logic gates. Today's leading FPGAs have already passed the 1M logic gates count barrier with internal system clock rates of 200 MHz. The most powerful FPGA architecture at this time is Xilinx Virtex II, which provides up to 10M gates. By exploring trade-offs between design flexibility, power, and performance, functions can be migrated from ASIC or DSP to the reconfigurable architecture, eliminating a need for the ASIC or releasing the DSP for other tasks [2].

Flexibility of the mobile terminal can be defined on two levels. First at the level of systems operation, flexibility can be defined as the ability of the mobile terminal to support many modes of operation, e. g., voice, audio, video, web browsing, GPS, data transmission, etc. while using the same limited set of hardware. Second, at the communication link level, flexibility can be defined as the ability of the mobile device to operate in two or more different wireless communication standards, e.g., GSM and IS-95.

Another important requirement for a mobile terminal is its adaptability. Adaptability is defined as the ability of the mobile terminal to easily and quickly accommodate changes in the standards or the introduction of new services. Flexibility can thus be viewed as a subset of adaptability. An adaptable and flexible hardware/software system-on-a-chip target architecture for digital baseband processing will consist of a mixture of DSP and micro-controller cores, and reconfigurable hardware parts. The system-on-a-chip also includes some glue logic and ASIC parts, as shown in Figure 1-1.

1.2 Related Work

During recent years, a number of research efforts focused on the design of new reconfigurable systems for general purpose and for particular areas of application. The driving force behind such growth in the number of research activities is the potential of reconfigurable computing to greatly accelerate a wide variety of

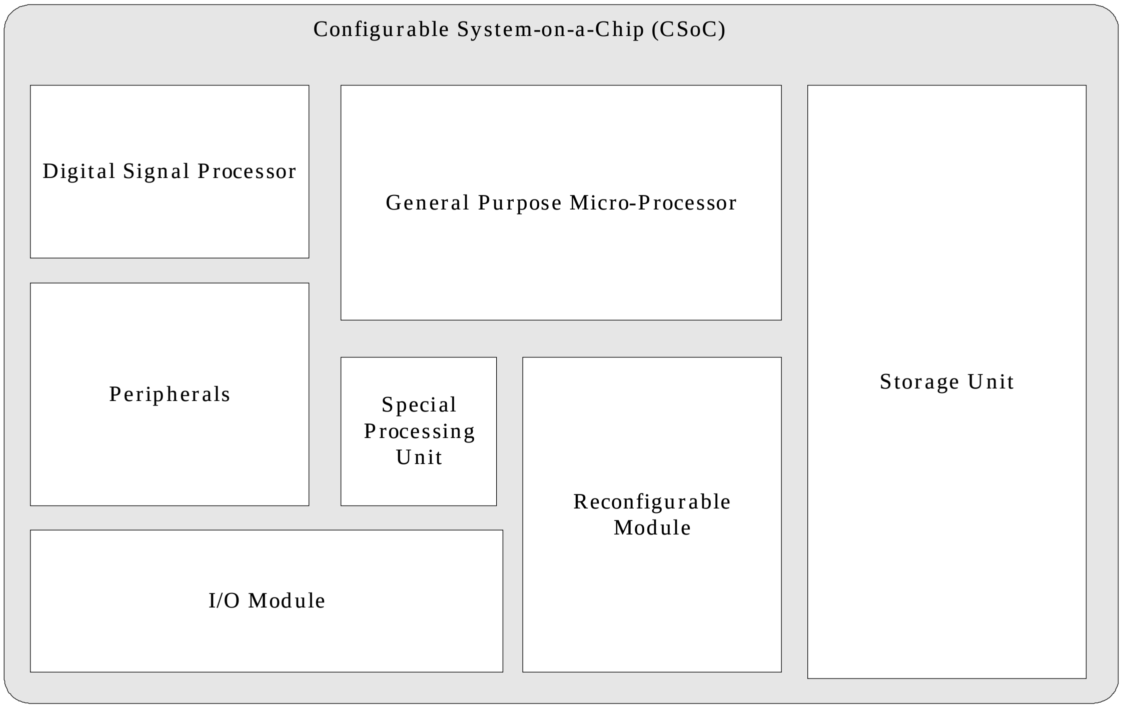


Figure 1-1: SoC-Architecture components of a Baseband single chip mobile receiver

applications. The work in this area flows in two major directions. The first hardware oriented direction is geared toward designing new hardware architectures or optimizing the current architectures. The second software oriented sub-area of research is focused on the investigation of new placement, routing, and mapping methods that tackle the dynamic reconfiguration challenges.

The work in the area of developing new reconfigurable architectures covers the research on coarse-grained and fine-grained reconfigurable architectures. Additionally, some of these architectures are created of more than one reconfigurable chips (see for example [3], [4], and [5]). These architectures (or rather systems) are called a Custom Computing Machines (CCM). In this section of the dissertation I will focus only on coarse-grained-single-chip architectures, since they are related to this work.

Fine-grain reconfigurable architectures are built around one-bit processing elements, called logic blocks. Commercial FPGAs like Xilinx 4000 FPGA family [6], and Altera Flex 8000 FPGA family [7] are good examples of fine-grain configurable architectures. Fine-grain architectures are useful for bit-level manipulation of data which can be found in data encryption and image processing applications. This type of architecture is not optimized for word-width data manipulation since a large area of the chip will be lost for routing signals. Coarse-grained architectures such as PADDI-2 [8], PipeRench [9], Morphosys [10], Grap [11], and Colt [12] are designed to implement word-width (or larger) data paths. Because the processing elements are optimized for large computations, they perform such operations efficiently in terms of time and area.

The authors of [5], [13], [14], [15], and [16] presented a comprehensive survey of available reconfigurable computing platforms in the academic and commercial. In [13] and [15] reconfigurable platforms were presented in more detail. The best references for each platform were given. However, some of the information on these architectures is either incomplete or not disclosed. The most known architectures are presented here.

Most of the architectures reported in the literature are mesh-based architectures. Figure 1-2 shows a general-block model of the mesh-based reconfigurable architectures. As their name implies, mesh-based architectures are a two dimensional arrangement of processing elements. Local connections are placed between the nearest four or eight neighbors. Additionally, vertical and horizontal communication paths are placed between the processing elements. Such an arrangement is suitable for stream-based applications or any application that requires simple

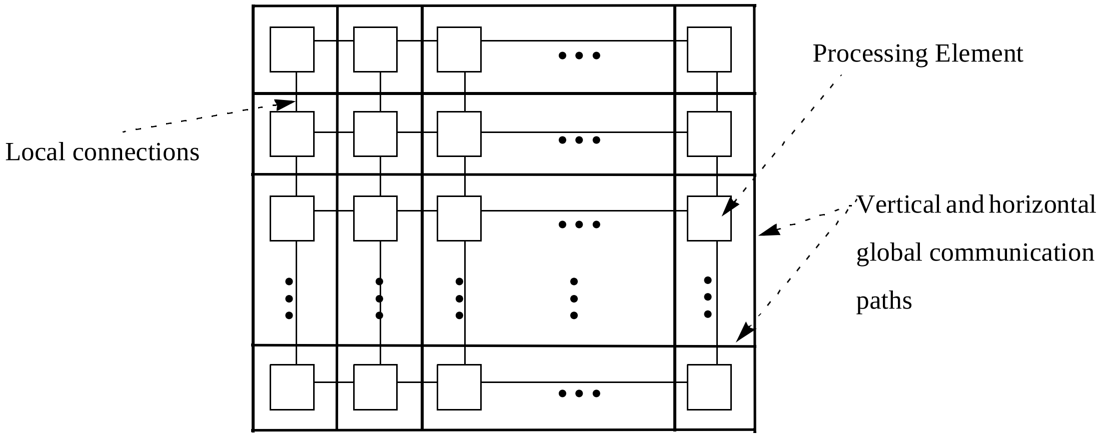


Figure 1-2: A block model of the mesh-based reconfigurable architectures with local connection to the nearest four neighbors

and regular computations. Examples of mesh-based reconfigurable architectures are KressArray [17], Colt [12], and Morphosys [10] and [18].

KressArray is a generalization of systolic array. It is a two dimensional array of 32-bit processing elements called reconfigurable Data Processing Units (rDPUs). The rDPU is designed to support all arithmetic operations of the C language. The rDPU is data triggered, i.e. rDPU performs the configured operation once the data is received [17]. Additionally, the rDPU includes a register file that can be used to store the input or output data. This register file enables the KressArray to perform deep pipelined execution of an application.

In KressArray the local connection of the rDPUs at the border of the chip are connected to I/O pins. This type of connections enables KressArry to be extendable, i.e. it can be connected to other KressArry chips to create even larger arrays of rDPUs. As shown in Figure 1-3[17] routing in KressArray is hierarchically divided into three levels. On the lowest level, a local connection to the nearest north, west, east, and south neighbor is implemented. On the second level, full length inner global buses that connect any two rDPU on the same row or same column is implemented. These buses can be segmented to form more than one bus. The third level of routing is provided by a serial system bus. The system bus is also used to transfer configuration bits to the rDPUs.

KressArray is designed for general use and was not designed for specific application. This is the reason that the rDPU is designed to be able to execute all C language operators. However, a specific application may not require all of the C language operations, resulting in a waste of area. KressArray supports partial

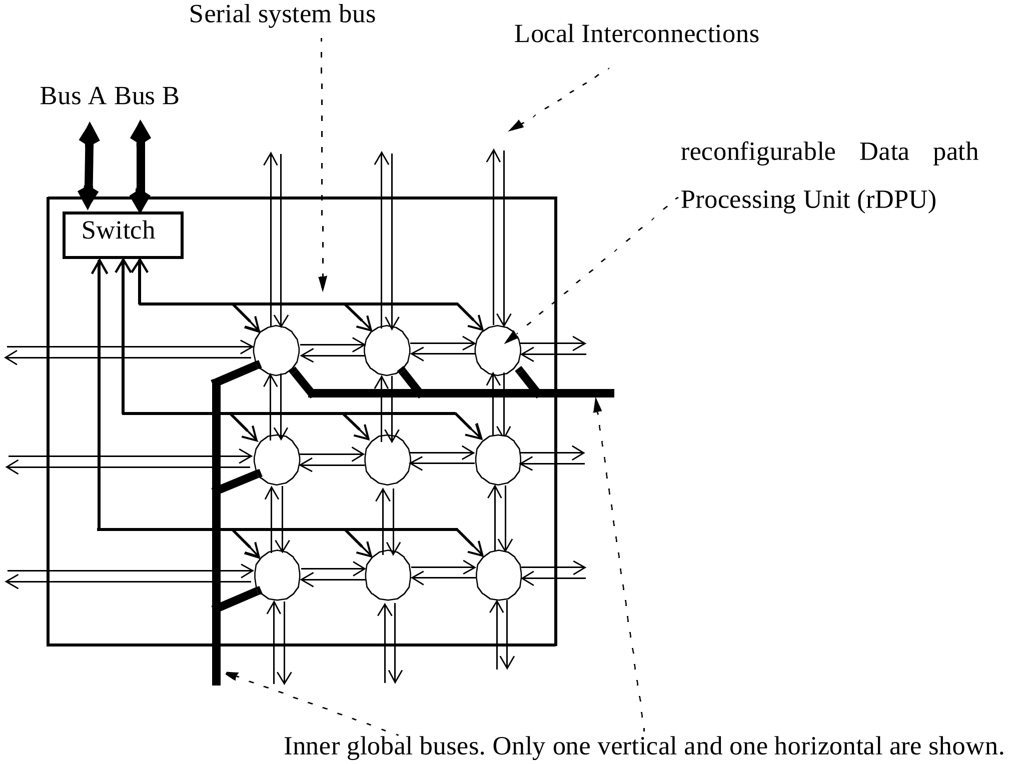

Serial system bus
Local Interconnections
Bus A Bus B
reconfigurable Data path Processing Unit (rDPU)
Switch
Inner global buses. Only one vertical and one horizontal are shown.

Figure 1-3: 3x3 segment of KressArray architectural structure

dynamic reconfiguration. However, no applications have yet been found to take advantage of such features [13].

The use of buses in the system and the use of one bus per row to transport the configuration data may be a limiting factor of the reconfiguration speed. As the number of rDPUs increases, the number of rDPUs per bus increases making the reconfiguration time even longer.

The Colt architecture is a product of work done by the Mobile and Portable Radio Research Group at Virginia Tech. The Colt integrated circuit is designed as

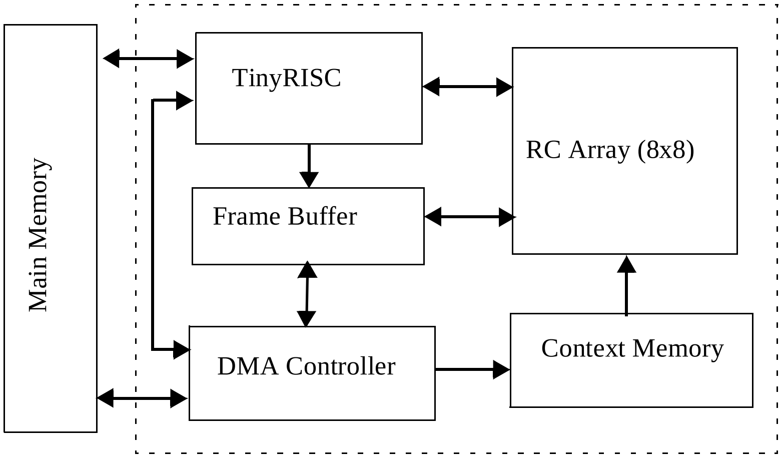


Figure 1-4: Morphosys architecture components

a prototype: the so called Wormhole Run Time Reconfiguration, and it is optimized for DSP-type operations [12].

Morphosys is a parallel reconfigurable SoC designed to speed up general purpose intensive applications. Usually these applications inherent parallelism and are highly regular. It is a combination of a RISC processor with an array of coarse-grain reconfigurable cells [10]. As shown in Figure 1-4 [10], the Morphosys architecture consists of a two dimensional array of reconfigurable cells, called a Reconfigurable Cell Array (RC Array); combined with a RISC control processor called TinyRISC; Context Memory; Frame Buffer; a DMA Controller, and a memory interface.

The TinyRISC processor is a MIPS-like processor. It has a 32-bit ALU, register file and an on-chip data cache memory. The processor is included in the system for controlling the execution of the operations and to control the main memory interface. For this purpose additional instructions which are specific to Morphosys have been added to the standard RISC instruction set.

The reconfigurable component is an array of 64 processing elements. The number of the processing elements in the array and the way they are arranged is influenced by one of the target applications: the image processing. Image processing applications tend to process the data in 8x8 segments [18]. Thus an RC array is arranged in a two dimensional array of 8x8 elements.

The reconfigurable cell of the Morphosys system is shown in Figure 1-5 [10]. The RC incorporates an ALU-multiplier unit, a shifter unit, input multiplexers

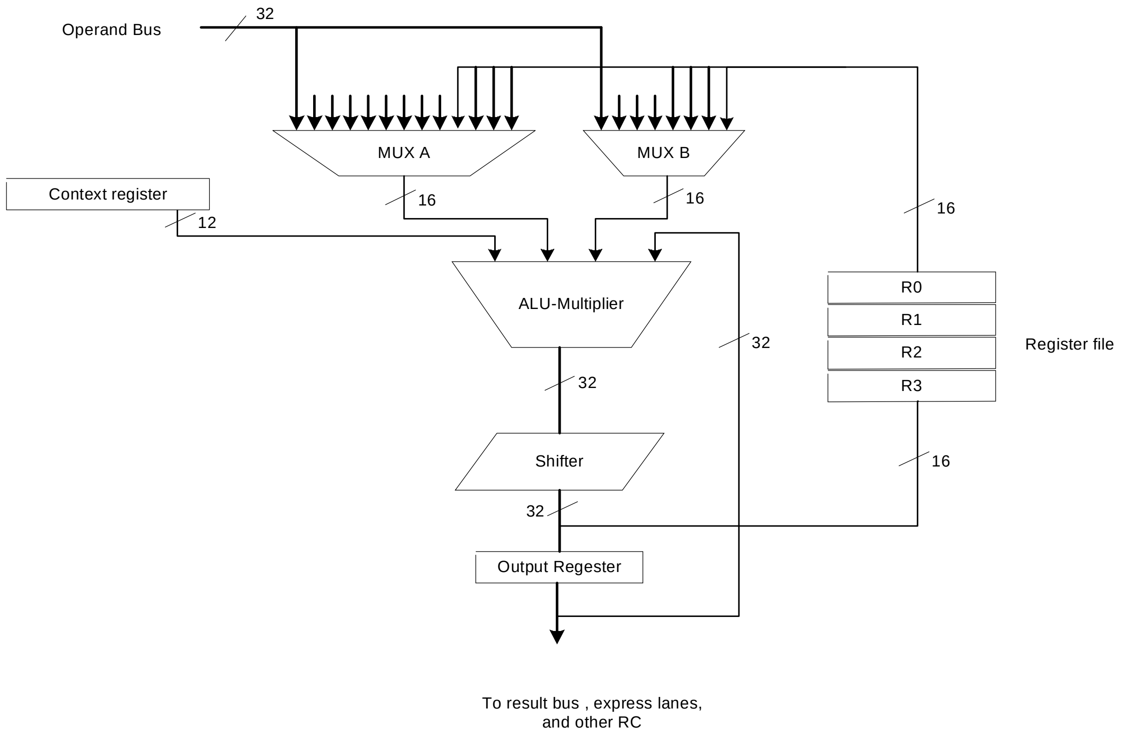


Figure 1-5: The structure of the Reconfigurable Cell (RC) of the Morphosys system

and a register file. The multiplier is included since many target applications require integer multiplication. The multiplier unit can perform integer multiplication and accumulation in one cycle. The multiplexers can select inputs from other RCs, from the operand bus, or from the internal register file.

The processing elements of the array are grouped either row-wise or column-wise for configuration. This reduces the number of configuration context of the array from sixty four to eight. In addition, this grouping of RC implies a SIMD computation model. The routing of the RC array consists of three layers, four nearest neighbor local connections, segments of the distance of two connections, and long global connections.

Morphosys is an efficient implementation system for specific applications, such as image processing or data encryption [10] and [18]. However, the SIMD computation model of the RC array limits the Morphosys from being able to efficiently implement other applications that do not inhabit block size data input or do not require regular execution styles. Additionally, the RC units of the array are very-coarse-grained (32-bit). This leads to few RC units in the array, and a waste of the resources (area) when implementing smaller data paths of 16-bit or less.

Examples of other reconfigurable computing engines that do not follow the two dimensional array topology are: The Pleiades (Ultra-Low-Power Hybrid and Configurable Computing) project at the University of California, Berkeley. This engine is designed to provide low power consumption coupled with high-performance for multimedia computing applications [19] and [20]. Pleiades is a crossbar-based architecture. The PipeRench architecture (Carnegie Mellon University) [9] is a reconfigurable fabric of processing elements for general purpose applications.

The PipeRench is specifically designed to speed up the reconfiguration process through a new technique called pipeline reconfiguration. Pipeline reconfiguration allows one stage of the pipeline path to be configured in every cycle, while concurrently executing all other stages.

Finally, the Grap architecture (University of California, Berkeley) [11] is tailored toward accelerating loops for general purpose computations.

All of the above mentioned architectures are being developed in the academia. Commercial solutions for tele-communication and wireless applications are being developed by Chameleon Systems [21], and MorphICs [22]. Chameleon Systems [23] announced the CS2000 family of multi-protocol multi-application reconfigurable platforms for tele-communication and data communication. Figure 1-6 [21] shows a block diagram of the SC2000 family. The CS2000 family incorporates a 32-bit RISC core, full memory controller, PCI controller, and a reconfigurable array. The reconfigurable array sizes come in 6, 9, and 12 tiles. The tile consists of seven 32-bit processing elements, four local memories of 128x32 bits, and two 16x24-bit multipliers. Every three tiles are grouped as a slice. Dynamic configuration is supported and can be accomplished in one cycle.

Although MorphICs announced its own version of reconfigurable chips targeting the next generation of wireless application, it never disclosed any information about the inner design of its solution.

Chameleon CS2000 can be considered as a general solution for the wireless application. However, it was not meant to be an implementation solution for the

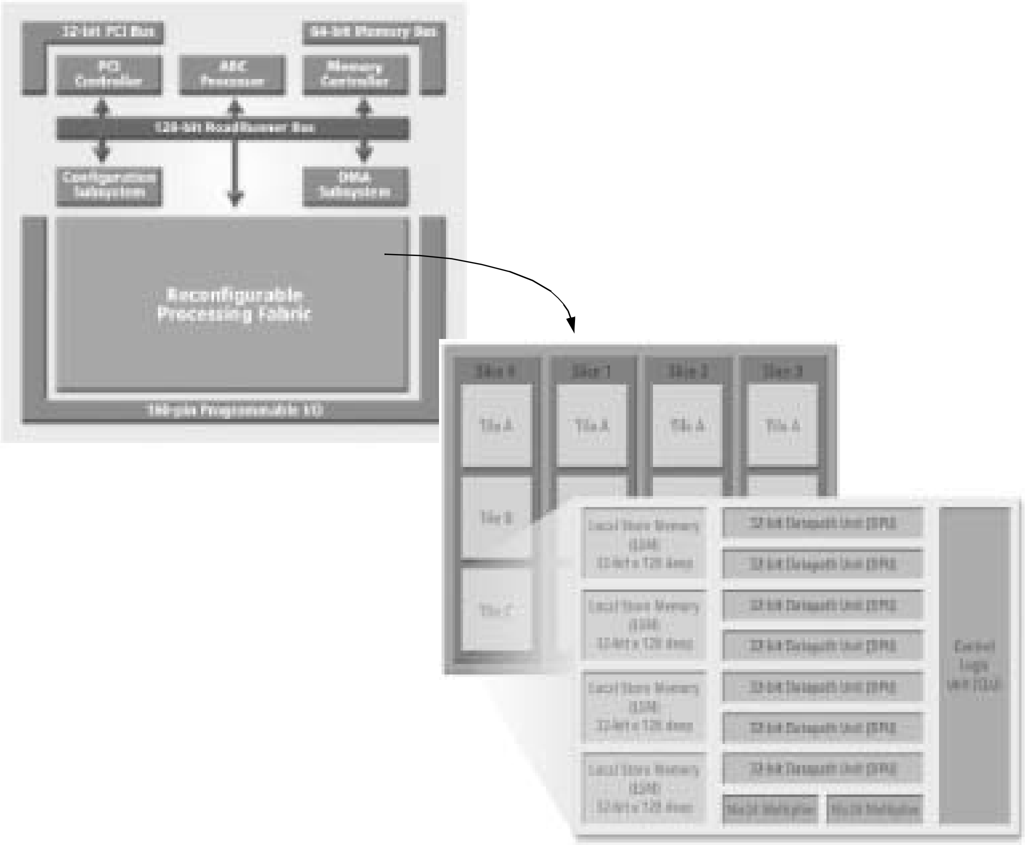

32-bit PCI Bus
64-bit Memory Bus
PCI Controller
ARC Processor
Memory Controller
128-bit RoadRunner Bus
Configuration Subsystem
DMA Subsystem
Reconfigurable Processing Fabric
160-pin Programmable I/O
Slice 0
Slice 1
Slice 2
Slice 3
Tile A
Tile A
Tile A
Tile A
Tile B
Tile C
Local Store Memory (LSM) 32-bit x 128 deep
Local Store Memory (LSM) 32-bit x 128 deep
Local Store Memory (LSM) 32-bit x 128 deep
Local Store Memory (LSM) 32-bit x 128 deep
32-bit Datapath Unit (DPU)
32-bit Datapath Unit (DPU)
32-bit Datapath Unit (DPU)
32-bit Datapath Unit (DPU)
32-bit Datapath Unit (DPU)
32-bit Datapath Unit (DPU)
32-bit Datapath Unit (DPU)
16x24 Multiplier
16x24 Multiplier
Control Logic Unit (CLU)

Figure 1-6: Chameleon CS2000 Reconfigurable Communications Processor and Reconfigurable Processing Fabric (RPF)

baseband processing of the handheld terminals. The CS2000 family's very sophisticated and in-homogenous array makes an IP-based mapping difficult.

Based upon all of the published work, no architecture was designed specifically for the wireless application. Additionally, general architecture that could perform well for all applications is very hard (if not impossible) to be designed [24]. Therefore, tailored architectures for different areas of application are a must. Presently, no formal methods or guidelines have been devised for a reconfigurable architecture design and specifically no methods exist for mobile communication.

Today mobile communication applications are of critical importance. Therefore, it is important to develop models and rules to design the reconfigurable architectures for these applications.

1.3 Project Goals and Objectives

The goal of this dissertation is to develop an array of coarse-grained Dynamically Reconfigurable Processing Units (DRPUs), which are internally connected with optimized and reconfigurable communication structures. The architecture provides efficient and fast dynamic reconfiguration possibilities (e.g., only partial and during run-time) while other parts of the reconfigurable architecture are active [25] and [26].

The choice of implementing algorithms depends upon the design of the hardware flexibility, speed, and power consumption requirements. Traditionally, algorithms that handle chip-rate signal processing (3.84M chip-per-second over sampled 4 or 8 times) are implemented on an ASIC. Whereas algorithms that handle symbol-rate signal processing (maximum of 2M samples-per-second) are implemented on a DSP. An ASIC generally runs faster and consumes less power when compared with the general purpose Digital Signal Processor DSP [27]. The developed architecture is to be able to handle both signal rates with performance and power consumption equivalent to or better than dedicated ASIC or DSP devices.

Modern communication systems operate with fixed, detailed execution formats that impose frequent deadlines. As a result, it is essential to know the execution times of various signal-processing algorithms. This is difficult with general-purpose processors, because they manipulate the flow of data and the instruction sequence to balance loading. Therefore, the reconfigurable computing architecture must provide hard timing limits for well-known execution times of intended functions [28].

The primary objective of this work is to design and simulate a new architecture for the mobile station for third generation (3G) systems. The design and simulation will be based on the principles and methods of Dynamically Reconfigurable Computing. The design of such architecture will cover broad disciplines. A low power design, hardware/software codesign, reconfigurable computing simulation, and computer programming are a few examples of these disciplines.

The design processes incorporate new design algorithms specially developed for third generation mobile systems. Such algorithms will be adaptable to channel changes, will be efficient, and will take advantage of the dynamic and parallel nature of the architecture. The final system will provide a better solution for mobile stations. The system will reflect the low cost requirement of the future systems and at the same time it will combine low power consumption with a powerful performance. Another important goal of this work is to develop the bases of a set of methods and rules for designing reconfigurable architectures for a specific area of applications.

1.4 Dissertation Outline

This dissertation is organized as follows: Chapter 2 introduces the up coming third and future generation mobile systems in general. Chapter 3 examines some of the most demanding applications at the receiver baseband processing of the mobile terminal. Chapter 4 discusses different platforms available for the targeted application and compares their pros and cons. A more detailed discussion about the dynamically reconfigurable architectures is presented in depth in this chapter. Chapter 5 introduces the Dynamically Reconfigurable Architecture for future generation Wireless mobile systems (DRAW). An abstract planar model of the architecture and a detailed description of the new features and aspects of the architecture are also presented in Chapter 5. Chapter 6 presents the design flow of the DRAW architecture. Many new concepts that emerge from the design of DRAW are also introduced and discussed. The architecture is composed of an array of processing elements dubbed DRPUs and dynamically reconfigurable communication resources. In order to demonstrate the new concepts introduced through DRAW, two applications an FIR filter and a Gold code generator were mapped into DRAW. Along with a discussion of the mapping process, simulation results are presented in Chapter 7. Chapter 8 provides a general conclusion and recommendations for future research in this area.

Chapter 2

Third and Future Generations Mobile Systems

2.1 Introduction

This chapter briefly presents the new evolving Wideband Code Division Multiple Accesses (WCDMA). Presently this is a standard of the third generation mobile systems developed by the third-generation partnership project (3GPP). 3GPP was established to harmonize and standardize many similar proposals from different parts of the world [29]. In addition, a brief look at the future wireless communication systems will be provided.

2.2 Overview of the Third Generation Systems

The first generation mobile systems in the 1970s were based on analog transmission schemes. They supported only the voice service. The second generation mobile systems in the 1980s were based on digital transmission scheme. The second generation was designed for circuit switched services. In addition to voice service, the second generation mobile systems supported low to medium packet based services. Third generation systems are designed for both circuit switched and packet based services. They will provide very high data rates that enable many new and interesting services [30], [31], [32], and [33]. Table 2-1 [34] lists the different mobile cellular radio systems, the services they offer, and their maximum data rates [34].

Table 2-1. A list of different wireless mobile generations

Mobile Generation	Services	Data rate
Old (1G)	Voice	13.3 kbps
Current systems (2G)	Voice, Text, Static Images, Data	9.6 - 41.4 Kbps
Emerging (2.5G)	Web based connection and Blue tooth	115 Kbps, Blue tooth 721 Kbps.
3G	Entertainment Education MP3 MPEG4 Games Videos	Up to 2 Mbps

The International Mobile Tele-communication 2000 (IMT-2000) is a family of systems for the third generation mobile telecommunications. This family of systems will provide wireless access any time and any where. IMT-2000 is one of the most exciting developments in mobile communication since the introduction of digital mobile systems in the early 1990s. IMT-2000 is developed by the International Telecommunication Union (ITU) [30].

The third generation systems are required to fulfill many objectives, the most important goals are high speed data services, flexibility, compatibility, and low cost. High data rate services are also known as broadband services. Current examples of an application that utilizes such services are high speed internet access and multimedia type applications. A flexible wireless mobile system is one that can easily support new services after the system has been deployed. As would be expected, once the system is full and running, new services that are presently beyond one's imagination will be demanded by the end user. A successful system must be able to accommodate these future services. Backward compatibility with second generation systems is a vital requirement for the success of the system. In addition to all the new high data rates services that are provided by the third generation system, the cost for the end user must be kept as it is today if not reduced [30], [31], and [35].

Third-generation mobile radio systems have been under intense research and discussion and will emerge around the end of 2002. The system is called the International Mobile Telecommunications-2000 in the International Telecommunications Union. Whereas in Europe, the system is called the Universal Mobile Telecommunications System (UMTS) [32]. The standardization process was long and went through many disputes, discussions, and harmonization. Toward the end of 1998 two new organizations were established: the 3rd Generation Partnership Project (3GPP) and 3GPP2. The goal of 3GPP and also 3GPP2 was to harmonize the large number of the WCDMA based proposals submitted by different bodies into one system [29], and [36]. Table 2-1 lists some of the proposed standards for the third generation mobile system to the ITU.

Both the UTRA developed by 3GPP and cdma2000 developed by 3GPP2 are based on WCDMA technology. Since the UTRA was developed earlier than the cdma2000, this dissertation was developed around the WCDMA specification published by 3GPP [29], [37], [38], [39], [40], [41], and [42]. Table 2-2 [45] lists the main parameters of WCDMA [46], and [66].

Table 2-2. Radio transmission technology proposals for IMT-2000

Proposal	Description	Source
DECT	Digital Enhanced Cordless Telecommunications	ETSI Project DECT
UWC-136	Universal Wireless Communications	USA TIA TR45.3
WIMS W-CDMA	Wireless Multimedia and Messaging Services Wideband CDMA	USA TIA TR46.1
TD-SCDMA	Time-division synchronous CDMA	China CATT
W-CDMA	Wideband CDMA	Japan ARIB
CDMA II	Asynchronous DS-CDMA	S. Korea TTA
UTRA	UMTS Terrestrial Radio Access	ETSI SMG2
NA: W-CDMA	North American: Wideband CDMA	USA T1P1-ATIS
cdma2000	Wideband CDMA (IS-95)	USA TIA TR45.5
CDMA I	Multi band synchronous DS-CDMA	S. Korea TTA

2.2.1 Characteristics of UTRA (WCDMA)

The WCDMA standard has two modes for the duplex method. A Frequency Division Duplex (FDD) and Time Division Duplex (TDD). The frequency bands allocated for UTRA are shown in Figure 2-1 [35]. In UTRA there is one paired fre-

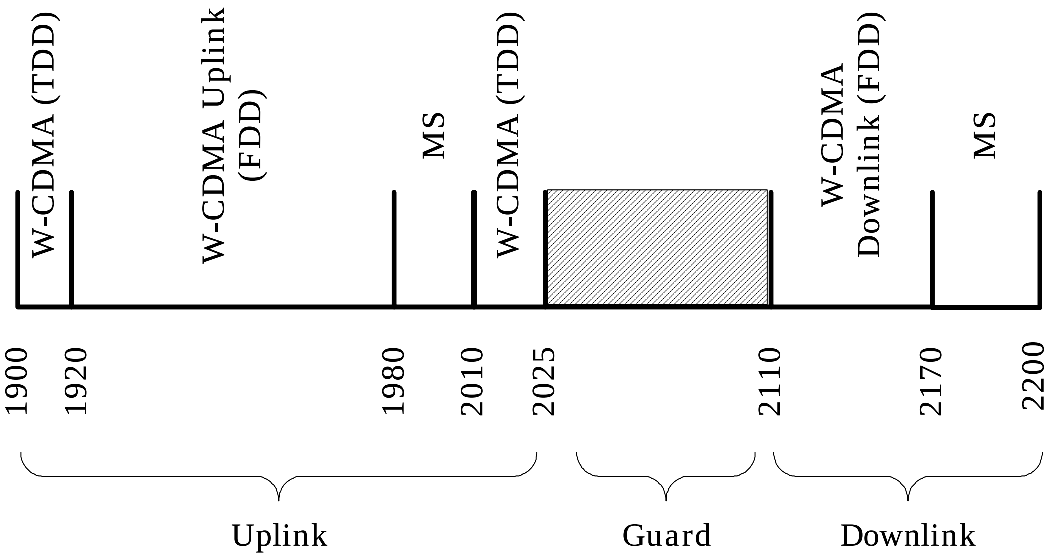


Figure 2-1: The frequency spectrum allocations for UTRA

quency band in the range 1920 –1980 MHz and 2110 –2170 MHz to be used for UTRA FDD. There are two unpaired bands from 1900 –1920 MHz and 2010 – 2025 MHz intended for the operation of UTRA TDD [31].

At the time when this work was developed, only the standard of the FDD mode developed by ITU were at an advanced stage of standardization. The TDD mode standard started later. For this reason this work assumes the FDD mode of operation for the receiver. Table 2-3 [45] lists the most important parameters of the UTRA FDD.

As can be seen in Table 2-3, the chip rate for the WCDMA standard is 3.84 Mcps [37], and [38]. Spreading consists of two operations. The first operation is the channelization operation where the spreading code is applied to every symbol in the transmitted data. Thus the bandwidth of the data signal is increased. In this

Table 2-3. Standardized Parameters of WCDMA

Channel bandwidth	5 MHz
Duplex mode	FDD and TDD
Downlink RF channel	Direct Spread
Chip rate	3.84 Mcps
Frame length	10 ms
Spreading modulation	Balanced QPSK (downlink) Dual-channel QPSK (uplink) Complex spreading circuit
Data modulation	QPSK (downlink) BPSK (uplink)
Channel coding	Convolutional and turbo codes
Coherent detection	User dedicated time multiplexed pilot (downlink and uplink), common pilot in the downlink
Channel multiplexing in downlink	Data and control channels time multiplexed
Channel multiplexing in uplink	Control and pilot channel time multiplexed I&Q multiplexing for data and control channel
Multirate	Variable spreading and multi-code
Spreading factors	4–256 (uplink), 4–512 (downlink)
Power control	Open and fast closed loop (1.6 KHz)
Spreading (downlink)	OVSF sequences for channel separation Gold sequences 218-1 for cell and user separation (truncated cycle 10 ms)
Spreading (uplink)	OVSF sequences, Gold sequence 241 for user separation (different time shifts in I and Q channel, truncated cycle 10 ms)
Handover	Soft handover Interfrequency handover

channelization operation, the number of chips per data symbol is called the Spreading Factor (SF). The second spreading operation is the scrambling operation, where a scrambling code is applied to the already spreaded signal. Both of the spreading operations are applied to the so called In-phase (I) and Quadrature-phase (Q) branches of the data signal. In the channelization operation, the Orthogonal Variable Spreading Factor (OVSF) codes are independently applied to the I and Q branches [37], and [39]. The resultant signals on the I and Q branches are then multiplied by a complex-valued scrambling code, where I and Q correspond to the real and imaginary parts respectively.

For the channel coding, the standard suggests three options of coding for different Quality-of-Services (QoS) requirements [40]. The three coding options are: Convolutional coding, Turbo coding, or no coding. The selection of one of the three options is done by the upper layers. In addition, bit interleaving is used to improve the Bit Error Rate (BER). The modulation scheme selected in 3GPP WCDMA standard is QPSK [37].

An important characteristic of the WCDMA system is that it is an asynchronous system, i. e. there is no global synchronization between base stations in the system. This means that each user can transmit independently of other users or base stations transmissions [35]. This eliminates the need for global clock similar to the IS-95 system proposed by the USA. IS-95 uses the Global Positioning System (GPS) clock as a global clock for synchronization between base stations.

Since this dissertation deals with the baseband of the mobile terminal's receiver, I will only explain the structure of the downlink. The downlink is the communication path from the base station to the mobile terminal. The physical chan-

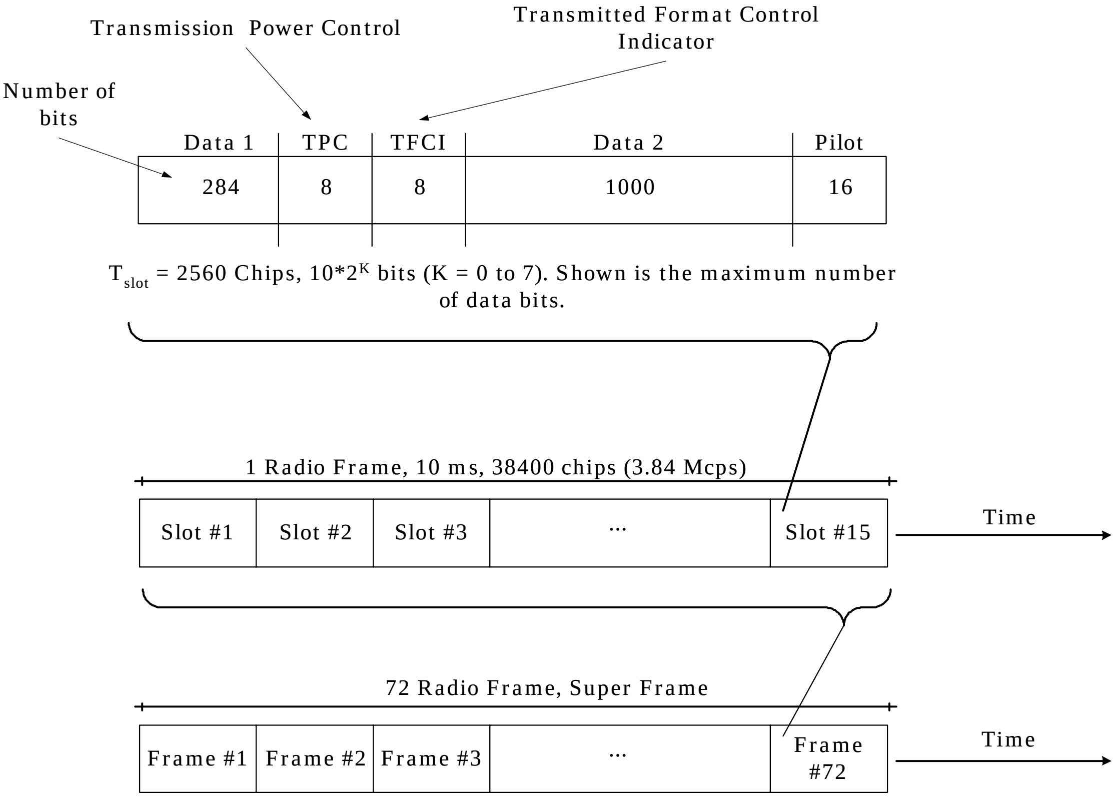

Transmission Power Control
Transmitted Format Control Indicator
Number of bits
Data 1
TPC
TFCI
Data 2
Pilot
284
8
8
1000
16
T_{slot} = 2560 Chips, 10*2^K bits (K = 0 to 7). Shown is the maximum number of data bits.
1 Radio Frame, 10 ms, 38400 chips (3.84 Mcps)
Slot #1
Slot #2
Slot #3
...
Slot #15
Time
72 Radio Frame, Super Frame
Frame #1
Frame #2
Frame #3
...
Frame #72
Time

Figure 2-2: The radio frame structure downlink DPCH of the WCDMA

nels of the WCDMA systems are structured in layers of radio frames and time slots. There is only one type of downlink dedicated physical channel, the downlink Dedicated Physical CHannel (downlink-DPCH) [39]. The structure layout of the downlink dedicated physical channel (DPCH) of the WCDMA signal can be seen in Figure 2-2 [39]. As shown in the Figure, the time line of the signal is divided into frames of 10 ms each. Each frame is then divided into 15 slots, i.e. 2560 chips/slot at the chip rate of 3.84 Mcps. In addition, every 72 frames constitute one super frame. The frame is a time multiplexed data and control bits from the Dedicated Physical Data Channel (DPDCH) and Dedicated Physical Control Channel

(DPCCH)[1]. The DATA 1 and DATA 2 are data bits that belong to DPDCH, while bits of Transmit Power Control (TPC), Transport Format Combination Indicator (TFCI), and Pilot belongs to the DPCCH. The number of bits in each field vary with the channel bit rate. The exact number of bits in each field is shown in [40]. The TPC bits are used by the base station to command the mobile transceiver to increase or decrease the transmission power. TFCI bits are the indicators of slot format.

The bit count shown in Figure 2-2 is the maximum possible number of data bits that can be transmitted in one slot. In a frame $15\times10\times2^k$ bits can be transmitted in every slot, where k is an integer in the range from 0 to 7. The parameter k is related to the Spreading Factor (SF):

$$SF = \frac{512}{2^k} \qquad \textbf{2-1.}$$

Thus the spreading factor SF may range from 512 down to 4 see [41], and [42].

2.3 Characteristics of cdma2000 system

The cdma2000 is a wideband spread spectrum radio interface standard developed by the Telecommunications Industry Association (TIA) of the USA. The cdma2000 meets all the requirements set by the ITU for the IMT-2000. The second generation CDMA standard developed by the TIA is called IS-95. The latest version of this standard is IS-95B. Since IS-95 based systems are deployed mainly in

[1] Abbreviations used here are the standard abbreviations used by 3GPP in the 3G-TS technical documents.

USA and in some countries around the world, the cdma2000 system is designed to be back compatible with IS-95. The backward compatibility feature of the cdma2000 (with the second generation system IS-95) reduces the cost of deploying cdma2000 by reusing the infrastructure of the IS-95. This will also insure a smooth and successful migration from 2G to 3G system [43] and [44].

The cdma2000 radio access is based on narrowband DS-CDMA with a chip rate of 3.6864 Mcps. This chip rate is three times the chip rate of the IS-95 system which is 1.2288 Mcps. The cdma2000 standard also supports higher data rates of N*1.2288 Mcps, where N is 1,3,6,9, and 12. Similar to UTRA, cdma2000 also supports FDD and TDD mode of operations.[35]

Table 2-4 lists some of the main parameters of the cdma2000 standard. The most noticeable departure from UTRA standard is the base station synchronization and the frame length. In cdma2000, the base station uses the GPS time as a reference timing for the shift in the cell scrambling code. This feature simplifies the cell search since only one code with different shifts are used to identify all cells in the system. The frame length is two times the frame length of the UTRA/ WCDMA system.

2.4 Future Generation Mobile Systems

The introduction of the third generation mobile system next year will revolutionize the world as we know it. The impact of the system on our lives is every thing but predictable. Despite this fact, forth and fifth generations are already

Table 2-4. Standardized Parameters of cdma2000

Channel bandwidth	5 MHz
Base station synchronization	Asynchronous, GPS time reference
Duplex mode	FDD and TDD
Downlink RF channel	Multicarrier Spreading and Direct Spread
Chip rate	3.6864 Mcps
Frame length	20 ms
Modulation	QPSK (downlink), and Hybrid Phase Shift Keying HPSK (uplink)
Channel coding	Convolutional and turbo codes
Coherent detection	Pilot time multiplexed PC (uplink), Common continues pilot channel and auxiliary pilot (downlink)
Multirate	Variable spreading and multi-code
Spreading factors	4–256 (uplink)
Spreading codes	Walsh code Pseudo noise code
Power control	Open and closed loop
Handover	Soft handover Interfrequency handover

being discussed. Figure 2-3 [34] outlines some of the current, emerging, and future mobile systems along with their supported data rates [34].

The term fourth generation is used to include several systems not only cellular systems, as well as many wireless communication systems [47], and [34]. It is well expected that the future systems will provide very high data rates in the tens and even hundreds of MB/S. It is even expected that these extremely high data rates will be provided with full mobility support. However, it is very hard to envision these systems beyond these two expectations. It is very difficult to implement such systems with such high data rate and mobility [34].

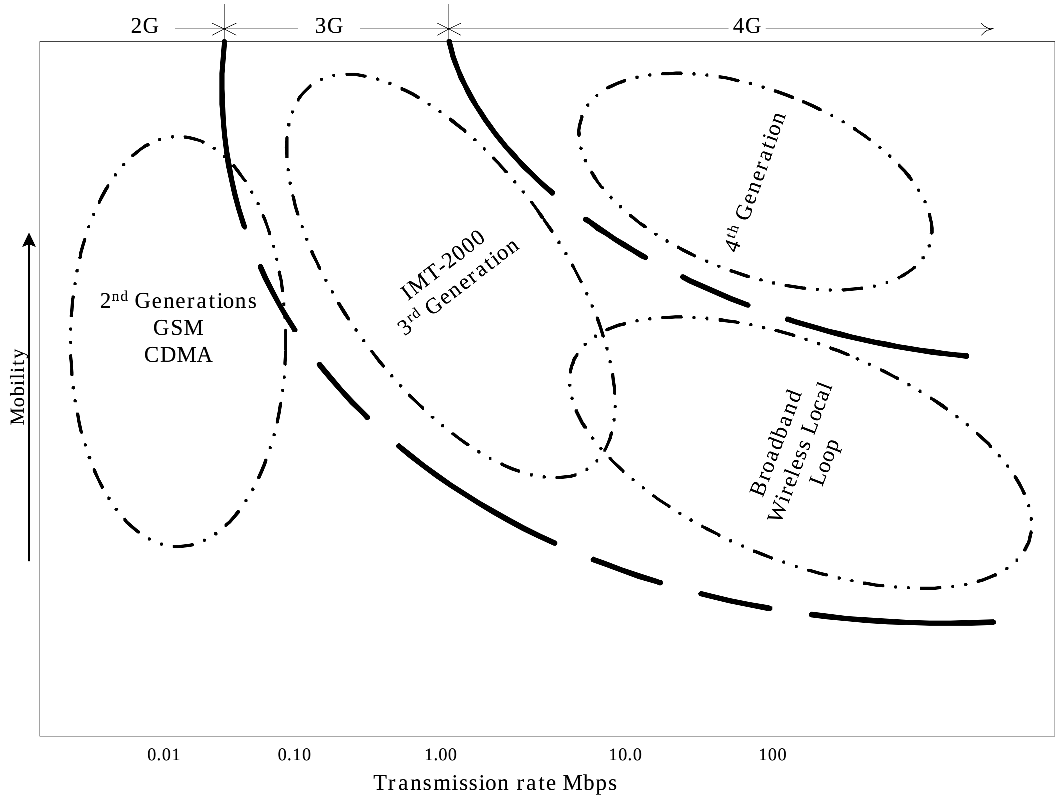


Figure 2-3: Current and future generations of wireless mobile systems vs. their transmission rates

Chapter 3

WCDMA Baseband Signal Processing

3.1 Introduction

This chapter examines the most demanding functions in the 3G baseband processing unit. The computation requirements which are the basis for the design of the reconfigurable architecture are determined. To design a good reconfigurable architecture for a specific application, some understanding of the most demanding functions of the baseband receiver are required. The target application must be well understood in terms of its:

- Performance requirements
- Routing behavior
- Input/Output requirements
- Memory requirements

The performance requirements of the target application are the first general characteristics that must be known and understood for the designer prior to designing the architecture. This requires determining rough answers to the following questions: 1) Is this a real time application or not? 2) What is the required execution time? 3) What are the required operations? 4) What are the data path width requirements? 5) What is the required throughput? 6) Is this a regular or irregular execution flow?

For routing we would like to know how different blocks of the applications are connected, i.e. is the algorithm locally or globally connected? How frequently the target application do use the routing channels? We also need to know the appropriate path-width for the routing channels of this application.

I/O requirements are usually the bottleneck of most applications. Therefore, it is very important to know in detail the required number of inputs and outputs. Simultaneously, we also need to know what are the surrounding interfaces that need to be addressed.

As important as the performance requirement is, the memory requirements also play an important role in directing the final design of the architecture. The knowledge necessary regarding memory requirements includes the type of memory that is required, and is it a one-port, dual-ports, or FIFO type of storage system? We also need to ascertain how large the required memory must be.

3.2 WCDMA Baseband Mobile Receiver

The work in this dissertation considers the Frequency Division Duplex (FDD) of the 3GPP WCDMA as a standard. In this section, I will examine the mobile baseband processing unit of the receiver. The reason behind selecting the

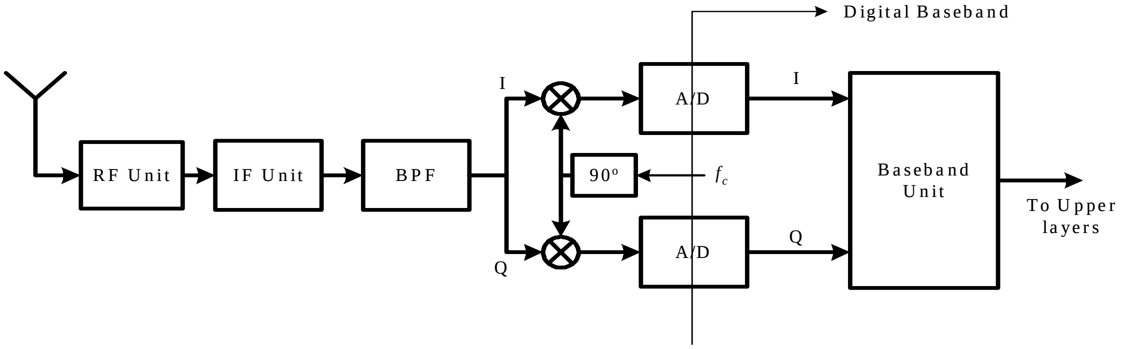


Figure 3-1: A block diagram of 3G receiver

receiver's baseband processing unit to perform the analysis rather than selecting the transmitter, is that the receiver is usually more complex and requires more processing power than the transmitter baseband unit. This is due to the fact that the receiver deals with signals that contain high levels of noise because of the nature of the channel. The mobile channel is a dynamic one, this behavior gets aggravated for communication at higher data rates and worsens when communicating with a moving mobile terminal at higher speeds. [48], [49], and [50]

Figure 3-1 outlines the block diagram of the complete receiver [51]. As can be seen in the figure, after the signal has been received by the radio frequency unit with a frequency in the range of 2110-2170 MHz, the signal is then down converted to an Intermediate Frequency (IF) level of 270 MHz. The desired 5MHz channel is filtered by an IF band-pass filter. The IF signal is fed to the demodulator circuit where it is mixed with the fixed local oscillator frequency to produce the zero IF baseband in-phase and quadrature (I and Q) signals, these are then fed to the

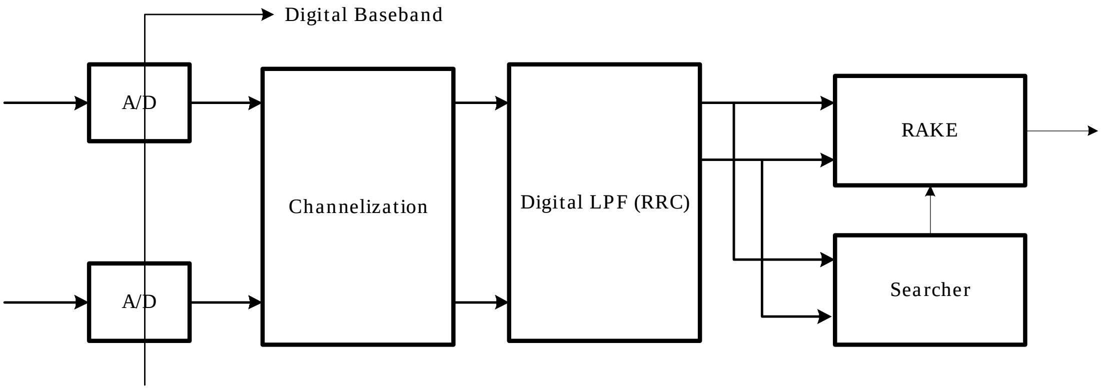


Figure 3-2: The baseband receiver front-end

analog-to-digital converter. The output of the A/D converter is then fed to the baseband unit for processing [30], and [33].

Figure 3-2 shows the block model of the receiver’s baseband processing unit. The Figure shows more detail of the inner signal path in the front end of the baseband receiver. The analog signal is converted into a digital signal using wideband analogue to digital converter typically running at 8 times the chip rate and producing 8 bit resolution [35]. The signal is then filtered using a Root Raised Cosine (RRC) filter with a roll-off factor of 0.22. The purpose of the root raised cosine filter is to reduce the inter-symbol interference. Subsequently the signal is fed to the RAKE receiver and to the searcher.

Figure 3-3, shows a block diagram of the additional processing steps at the baseband of the WCDMA receiver. As shown in the figure, the incoming chips from the A/D converter are introduced to the RAKE receiver and to the searcher. The

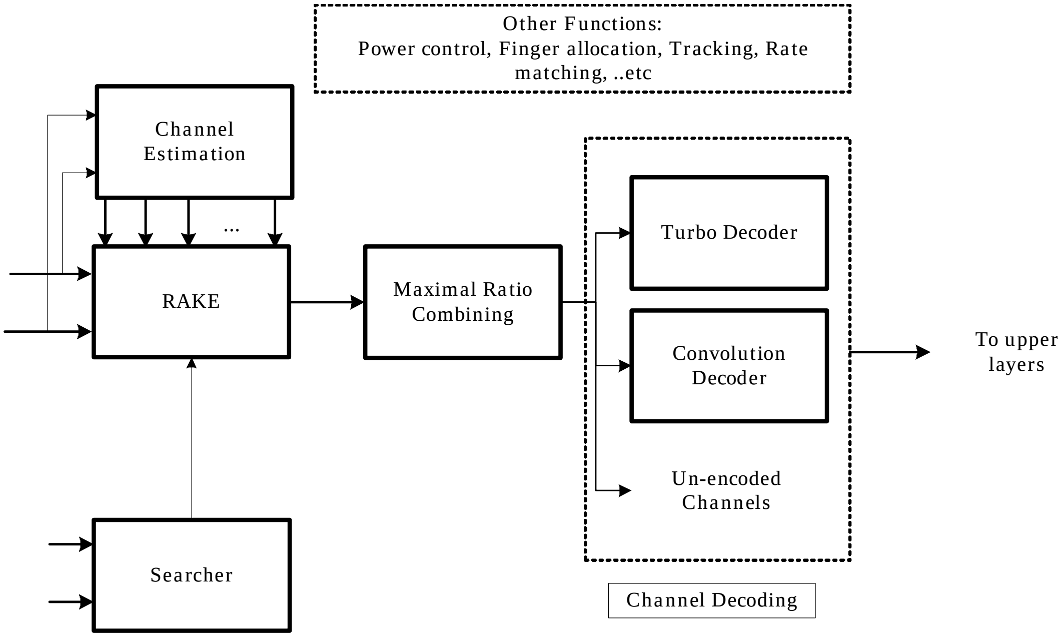


Figure 3-3: Complete block diagram of the receiver front end

searcher provides an estimation of the multipaths delays which are used by the RAKE receiver to resolve different paths signals. The maximal ratio combiner then weights the signals and sums them together [52]. In addition, the searcher provides signals for the Automatic Gain Controller (AGC), Automatic Frequency Controller (AFC), and Power Control Loop (PCL) (not shown in figure). Depending upon which signal quality is chosen, the signal is then routed to a Turbo coder, Convolutional decoder, or pass-over the decoding unit completely as in the case of un-coded channels. The selection of the type of decoding depends upon the quality of service required. Additional processing is then performed by the channel processing unit. Such additional operations may include viterbi decoding, de-interleaving, reed Solomon decoding etc. [40].

3.3 Baseband Signal Processing Requirements

Baseband signal processing in a system based on WCDMA is so complex due to the fact that a wide variety of services are offered. Services such as high-quality voice and high-quality videos are provided through high data-rate wireless channels [53]. At the same time the hand-held mobile terminal is expected to be similar to the 2nd generation mobile terminals in terms of cost, power consumption and size. In addition, the mobile terminal solution must accommodate the changes in the standard as the standard solidifies, and also accommodate other 3rd generation systems such as cdma2000. The mobile terminal solution must also be flexible and have the capacity to implement any new services envisioned in the future many of which are not fully yet understood [51]. When 3G systems come to market, first and second generation systems will be still widely used. Thus 3G mobile terminals must have a backward compatibility with previous wireless mobile systems.

Currently, there are two primary hardware realizations for the baseband processing in the 3G mobile terminal, ASIC, and Digital Signal Processor. A comparison of the two solutions' strengths and weaknesses will be fully explored.

According to J. Rabay [64] high signal processing demands of the baseband processing unit in the 3G mobile terminal are mostly concentrated in five sub-functions of the baseband. As seen in Table 3-1 the five functions are:

- Digital FIR filtering
- Searchers (frame, slot, delay path, etc.)
- RAKE reception

- Maximal Ratio Combining
- Turbo Decoding

Table 3-1. Estimation of signal processing load for 3G baseband @ 384 kbps

Function	Number of Million Instructions per Second (MIPS) @ 384 Kbps
Digital Filters (RRC, Channelization)	~3600 MIPS
Searcher (Frame, slot, delay path, etc.)	~1500
RAKE Receiver	~650
Turbo coding	~52
Maximal Ratio Combiner (MRC)	~24
Channel estimation	~12
AGC, AFC	~10
De-interleaving, rate matching	~14
TOTAL	~5860

By speeding up the execution of these functions, an overall speed-up of the baseband processing will be obtained. Each one of these functions will be discussed in detail.

The implementation of the baseband unit on a reconfigurable hardware can benefit from the fact that each part of the baseband unit performs a set of fixed operations on a block of data. The block of data can be one frame data or one slot data. As a result, if the hardware can perform all the required operations before a new block of data arrives then, one can configure the different parts of the baseband unit on the reconfigurable hardware sequentially. In addition, if the hardware is capable of executing all the operations fast enough, then it can be turned off to save power until the new block of data arrives [50], and [55].

The selection of the size of the data block affects the final performance in different ways. For example, setting the block size to a small size like the data in one Slot (0.66 ms), will reduce the area for storing the incoming data and reduce the area of the executing hardware of each part of the baseband unit.

The reduction in the required area comes from the assumption that the processing of data is performed in parallel. This means a smaller block of data will require smaller hardware for execution. But selecting a smaller block of data may reduce the performance of some algorithms. It may even result in a data block that is smaller than the amount of data required by the algorithm to operate correctly.

3.3.1 Digital Filters

One of the tasks for the implementation of the 3rd generation wireless mobile baseband unit is the design of the digital filters for the receiver. In order to transmit larger amounts of data through a limited frequency bandwidth, the receiver filter must follow very strict specifications. These specifications result in a larger number of taps and increase the storage requirements for the coefficients. This increases the required processing power of the filter [33], and [56].

For a 3G standard, a Root Raised Cosine (RRC) filter is proposed for pulse shaping with a roll-off-factor $\alpha = 0.22$ [42]. The root raised cosine filter is implemented to reduce the inter-symbol interference. Typically, Finite Impulse Response (FIR) filters are used in baseband signal processing since they are stable. However, for a given frequency response, FIR filters are a higher order filters (compared to Infinite Impulse Response (IIR) filters). Hence they are more computa-

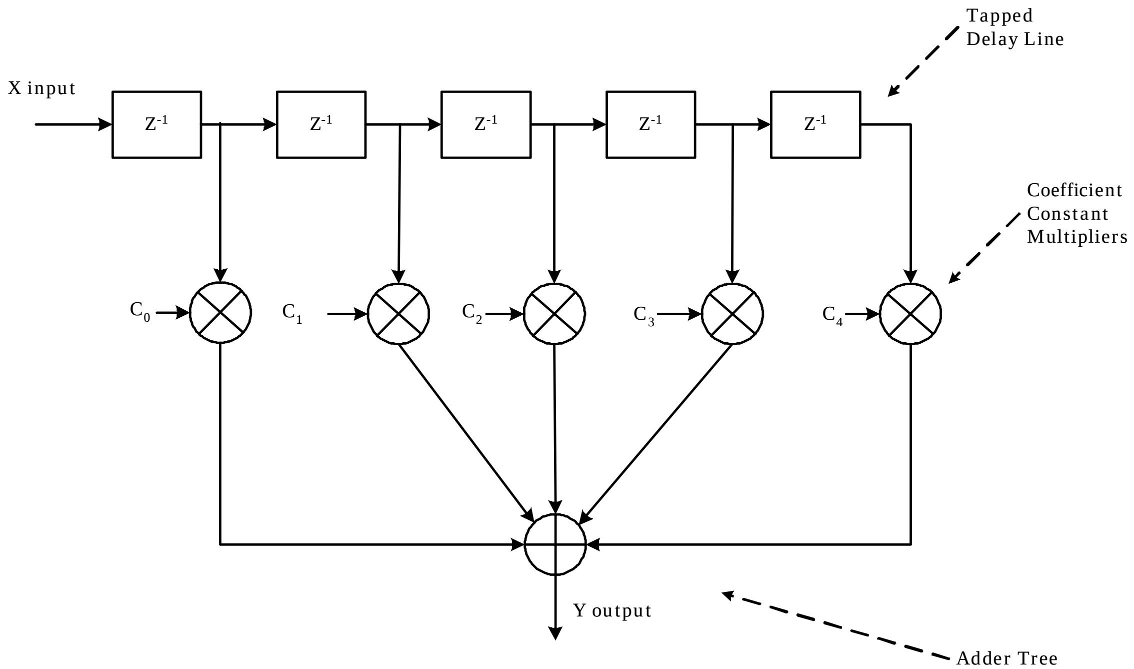


Figure 3-4: A tapped delay line model of an FIR filter

tionally expensive. The structure of a FIR filter is a weighted-tapped delay line as shown in Figure 3-4.

The Filter design process involves calculating the coefficients that match the frequency response required for the system. The values of the coefficients determine the response of the filter. By changing the values and/or the number of the coefficients we can change which signal frequency passes through the filter. As can be observed from the filter model shown in Figure 3-4, the hardware implementation of the filter involves an *N* number of Multiply and Accumulate (MAC) operations [51], where *N* is the number of filter taps.

The traditional hardware implementation of the FIR filters is a difficult one. The designer is forced to compromise between the flexibility of the implementation and the performance. The reason for the compromise is that if the filter is to be implemented on a Digital Signal Processor (DSP), the performance will be compromised since DSP usually have a limited number of MAC units. Implementing the filter on Application Specific Integrated Circuit (ASIC) will almost eliminate the flexibility unless a large area is used by the design.

The top of the line DSP has four MAC units [48]. This will require the DSP to either run at very high clock rates, which is not desirable in a mobile terminal, or it will take many clock cycles to compute each output value. However, an ASIC could be used to implement the filter. This solution would provide the highest performance and the lowest power consumption, but it would heavily compromise the flexibility. Since the hardware implementation of the filter is frozen in silicon, it is impossible to change the width of the data path or the length of the filter.

It is not feasible to design an ASIC solution with some changing variables in the FIR filter. This design would result in a large silicon area, which would be underutilized most of the time. Cost considerations, long design time, and the effectiveness of the final solution deem this unacceptable.

The implementation of digital filters on a reconfigurable hardware is a straightforward affair. The implementation is a matter of allocating a MAC unit to each tap of the filter. However, in a 3G receiver the matter is complicated by the tight bandwidth (5MHz), high dynamic range, and the steep requirement of the filter roll-off factor α. When a filter with such requirement is implemented in digital, it leads to a large filter architecture. Typically the length of the filter must be truncated in order to minimize complexity. The use of windowing alleviates the effects on the spectral shape.

If implemented in a reconfigurable computing hardware, the filter can be implemented in more than one shape. For example, if a sufficient hardware is available, then the filter can be implemented without truncating some taps, or else the filter can be implemented with fewer taps.

3.3.2 Searcher

The fundamental concept behind the Direct Sequence-Spread Spectrum (DS-SS) communication system is that channels are broadcasted on the same frequency using orthogonal spreading codes [57]. Due to the orthogonal nature of these codes, when a pattern is received and correlated with a reference code it will result in a value of 0 for all of the other signals that do not match the code. For the desired transmitted signal, the result will be non-zero, where the sign of the correlation value will indicate wether the transmitted bit is 0 or 1.

WCDMA standard [39], and [42] mandates the use of two levels of spreading. The first is the channelization spreading which uses Orthogonal Variable Spreading Factor (OVSF) codes to preserve the orthogonality between the users. However, implementing the orthogonal spreading codes is not sufficient. A long run of ones can hinder the clock recovery and transmitted power level. Also, if an adjacent cell uses the same code for spreading, then this will result in the wrong data being recovered. For these reasons the standard adds a second level of spreading. A pseudo-random scrambling code is used to scramble the channels that are transmitted from one cell. By using different scrambling codes for different cells these problems are avoided. Additionally, the standard states the use of spreading codes for cell search. For this purpose, 512 different downlink spreading codes of a length of 38400 chips are identified. It would be very complex for the mobile terminal to search all 512 codes and all of their different shifts. Therefore, the 3GPP standard proposes the use of unscrambled synchronization channels (SCH) in the downlink. Thus, the mobile terminal can use the SCHs to find the Broadcast Control Channel (BCCH) of the cell [30].

The North American 3G system proposal called Cdma2000 uses the Global Positioning System (GPS) time mark to synchronize all cells. In contrast to cdma2000, the WCDMA system is an asynchronous system, i.e. the base stations are not synchronized in time. This adds more emphasis on the searchers.Thus the synchronization is done by searching the incoming signal for the beginning of a slot and the beginning of a frame. For synchronization, the SCH has two sub-channels; Primary Synchronization Channel (PSCH) and Secondary Synchronization Channel (SSCH). Figure 3-5 shows the structure of the synchronization channel SCH [30][33].

As shown in Figure 3-5, the PSCH is a 256 chip long symbol at the beginning of the slot, it is repeated continuously for every slot and it is identical for every cell. Finding one of these 256 codes corresponds to finding the beginning of one of the slots. The SSCH is different from the PSCH. It uses 512 different scrambling codes.

Figure 3-6 shows these codes. The codes are divided into 32 groups and each group contains 16 scrambling codes. The SSCH transmits a word which points to one of the 32 scrambling groups. The word consists of 16 symbols, one for each slot.

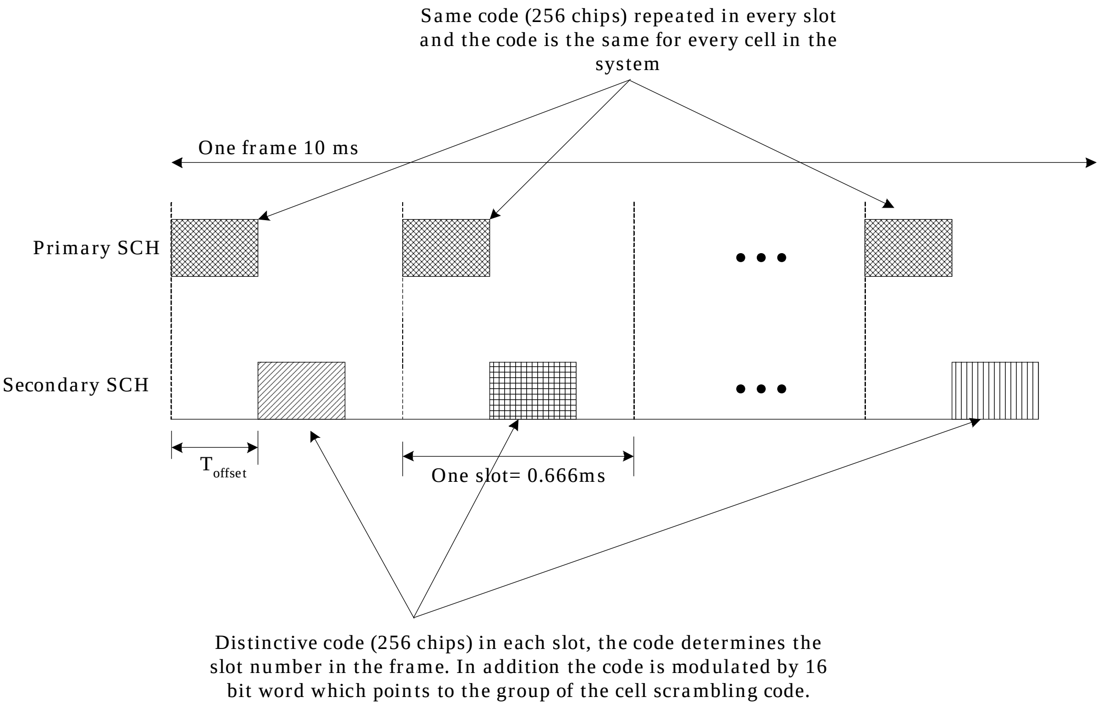

Same code (256 chips) repeated in every slot and the code is the same for every cell in the system
One frame 10 ms
Primary SCH
Secondary SCH
T_{offset}
One slot= 0.666ms
Distinctive code (256 chips) in each slot, the code determines the slot number in the frame. In addition the code is modulated by 16 bit word which points to the group of the cell scrambling code.

Figure 3-5: Structure of the synchronization channel (SCH)

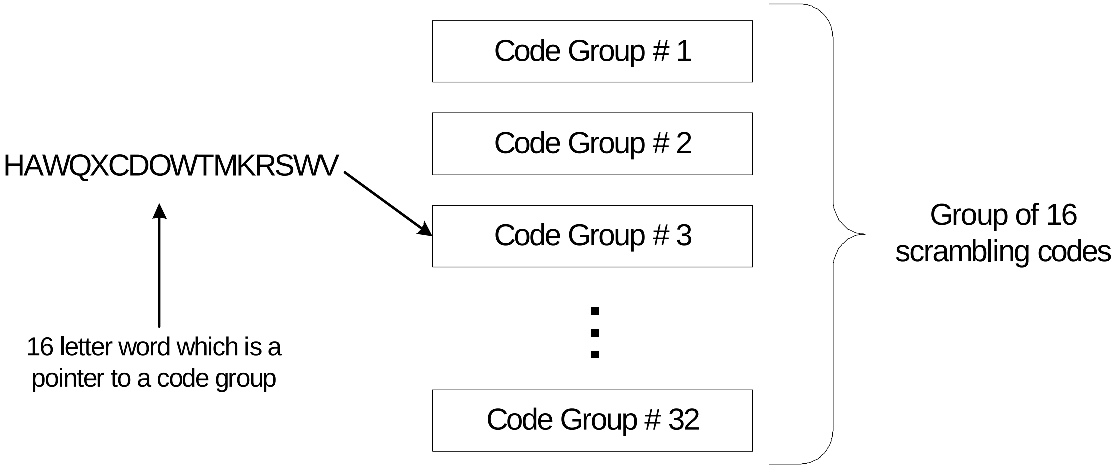

Code Group # 1
Code Group # 2
HAWQXCDOWTMKRSWV
Code Group # 3
Group of 16 scrambling codes
16 letter word which is a pointer to a code group
Code Group # 32

Figure 3-6: The secondary synchronization groups of codes

The 16 symbol words are distinct under symbol-wise cyclic shift. This allows the mobile terminal to find the beginning of a frame through locating the beginning of the word. Searcher is also used for the RAKE receiver to provide channel estimation [30], and [33].

As an example of the synchronization process, the Matlab code simulating the synchronization channel is shown in Figure 3-7. The PN code is generated by the down link scrambling code generator “dl_sc_code” (see Appendix A). The PN code is then inserted in a one slot long (2560 chip long) random data. The location of the beginning is randomly set by the *Shift* variable. At the receiver the slot long data is correlated with the PN code. The peak resulting from the correlation is the location of the beginning of the PSCH, which also corresponds to the beginning of a slot. A simulation result of an example of the PSCH search is shown in Figure 3-8. In the simulation the PN code of the PSCH was inserted at chip number 413. As seen in the figure, the peak is located at chip number 413 which points to the start of the slot.

The PSCH data is a random data of 1s and -1s. Figure 3-8 shows the PSCH as a bold line between 1 and -1. The resulting correlation is shown as the light line in the figure.

The location of the beginning of the slot is then passed to the SSCH search hardware. The SSCH search process is accomplished by 16 parallel correlators. Each correlator searchers the data of the SSCH for one of the 16 prior known PN codes. Since the beginning of the SSCH data is known from the PSCH search step, all correlators start the correlation operation at the same time. Only one of the correlators will recover the 16 bit binary data in the SSCH which points to the group of the cell’s scrambling code.

The Channel estimation unit estimates the multipaths delays and complex tap coefficients (phase and amplitude) [33]. The channel estimation performance

```
function [LOC]=PSCH
PN=dl_sc_code(300);
PN=PN(1:256).*-2+1;
X1=round(rand(1,2304)).*-2+1;
X2=[PN, X1];
Shift= round(rand^2*1000)
X=[X2(Shift+1:2560), X2(1:Shift)];
XD=[X,X];
PEAK=[];
LN=length(PN);
LOC=0;
thr=100/100*LN;  %if 90% of the PN matches X then it is a peak.
for i=1:length(X)
     Peak=sum(XD(i:LN+i-1).*PN) ;
    PEAK=[PEAK, Peak];
    if Peak == thr

        LOC=i

    end;
end;
LOC=2560-LOC+1  %+1 since index starts at 1.
 plot(1:length(X),XD(1:length(X)))
 hold on
 PEAK=PEAK(length(X):-1:1);
 plot(1:length(X),PEAK,'r')
 hold off;
```

Figure 3-7: A Matlab code fragment to simulate the Primary Synchronization Channel (PSCH)

is heavily affected by the channel quality. The implementation of the searcher is typically based on sliding correlators. The greater the number of parallel sliding correlators the faster the channel estimation can be done.

3.3.3 RAKE Receiver and Maximal Ratio Combining

The WCDMA communications system is based on the DS-SS baseband data modulation. This implies that the signal's spectrum is expanded, i. e. the signal energy is distributed over a much larger bandwidth than the minimum required for transmission. In direct sequence spread spectrum (DS-SS), the signal is spread

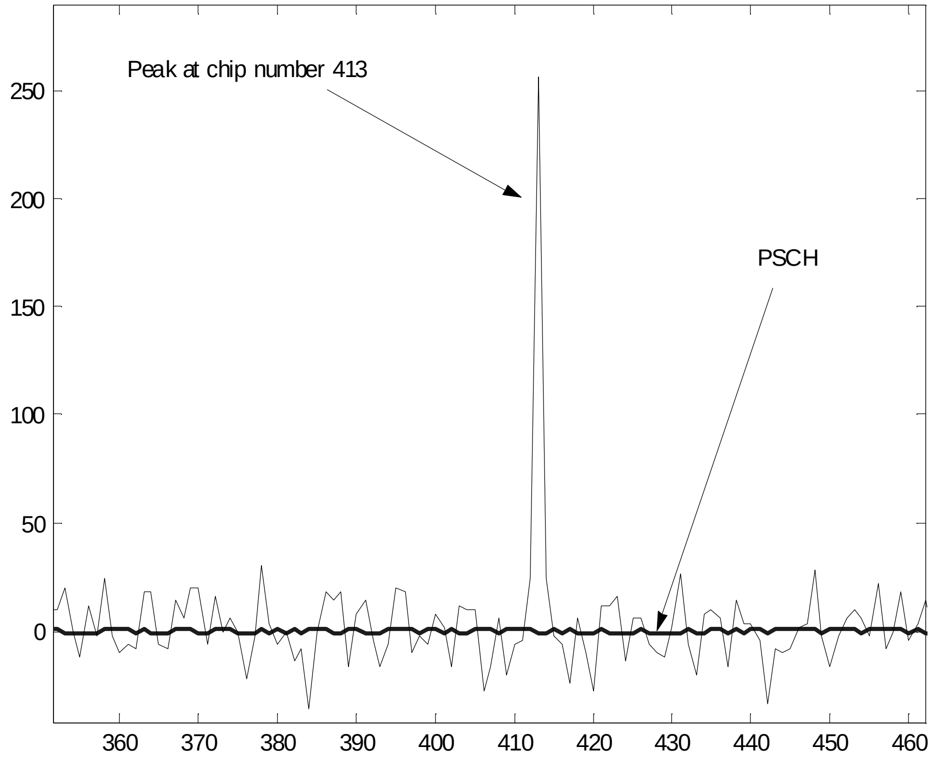


Figure 3-8: Matlab simulation of the PSCH search

by multiplying it with a PN sequence with a much higher chip rate. In the transmitter, the signal is multiplied by the spreading sequence which causes a spectral spreading of the original narrow band signal. At the receiver the signal is multiplied by the spreading sequence again. If the reference sequence of the receiver is synchronized to the data modulated PN sequence in the received signal, the original signal can be recovered [33], and [56].

The RAKE is a special type of receiver that takes advantage of the multipaths propagation. If the time spread of the channel is greater than the time resolution of the system then different propagation paths can be separated, and the information extracted from each path can be used to increase the signal to noise ratio (SNR). The time spread of the channel is given by the maximum delay between the arrivals of a transmitted signal on different propagation paths. The time resolution of the system is given by the inverse of the bandwidth of the radio frequency signal, or is equivalent to the chip period of the PN sequence [58], and [59].

Figure 3-9 is a block diagram of L-arm RAKE receiver. The RAKE receiver is composed of two or more correlation arms, which extract the signals, arrived on different propagation paths. This is possible because the correlation between two versions of the PN sequence delayed by one or more chips is almost zero. Therefore the propagation paths are separable [60].

As shown in Figure 3-9, once the different paths are resolved, they are combined based upon their relative weights. Various techniques are known to combine the signals from multiple diversity branches. In Maximum Ratio combining each

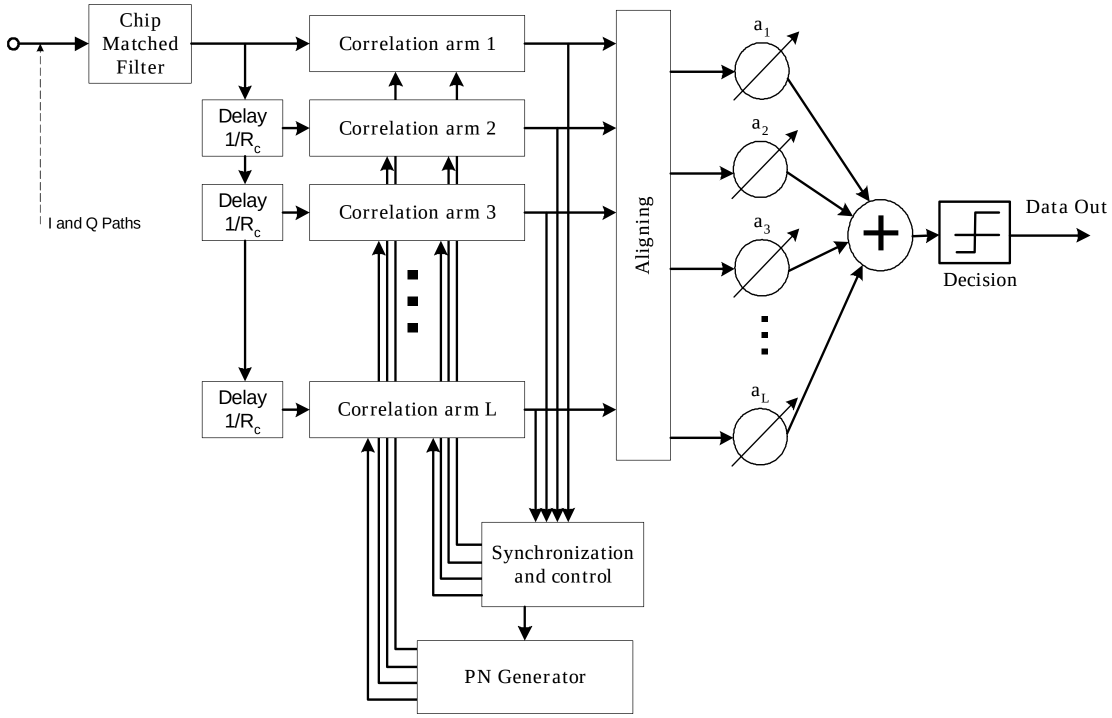


Figure 3-9: A block diagram of L-arms RAKE receiver

signal branch is multiplied by a weight factor that is proportional to the signal amplitude. Therefore, branches with strong signals are further amplified, while weak signals are attenuated.

A communication-link level simulation in Matlab was carried out (it is also reported in [61] and [62]). The communication-link level model is shown in Figure 3-10. A random binary (+1, -1) data is generated by the Data Generator. The data is generated at a rate of R_d. The random data is then up-sampled to a chip rate R_c

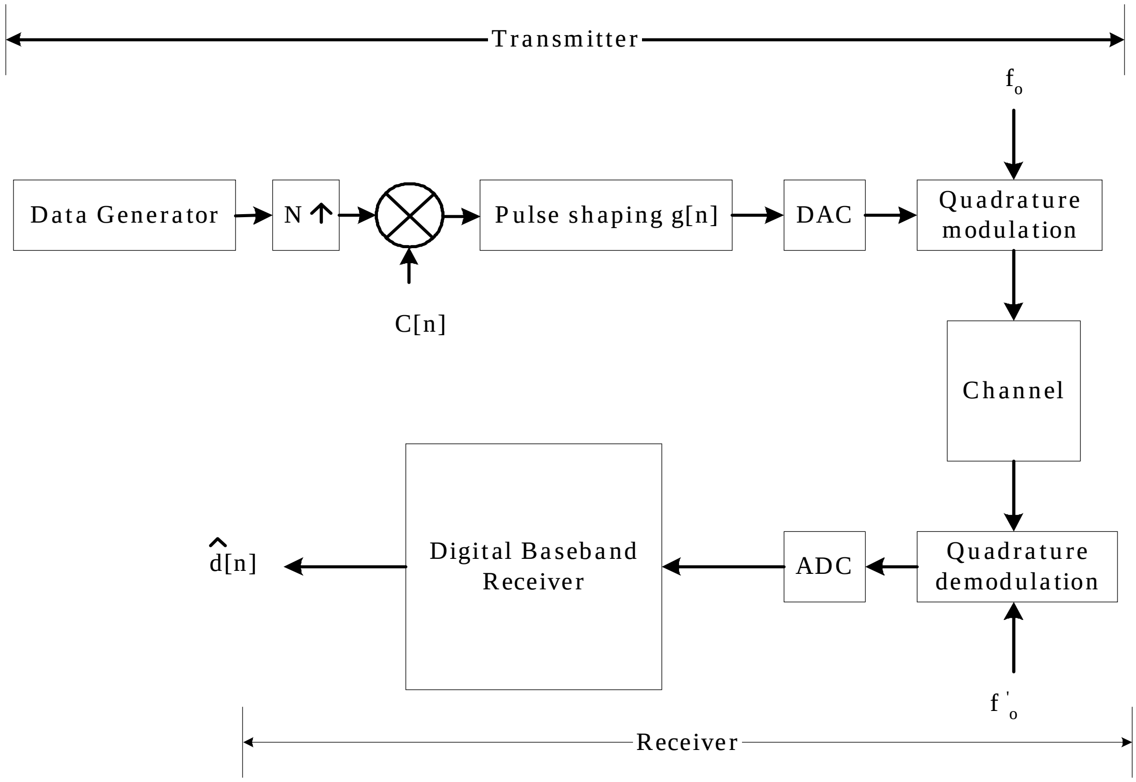


Figure 3-10: Communication-link level model for the RAKE receiver simulation

for spreading. Subsequently a complex spreading operation is carried out on the binary data. The PN sequence used is a complex PN code:

$$c[n] = c1[n] + j^*c2[n] \qquad \textbf{(3-1)}$$

Where $c1[n], c2[n] \in \{-1, 1\}$.

The spreading gain N is:

$$N = R_c/R_d \qquad \textbf{(3-2)}$$

To reduce the inter symbol interference a Root Raise Cosine filter is implemented. Next, the signal is modulated and sent through the channel.

The channel was modeled by a Tapped delay model shown in Figure 3-11.

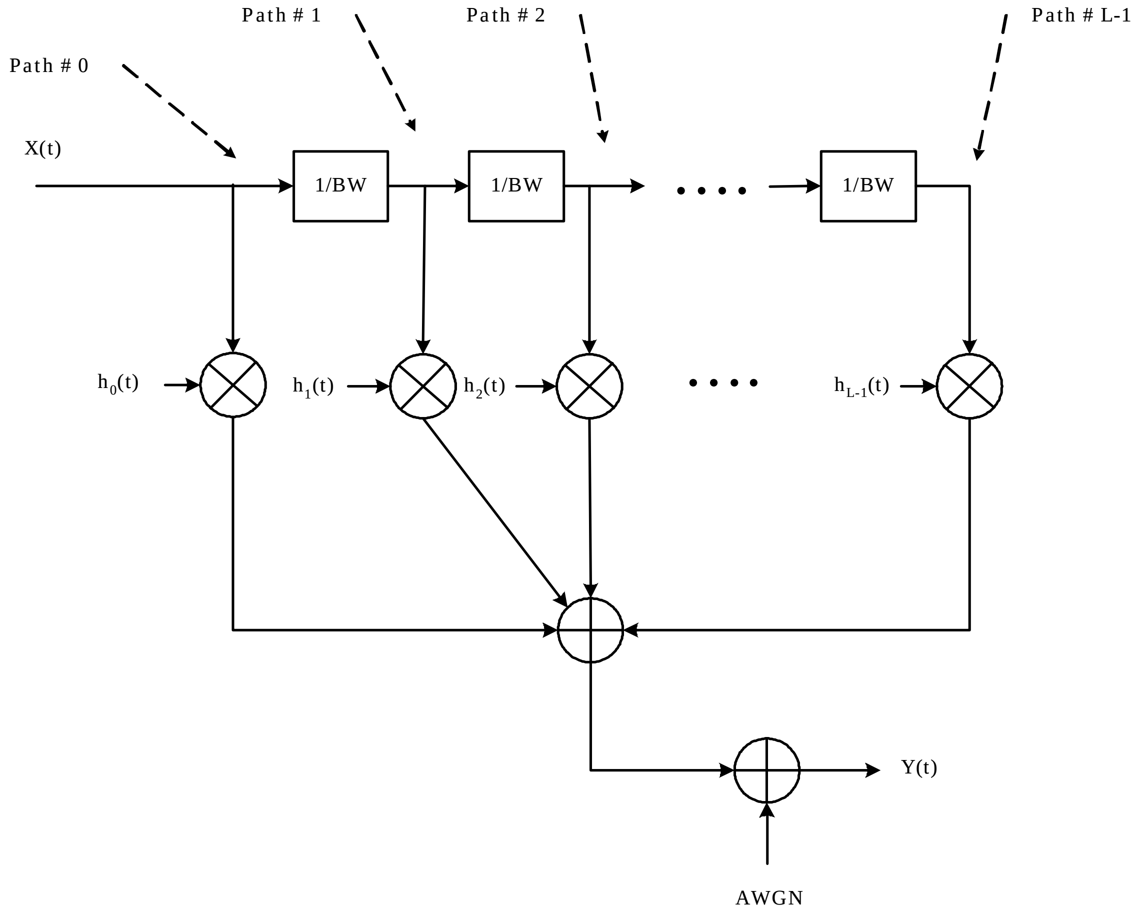


Figure 3-11: Tapped-Delay channel model

The channel model represents a Rayleigh fading channel, with L separable paths:

$$L = int(B_w T_d) \qquad \textbf{(3-3)}$$

where B_W is the bandwidth of the transmitted signal and T_d is the time spread of the channel. The coefficients $h_0(t)$, $h_1(t)$,... $h_{L-1}(t)$ in Figure 3-11 are the complex impulse response of the channel. Noise generated by different parts of the

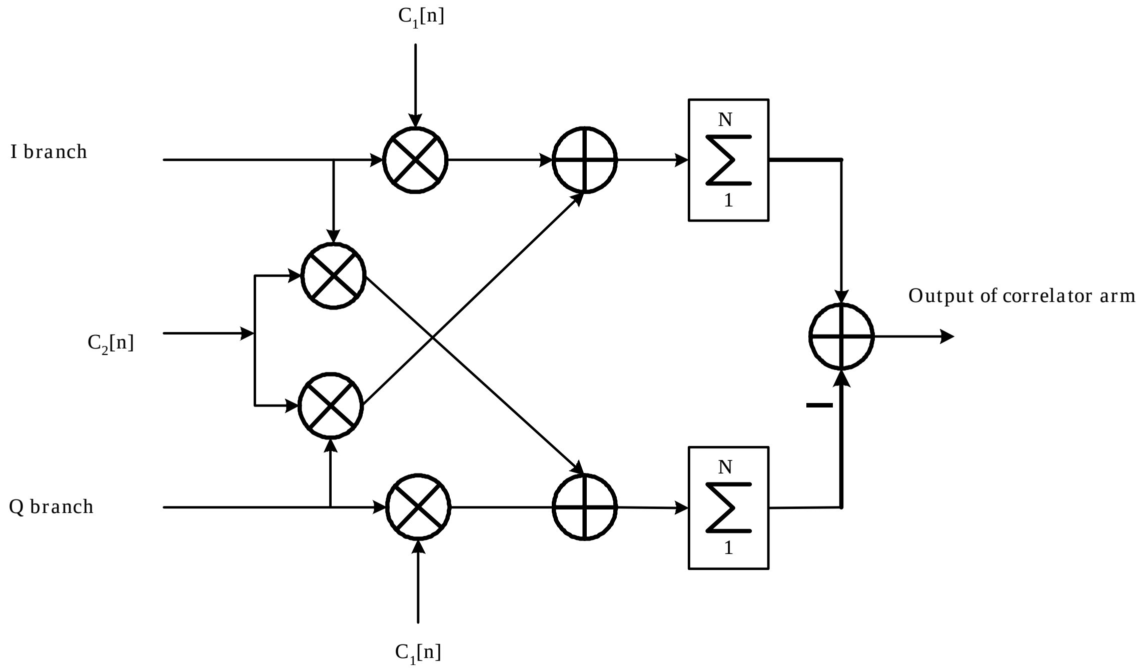


Figure 3-12: One Rake finger correlation arm

front-end of the receiver has a relatively flat power spectral density and a Gaussian probability density. This noise is modeled as additive white Gaussian noise (AWGN).

The signal is demodulated in the receiver then it is sampled and converted from analogue to digital by the ADC block. It finally enters the digital baseband receiver, as presented in Figure 3-9. The chip matched filter shown in Figure 3-9 has the same impulse response as the pulse shaping in the transmitter, resulting in a raised cosine filter effect which gives a zero inter symbol interference.

The correlation arms perform the correlation with the synchronized copy of the PN sequence used in the transmitter. A one correlation arm of one RAKE finger is shown in Figure 3-12.

Two simulation results (other simulation results reported by a collaborative work in [61] and [62]) are relevant here. The BER dependence on the number of quantization bits and BER dependence on the number of correlation arms.

For the BER versus the number of quantization bits, two simulations were performed, one for SNR = 5dB, another for SNR = 10dB (see Figure 3-13). As expected, the increase in the quantization bits decreases the BER. Interestingly, for more than 6 bits/sample the improvement in the performance is not linear. If higher signal to noise ratios values are received then the number of quantization bits can be decreased to 4 or less.

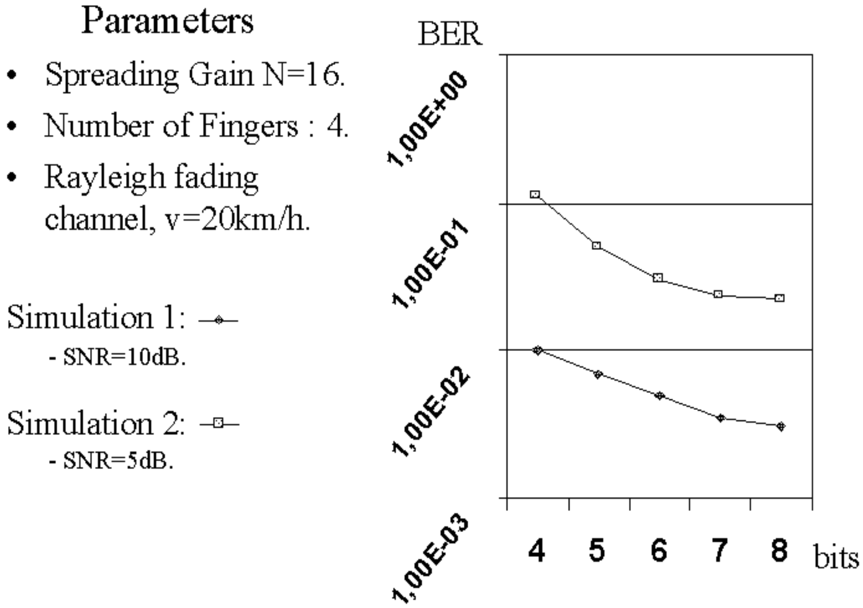


Figure 3-13: The BER as a function of the number of bits used for the quantization

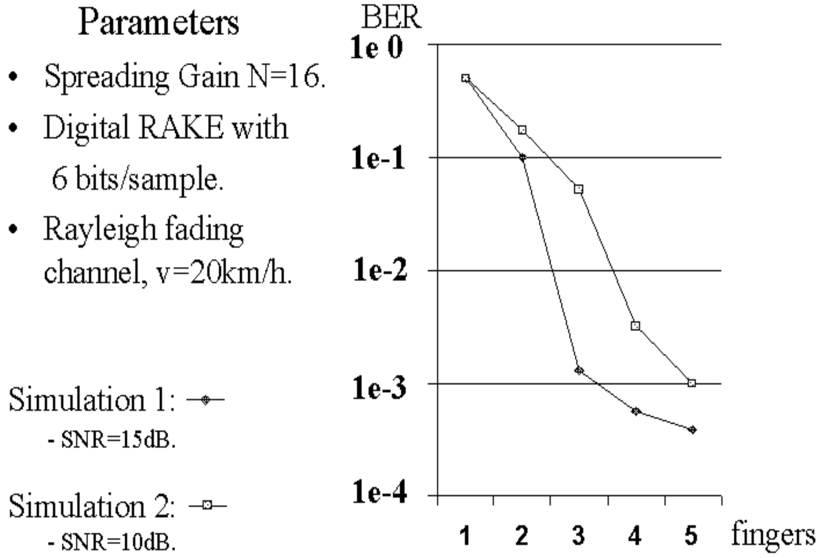


Figure 3-14: BER as a function of number of RAKE fingers

To characterize the BER versus number of RAKE fingers two simulations were performed, one for SNR = 15dB, another for SNR = 10dB. The simulation result is shown in Figure 3-14.

As expected from Figure 3-14 one can see that as the number of fingers increases the BER decreases. For a receiver that uses less than 3 fingers, a severe amount of error bits will be present. This simulation did not consider the use of Forward Error correction techniques or interleaving.

3.3.4 Turbo Coding

Turbo coding is an advanced forward error-correction algorithm. It is a standard component in third generation wireless communication systems [39]. Basically, turbo coding involves two operations. The turbo encoder generates a data stream that consists of two independently encoded data streams and a single unencoded data stream. Due to the interleaving, the two parity streams are weakly correlated. In the turbo decoder, the two parity streams are separately decoded with soft decision outputs, which are referred to as “extrinsic” information. The strength of the turbo decoding lies in its sharing of the extrinsic information (as the information is passed over parity decoding steps) over a number of iterations [63], and [56].

The 3GPP standard channel coding schemes which can be applied to the incoming data are either turbo coding, convolutional coding, or no coding. The scheme of the turbo coder is a Parallel Concatenated Convolutional Code (PCCC) with two 8-state Recursive Systematic Convolutional (RSC1 and RSC2) encoders and one turbo code internal interleaver, as shown in Figure 3-15.

The three output signals generated by the encoder are called: systematic (original signal *S*), and redundant signals (Parity bits *P1*, and *P2*). The input to the second encoder RSC2 is first interleaved using a random-like interleaving process. The encoder operates on a block of bits, which is between 40 to 5114 bits long [71].

General Turbo decoder structure is shown in Figure 3-16. The turbo decoder consists of two Soft-Input Soft-Output (SISO) decoders. An SISO decoder is capable of computing a *posteriori* likelihood of the constituent codes.

As can be seen in Figure 3-16, the first decoder is provided with inputs that correspond to the information bits ($S_1^{'}$) and the parity check bits ($P_1^{'}$). The second decoder is provided with inputs that correspond to the information bits ($S_1^{'}$) and

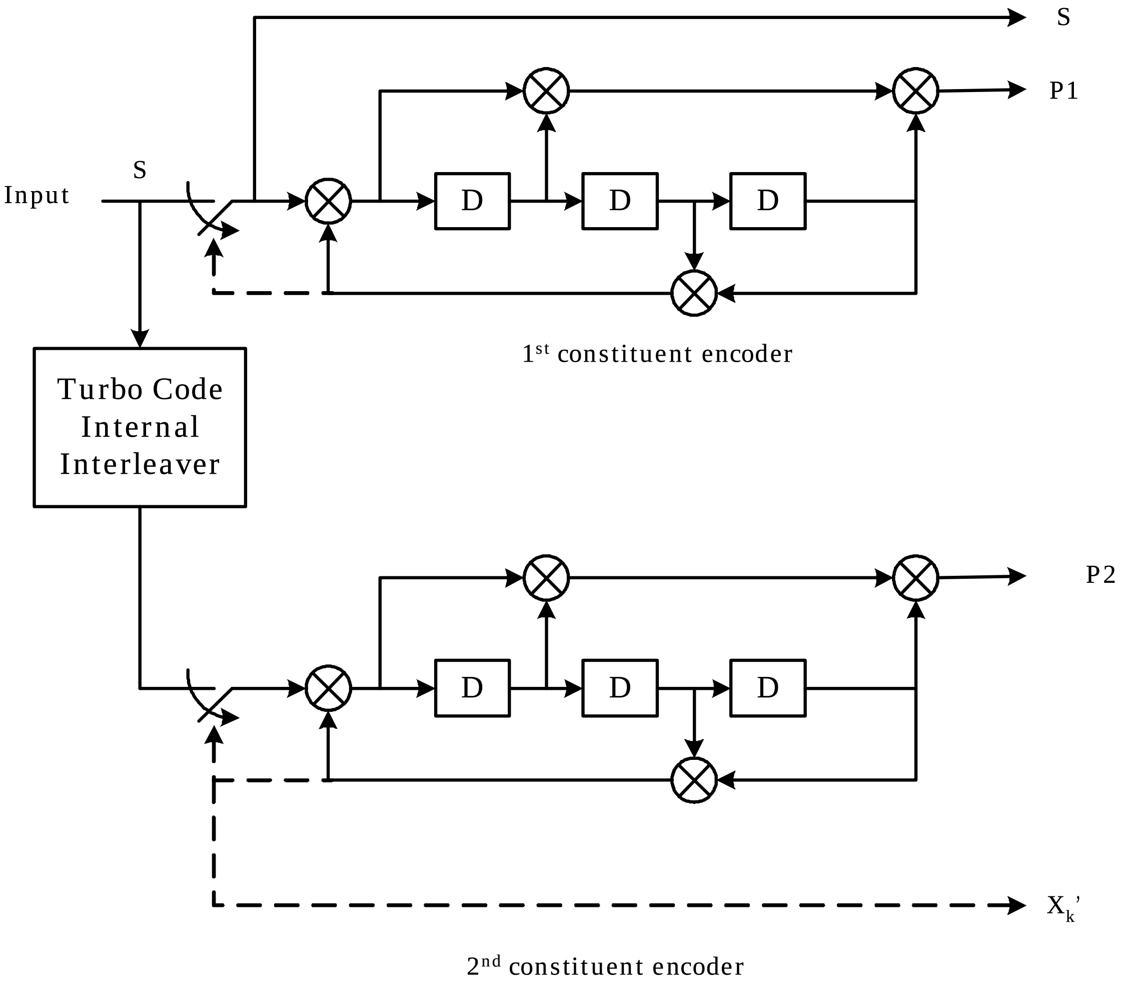


Figure 3-15: 3GPP proposed structure of the turbo coder for WCDMA

the parity check bits ($P_2^{'}$). Each decoder then attempts to decode the incoming bits stream by computing the *posteriori* probability for each information bit and then passing the soft-output information to the other decoder which uses this soft output as a *priori* informations to improve its probability of correct detection [56].

A matlab simulation of the turbo encoder and the turbo decoder using Max-Log-MAP algorithm was carried out. Random binary data (+1,-1) was generated and encoded. The simulation parameters are shown in Table 3-2.

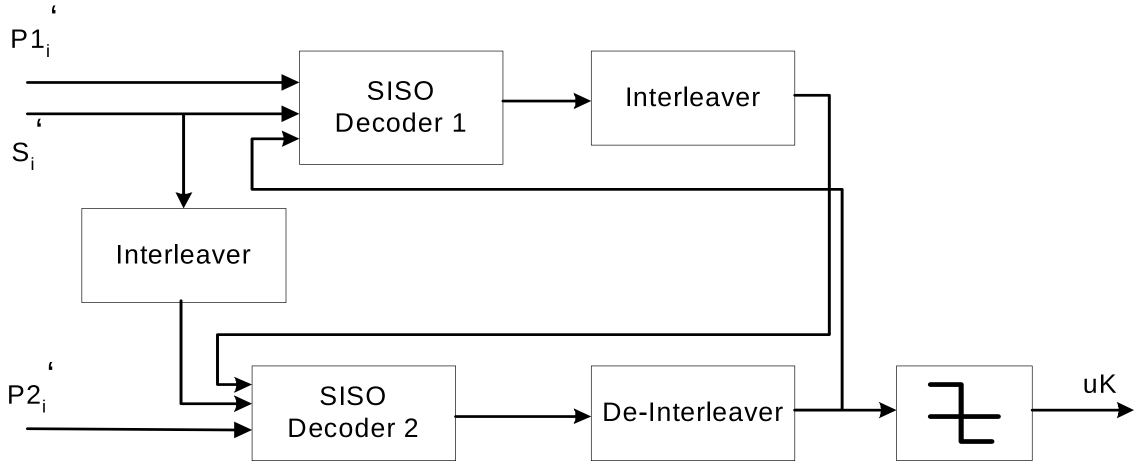
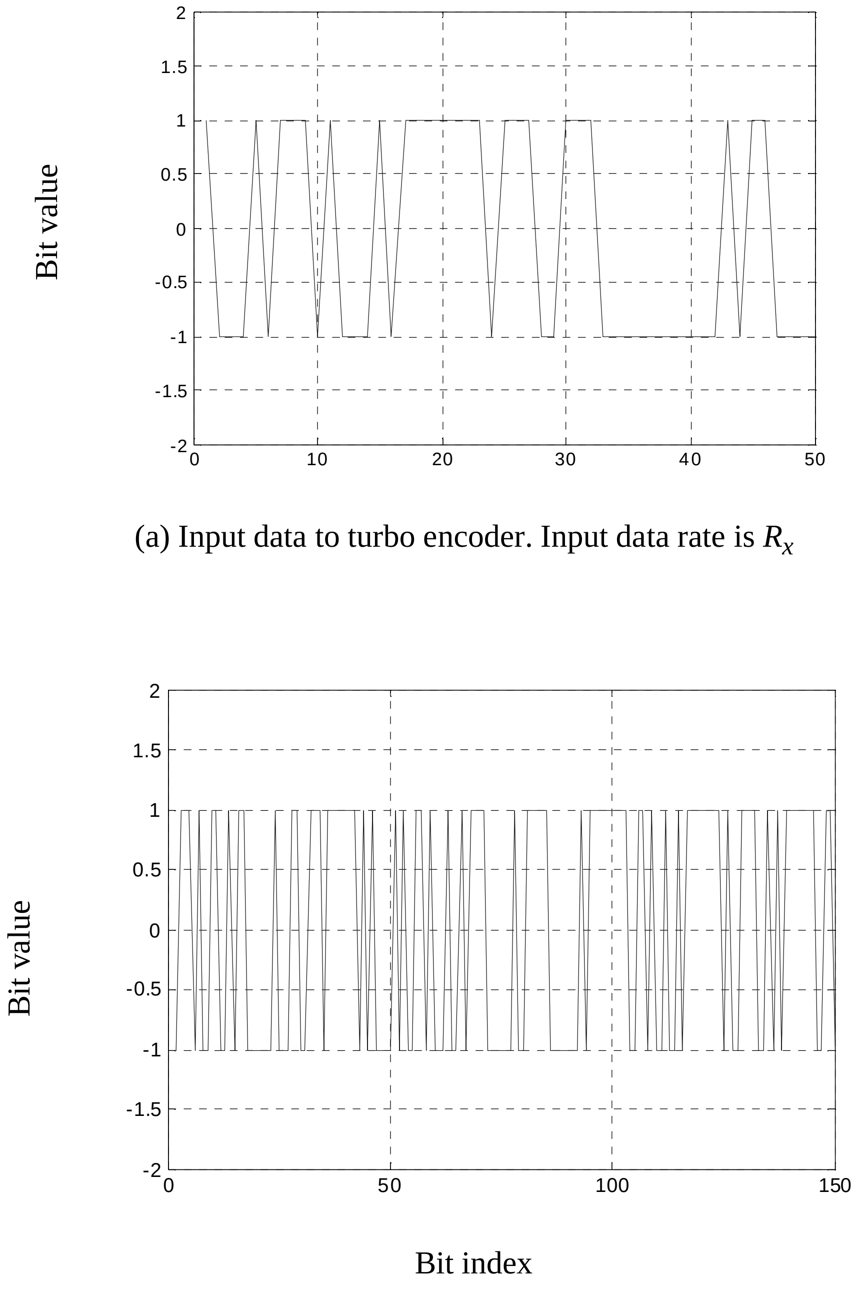


Figure 3-16: Turbo Decoder structure

Table 3-2. Matlab simulation parameters of the turbo encoder/decoder.

Parameter Name	Parameter value
Frame size	512 bits
Punctured/Unpunctured	Unpunctured
Encoder Rate	1/3
Number of Decoding Iterations	3
E_b / N_0 (dB)	2 dB

Figure 3-17 shows the input and output of the turbo encoder. As shown in the figure, the output data of the encoder is 3 times the rate of the input data. The output data of the encoder is then sent through the communication link channel were a white Gaussian noise is added to the signal. The decoder at the receiver uses a Max_Log_MAP algorithm. The output of the decoder is shown in



(a) Input data to turbo encoder. Input data rate is R_x

(b) Output data from the turbo encoder. Output data rate is R_e

Figure 3-17: Matlab simulation of turbo encoder: (a) input to the encoder (b) encoded data bits

Figure 3-18 . For the Max_Log_MAP decoder, three iterations were sufficient to reduce the BER to zero using the given parameters.

According to the BCJR algorithm [1], the calculation of *a posteriori* probability can be decomposed to the calculations of α, β ,and γ parameters, as follows:

$$p(s', s, y) = \alpha_{k-1}(s')\gamma_k(s, s')\beta_k(s) \qquad \textbf{(3-4)}$$

Where s and s' are the encoder states at time k, and k-1 respectively, and:

$$\alpha_k(s) = \sum \alpha_{k-1}(s')\gamma_k(s, s') \qquad \textbf{(3-5)}$$

$$\beta_{k-1}(s) = \sum \beta_k(s')\gamma_k(s, s') \qquad \textbf{(3-6)}$$

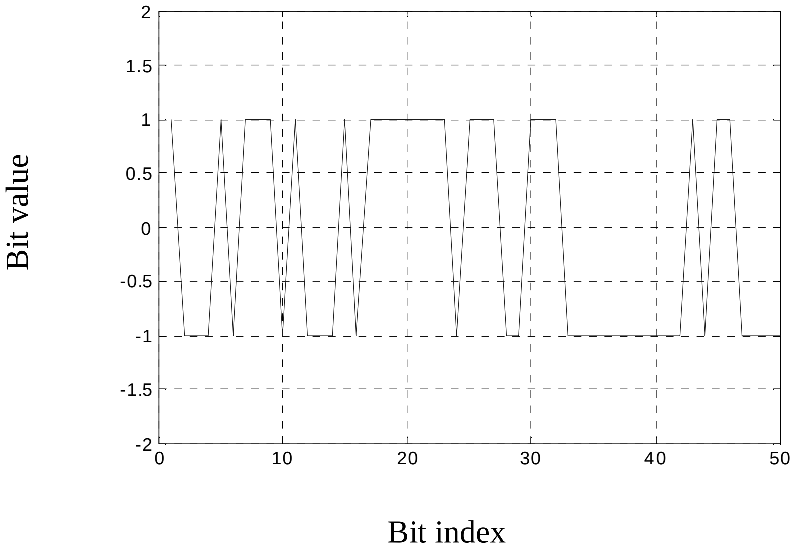


Figure 3-18: Turbo decoder output data after three iterations

In an Max-Log-MAP algorithm, the calculations of α, β ,and γ are simplified to performing a sequence of addition, subtraction and maximum operations as reported in [63].

Chapter 4

Computing Architecture Models

In this chapter I will investigate different architecture models available as computing platforms for the current and future generations of handheld wireless mobile terminals. An in-depth discussion will reveal the differences between the classical computing platforms and new emerging reconfigurable computing platforms. I will then concentrate on discussing different reconfigurable computing platforms that are available on the market and in the research organizations.

4.1 Introduction

Most of the computing platforms used for the wireless mobile terminal application today are a modified version of a general purpose Digital Signal Processor (DSP) [8]. These specially designed DSP are an improvement over general purpose DSPs in terms of performance and power consumption. They are widely used in mobile terminals since they provide a very fast time to market. They can be easily upgraded by updating the program code written in high level languages.

The code enables the terminals to perform any new functional requirements that may arise from changes in the standards.

Other operational functions with higher performance requirements than those provided by DSP are implemented using dedicated hardware VLSI chips. These chips provide the highest performance and lowest power consumption for a selected application. However, they are the most expensive and time consuming in terms of design, fabrication, and validation. Presently, a DSP-ASIC combination, in addition to other supporting blocks such as memories and I/O interfaces, have always been the choice of all previous and current wireless mobile terminals [64]. However, as the number of functions provided by the mobile terminal increases, additional area and power are required. Reconfigurable computing architectures are able to efficiently use the scarce area and save the power on the mobile terminal.

4.2 Current Computer Architectures

The typical implementation process of a given function starts with defining the function with a natural description, i.e. in terms of the spoken words. The designer will translate the natural description into an algorithm written in a standard sequence of tasks (instructions) that will perform the intended function. In the final step, the designer will look for physical hardware to implement and execute the set of instructions. In the current hardware implementation, there are only three options to implement an algorithm. They are: ASIC, General-Purpose

Microprocessors (including DSP), and Field Programmable Gate Array (FPGA). See Figure 4-1.

Once one of the above approaches has been selected, we lose the chance to take advantage of the natural functionality of the algorithm. These characteristics include the required resolution (i.e. number of bits) for the operations and the parallelism available in the original description.

The dynamically reconfigurable architecture developed in this dissertation provides a deep pipelined and system level parallelism implementation hardware. The architecture uses a 16-bit grain datapath. Although, a smaller data-path width may be sufficient for some of the applications in hand, as new functionality arises from the available high data rates, wider datapath widths will be needed.

Designing an architecture that can adopt to any number of bits require that the design of processing elements operate on a one-bit level. Such architecture is a fine-grain architecture. A major drawback of such architectures is the problem of routing the signals inside the architecture. By looking at the target application we see that all the functions require a minimum of an 8-bit data width. The bit width of the output of the analog-to-digital converter also plays a role on determining the number of available bits for the processing elements. Current ADC can provide 12-bits and more sampling resolution. Some of the functions of the baseband unit can take advantage of a wider data width. For example, increasing the number of bits representing the coefficients of the FIR filter reduces the required number of taps.

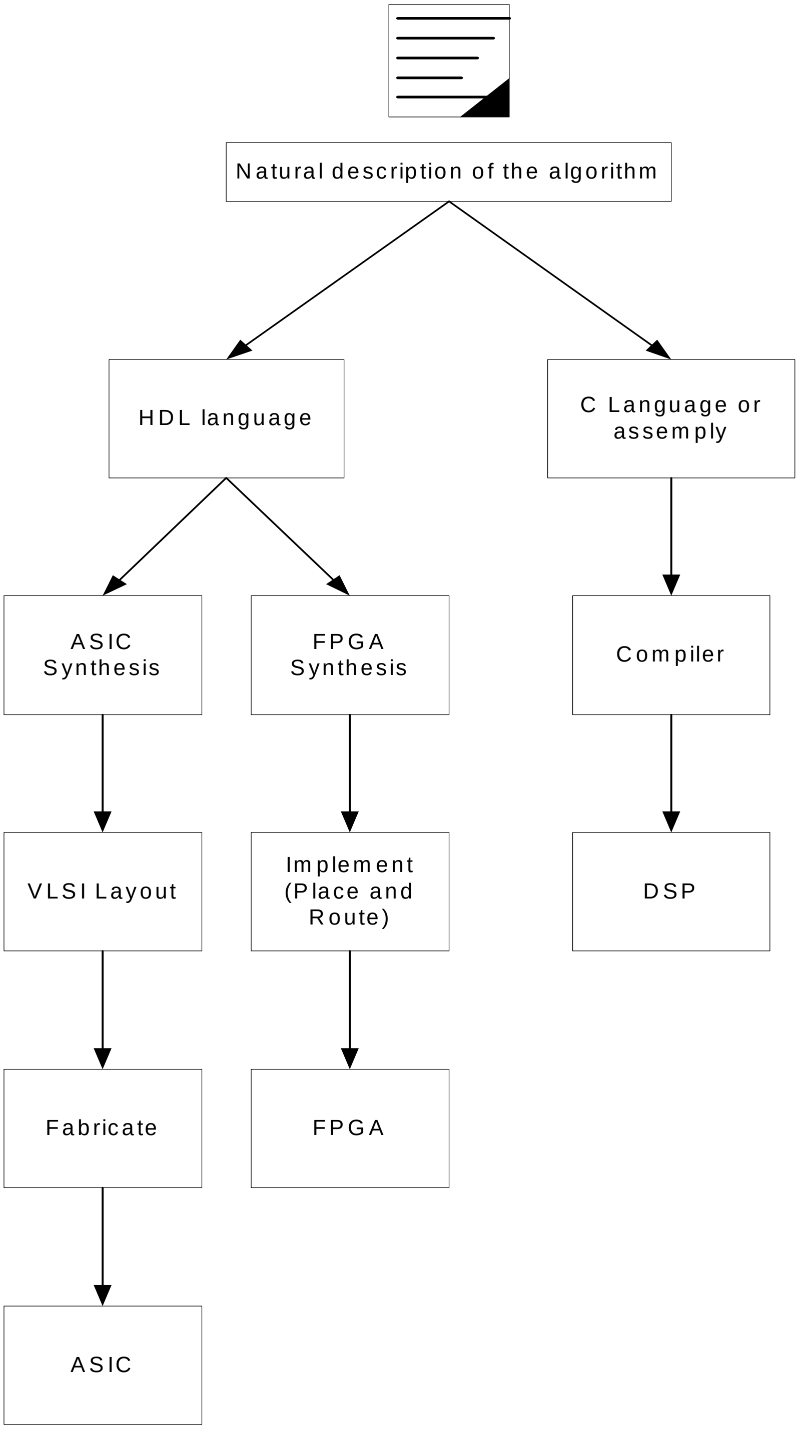


Figure 4-1: Design process flow of different hardware implementations

4.3 ASIC implementation

ASIC is one of the most common methods of the target algorithm implementation today. ASIC uses a direct mapping of the natural algorithm into hardware. This approach yields the highest performance and the lowest levels of power consumption. As the name implies, each ASIC chip is designed for one specific purpose or function. Once the design is committed to silicon no changes to the implementation are possible. Additionally, since an ASIC needs to be designed for each function, systems tend to incur a huge final cost. Therefore, ASICs are best suited for applications that are produced in huge quantities.

Other limitations to ASIC's implementation approach comes from the way ASICs are designed. Typically, the target algorithm needs to be coded in Hardware Description Language (HDL). This procedure is necessary in order to feed a HDL code to a synthesis tool which will then translate the HDL code into boolean equations that are suitable for mapping into logic gates and other logic units. A problem that is created by this procedure stems from the fact that HDL languages and particularly VHDL are designed to describe hardware functionality (implementation) and are not meant to describe the algorithms in its mathematical nature. Therefore, whenever we describe an algorithm in VHDL we indicate how the implementation of the hardware is going to be designed.

There is a major obstacle related to using ASIC in a system. Using ASICs imposes an extensive real state requirement on the mobile terminal. We need an ASIC chip for every function we have in the system, whether it is required at a given time or not. Implementing such a system with ASIC chips would result in a bulky final product that would be impractical to use [51].

Another limitation of ASICs pertains to their time-to-market cycle. It takes three months (or more) for an ASIC to be designed, verified, and fabricated before hitting the market. For dynamic systems, like the wireless mobile system, the standards are continuously evolving. Even when the standards become mature and freezes new services will emerge. New algorithms for already known services may be developed. Therefore, the ASIC would become obsolete while they are in the process of being designed.

4.4 DSP Implementation

Digital signal processors are widely available and are used in many systems including current mobile terminals. They are available in a variety of sizes and capabilities. Each DSP is a modified version of a general purpose micro processor. Thus, they all experience the sequential architectural model, i.e. fetch-process-save. To implement an algorithm in DSP, one needs to code the algorithm either directly in assembly or in a higher level language such as C which is then converted by the compiler into a sequence of machine instructions.

In contrast to the ASIC implementation, the DSP implementation is the most flexible. It provides the designer with the ability within a few minutes to modify the code and accommodate any changes that may occur in the algorithm. This implies that this approach enjoys the shortest time to market cycle. The speed of modifying the DSP code explains the reason why it is widely spread use in systems that have fast dynamic changing behavior such as the mobile system.

The coding of the algorithm into a sequence in a high level language implies that the algorithms intrinsic parallelism is lost. Additionally, the required processing may not be compatible with that of the processor. Therefore, the designer is required to modify the algorithm even more in order to match the algorithm processing with that of the processor.

A significant silicon time-area overhead is incurred in DSPs. The reason for this is that for any given instruction only a small portion of the time (allocated for that instruction) and a small number of logic cells (available for the instruction) are utilized to do anything related to the original operation [64]. For example, when a DSP is instructed to add two values, most of the clock time is wasted in fetching the instruction, incrementing the counter and setting the appropriate control signals. Only a small portion of the time and hardware are consumed by the addition operation.

4.5 FPGA Implementation

Selecting an FPGA implementation approach would provide a better hardware utilization than a DSP approach and be more flexible than the ASIC approach. The FPGA can be programmed to reflect the appropriate processing width and also at the same time, be reprogrammable in a short time. Field programmable devices are an advanced class of traditional programmable logic devices. Unlike ASIC, a programmable device is a general-purpose device capable

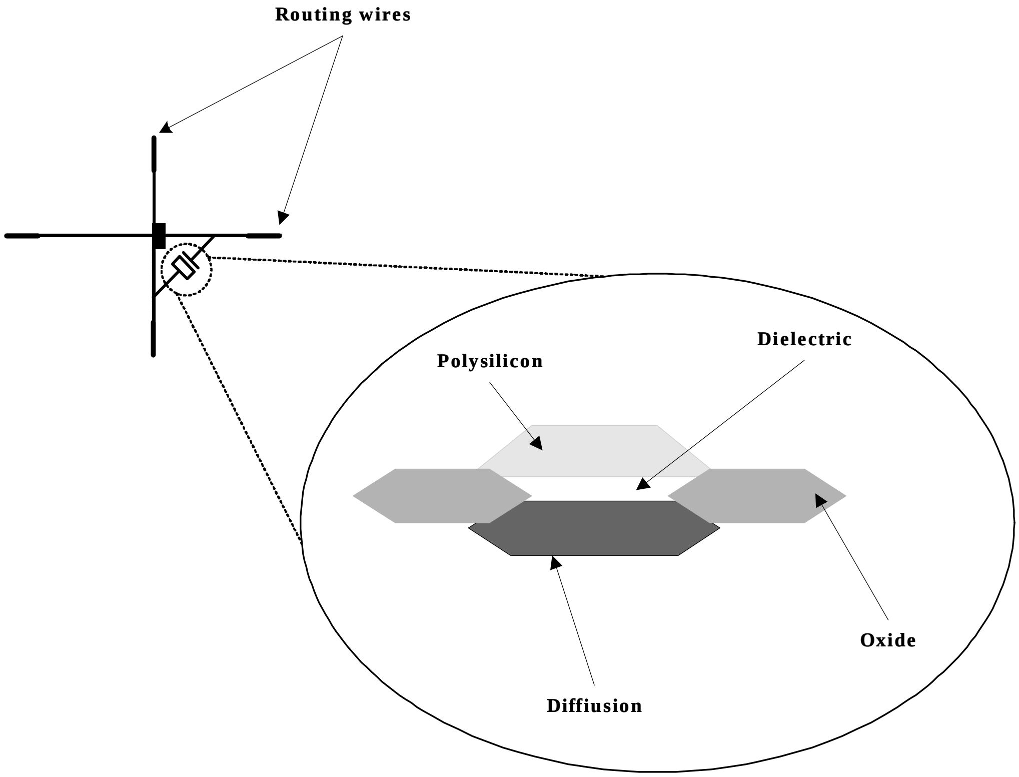


Figure 4-2: An Anti-fuse switch for non-volatile FPGAs

of implementing any logic function. It is programmed by downloading a finite number of hardware controlling bits into the device.

FPGAs are composed of two main components. The Processing Elements (PE) and the routing resources. They can be classified into two main categories depending upon their programming methods. The two FPGA types are anti-fuse FPGAs and Static RAM FPGAs. These two types come in two different architectural-granularities variations, which are mostly fine-grain or coarse-grin FPGA architectures. Anti-fuse FPGAs consist of a number of PEs where the configuration is done by burning fuses inside the PE and in the routing paths (see Figure 4-2).

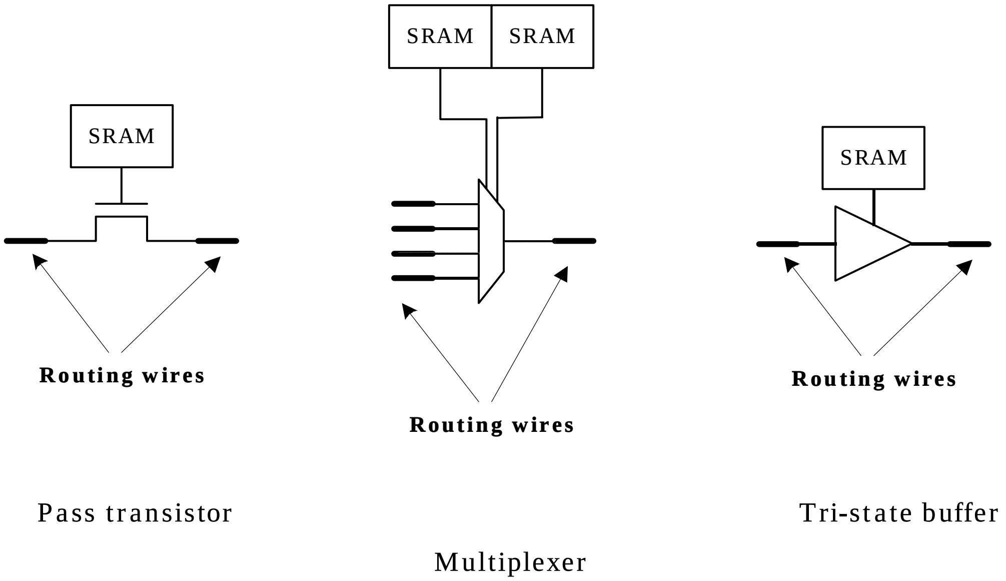


Figure 4-3: SRAM cell for SRAM based FPGAs

This method of programming restricts the chip to only being able to be programed once. But the advantage of such a method is the elimination of switches from the path of the signal inside the chip, which results in higher levels of performance.

SRAM FPGAs are composed of a sea of PEs which can be configured by setting SRAM bits inside the PEs and in the routing paths. These stored bits are used to drive routing switches. Figure 4-3 shows three types of programmable switches: pass transistors, multiplexers, and tri-state buffers. This method provides the ability to reprogram the chip an unlimited number of times [22].

Due to the fact that FPGAs were originally designed for general purpose usage such as glue logic, they are designed to perform processing at the one-bit level. This greatly limits the ability to use them for processing wide data-paths such as 8 or 16-bit widths. Implementing a data-path processing engine of such widths would consume a huge portion of the FPGA chip area, since each bit processing needs to be set separately. It will require a large number of configuration bits to configure the processing elements and the routing paths. Whereas it could be configured using only one bit if the FPGA routing lines were 8-bit in width. For example, if the data path were 8-bit, then one needs to configure eight similar processing data paths. One also needs 8-bits to switch this 8-bit bus at the switch box, which route the line in different directions. Instead with coarse grain FPGAs, this can be done by using only one set of configuration bits for the entire eight bits processing data path.

FPGAs, in general, suffer from the slow programming mechanisms. Typically, the programming time of an FPGA is in the millisecond range and varies depending upon the FPGA type, the chip size, and the number of gates in the design. Additionally, configuring the FPGA is known to draw a great amount of power [22].

Dynamic reconfiguration means that an FPGA can be reconfigured while it is running. The ability to reconfigure part of the chip on the fly is a very complicated issue. Most of today's FPGAs either do not support dynamic reconfiguration at all or they would support it in a very restricted way. The result is the loss of the desirable features of dynamic reconfiguration.

In the dynamic reconfigurable architecture developed in this dissertation, a full dynamic reconfigureability is supported. The configuration/reconfiguration process is accomplished in a one clock cycle. This is done by designing the configuration hardware to have its own routing lines and not sharing the system routing lines as it is done commercial FPGAs [6][7]. In addition, the configuration contexts bits are pre-cashed and stored in the configuration memory unit located very close to the processing element. A detailed description of the configuration mechanism is given in section 5.3.7.

FPGAs are designed to be a general prototyping target. However, they suffer from two routing problems. On one hand, routing usually results in un-predicted delays in the signal path. On the other hand, the inefficiency of the routing tools used in FPGAs typically results in wasting almost 50% of the computing resources on the chip [22].

Like ASICs, FPGA design tools and programming environments are built to use hardware description languages such as VHDL as the main formal description. What was said about using HDL in ASIC can be applied to FPGAs. Using HDLs to describe the function on the algorithmic level will result in losing many of the inherent characteristics of that algorithm. For example, assume that a function *F* can be described by two algorithms *A1* and *A2*. Also assume that algorithm *A1* requires the least power-consumption of the two algorithms, while *A2* algorithm provides a faster operation. Suppose that the function *F* is a dynamic function that is related to external parameters, such as the communication link channel. Now, if we selected *A1* or *A2* algorithm for the implementation of the function *F* on ASIC or FPGA then we either lose flexibility or performance. *F* may sometimes need a high speed execution, while at other times it may not need that speed and we could save power. If a reconfigurable hardware is fast enough to switch between *A1* and *A2*, then we can save power whenever a high speed execution is not required.

4.6 Dynamically Reconfigurable Computing

The introduction of the third generation wireless mobile system coupled with a huge growth of wireless subscribers and the large number of new applications and services, creates a great challenge to the current implementation platforms. The designer must come up with a mobile terminal that is small, has a low power consumption, and low cost. It has been observed in [64] that the complexity of wireless systems is outgrowing the available silicon implementation predicted by Moor's Law as shown in Figure 4-4.

Moore's Law states that the number of transistors that can be fabricated on one unit area will double every 18 months. Moore further stated that the cost of doubling the number of transistors would remain the same. I.e. we will have double the number of transistors every 18 months without an increase in price. This abundant amount of processing power will enable the concept of System-on-a-Chip (SoC) to become a reality. A SoC would integrate different computation models on one die. It would merge DSP, micro processor, memory, reconfigurable array, special modules, the needed interfaces and I/O modules.

J. Rabaey in [64] pointed out that even though the SoC will be the obvious implementation methodology for future mobile systems, the SoC solution still needs to be constructed with a specific target domain of applications in mind. e.g. the SoC solution must balance the flexibility with performance within a specific application domain. Since the reconfigurable computing array is going to be part

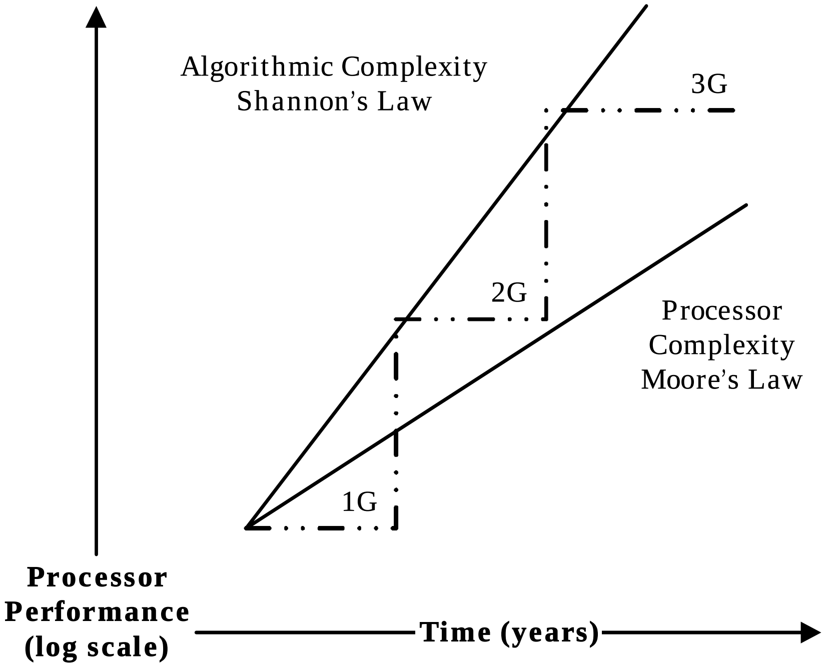


Figure 4-4: Algorithmic complexity of wireless mobile systems is increasing faster than the increase in the available processing power. Source: Jan Rabaey,

of the SoC solution, then it also must be designed for a specific target application. A platform-based design concept has been proposed by Kurt Keutzer et.al. in [65].

A general dynamically reconfigurable architecture that would provide a high level of performance for any application is a "myth" and cannot be obtained. In [13] Reiner Hartenstein states that "*universal RA (Reconfigurable Architecture) obviously is an illusion. The way to go is toward sufficiently flexible RAs, optimized for a particular application domain like wireless communication*".

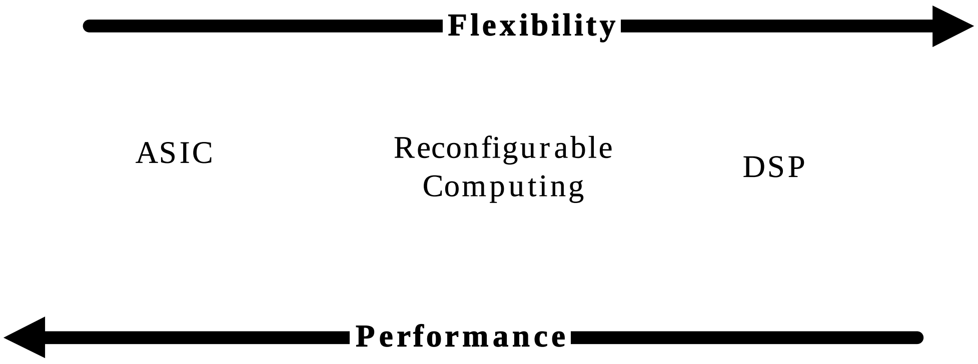


Figure 4-5: Performance and flexibility for different implementation methods

Dynamically reconfigurable architectures which are designed and optimized for one specific domain of application can provide the required balance between flexibility and performance, which are not available from an ASIC or DSP alone. Reconfigurable computing will play an intermediate role by complementing ASIC in terms of flexibility and complementing DSP in terms of performance and power dissipation. Figure 4-5 depicts where one would locate the reconfigurable computing in the performance versus flexibility axis.

As mentioned above, third and potentially future generations of mobile computing are being developed at a very fast pace. Trying to develop an implementation solution for such an application would be very difficult due to the short development time required. For example, if after an ASIC chip was developed, a new algorithm was developed which would improve the Bit Error Rate (BER) of some service by approximately 20%, then it would be impossible to conceive of updating the algorithm without having to redesign the whole ASIC. If the new algorithm required 20-bit words instead of 16-bit words and a DSP with a data path of less than 20-bits was the implementation solution, then DSP would stop short of being able to implement such a change to the algorithm. A reconfigurable computing architecture would be able to accommodate any changes in the algorithmic level as fast as the DSP would do, and provide an equivalent performance as that of ASICs.

In a typical system operation, not all of the functions implemented in the system will be running simultaneously or as frequently as the others. For example, according to the standard, a RAKE receiver would be an essential part of the receiver. Therefore it will run frequently. On the other hand, turbo coding will only be used when a high quality of service is needed. Implementing these functions in a reconfigurable architecture will give the ability to run the required function only when it is needed. In addition, the same function can be modified based upon the environment real-time conditions. For example, a RAKE receiver is typically designed to have four arms [60]. When implemented in a reconfigurable architecture, the number of arms can be increased as the mobile terminal moves to a location of many multipaths channel. Likewise the number of fingers can be reduced under different conditions.

Although current FPGAs in the market today enjoy similar architectural model as the reconfigurable architectures, they cannot meet the demands of the third generation or future mobile systems. As mentioned above, FPGA cannot support full dynamic and fast reconfiguration in the nano-second range. In addition, FPGAs are not efficient in utilizing the available logic due to the routing limitation and the fine-grain processing elements.

The dynamically reconfigurable coarse-grain architecture developed in this dissertation, presents a balance between flexibility and performance that will be required in the third and future generations of a wireless mobile system. As more SoC solutions are developed for different domains of applications following the platform-design concept presented in chapter 1, the dynamic reconfigurable architectures will be an essential part of the complete SoC solution.

4.7 IP-based Mapping

The architecture developed in this work is designed to fully support Intellectual Propriety (IP) based mapping techniques. IP mapping is the method in which a developed core is automatically mapped onto the target architecture to perform one or more functions. The IP-core concept can save thousands of hours of design work. The concept is spreading rapidly. Such cores can be mapped efficiently and rapidly onto free array parts of the underlying architecture. The cores include necessary communication interfaces between different IP-components and objects. Figure 4-6 shows the dynamically reconfigurable architecture mapping technique.

Based on functional and physical complexity levels, IPs currently available fall into three categories:

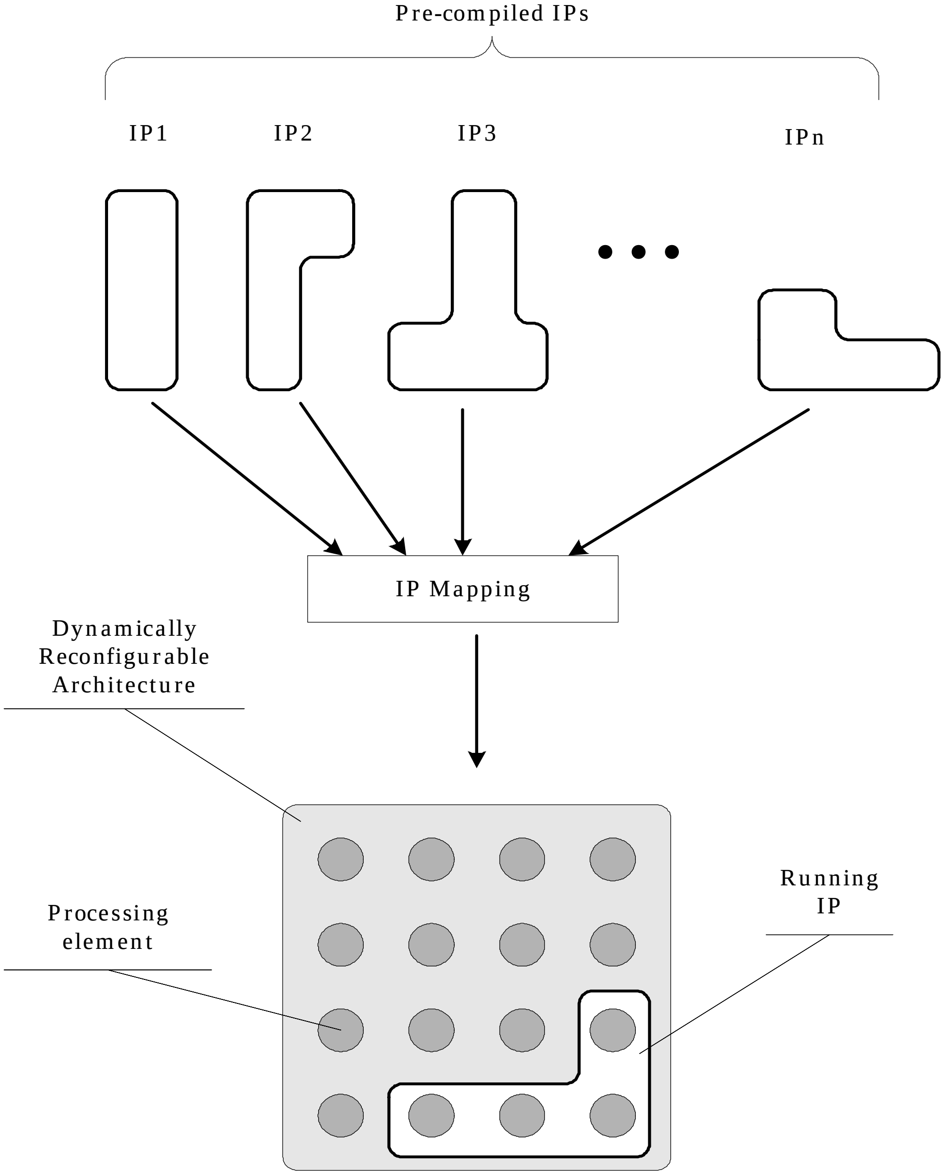


Figure 4-6: IP-based mapping method

Hard Cores: Hard cores are black boxes that have been fully implemented down to the mask-level data required to fabricate the block in silicon. They have technology-specific timing information and a fixed physical layout that allows maximum optimization in terms of performance and density. Typically, they are targeted to perform dedicated functions, such as an Ethernet interface or an MPEG decoder. However, hard cores have the most limited vendor portability and greatest difficulty of reuse when moving to a new process technology.

Firm Cores: Firm cores are technology-dependent synthesized gate-level netlists that are ready for placement and routing. They provide flexibility in the chip layout process because the core form is not fixed. A firm core's size, aspect ratio, and pin location can be changed to meet a customer's chip layout needs, and floor planning guidelines assist the chip designer in making trade-offs. The technology-specific nature of a firm core makes it highly predictable in performance and area.

Soft Cores: Soft cores consist of technology-independent synthesizeable HDL (Hardware Description Language) descriptions. They do not have any physical information.

4.8 Modeling a Reconfigurable Architecture

As with any other design problem, when designing a specific reconfigurable system it is necessary to perform two general steps. First, a deep understanding of the target application or set of applications is required. This would include analyzing algorithms of the target application(s), and simulation. Second, after designing the platform architecture, a faithful model of the architecture is required to perform a prototyping and functional as well as timing simulation.

In DSP designs the first step is usually done with MATLAB. This works great since the underlying execution behavior of the DSP and the MATLAB (general purpose processor) are identical, i.e. both execute a set of instructions sequentially. But when designing a parallel and reconfigurable architecture, MATLAB will not provide correct information about the behavior of the overall system.

When modeling a static system, VHDL is a very rich language that can easily describe the functional and (with some additional steps) the timing characteristics of virtually any system. However, modeling dynamically changing systems in VHDL can become a burden to the designer. In modeling static systems, different components are designed to do a specific task and to interact with other system components in a specific way. When the full description of such a static system is completed, any functionality of a component can be modified without affecting the other parts. Eventually it will not require modification of the overall system.

The choice of whether to model the system in a structural or behavioral modeling will not affect the final system description. Unlike the static design, dynamic designs require one to consider reusing the same hardware to do different tasks based on the configuration bits. A configurable circuit would be modeled in a structural style. However, this would require previously decided designs and hard compromises need to be taken prior to the modeling process. As a result, the final system description will result in a rigid design. A rigid design means that changes to the design require a modification of most of the design components.

This style of modeling would be more suitable at the final stage of designing a dynamically reconfigurable system when all modifications have been thought-out and a final design functionality is well established.

In the design of the reconfigurable architecture developed in this dissertation, a VHDL design model was developed. A number of compromises were taken during the design. For example, in the design of the processing element (Dynamically Reconfigurable Processing Unit DRPU), the routing of the input signal to the inside of the unit was limited to three signals selected from eight possible input signals. The decision of routing only three signals instead of routing all eight is imposed by components inside the DRPUs, i.e. by the maximum fan-in of all components. The selection of only three signals is also imposed by the number of configuration bits that will be needed by each component inside the DRPU.

Chapter 5

Dynamically Reconfigurable Architecture

5.1 Introduction

The Dynamically Reconfigurable Architecture for Wireless Mobile Systems (DRAW) combines many new concepts and features. This chapter will present a planar model suitable for analyzing DRAW architecture. Concepts such as run-time reconfiguration and run-time reconfigurable communication resources, will be discussed in detail in this chapter. Features such as dedicated spreading unit, and configurable linear feedback shift register are also presented here.

5.2 The Dynamically Reconfigurable Architecture (DRAW) Model

The DRAW architecture is designed to be a part of a System-on-a-Chip (SoC) solution for the third and future generations of wireless mobile terminals. However, the enormous size of such a project and the time restrictions imposed on this dissertation compelled us to modify the design of DRAW to a stand alone platform. It can be modified at later date to be part of a SoC platform. The completion of the design and simulation of DRAW as a stand alone platform will prove the applicability to implement reconfigurable computing platforms for the wireless mobile application. It will also enable us to experiment with architecture without the need for the complete SoC system. Additionally, DRAW can be used as a prototyping platform for the development of algorithms for the mobile baseband processing.

The set of all possible designs for a particular architecture is the design space of that architecture. When designing a reconfigurable architecture there is an unlimited number of designs to build a reconfigurable computing engine. The reason for such a wide space is the large number of design variables that need to be set (selected) prior to and during the design process.

For example, bus width, word length, and floating/non-floating point processing are major variables which need to be decided in advance. Setting some design variables at the beginning reduces the available design space. This reduces the design problem and makes the situation easier to comprehend. However, this will also reduce the chances of reaching the optimal design of the reconfigurable engine to match the desired application requirements.

The reconfigurable computing architecture organization structure is a combination of a software-programmable processor and the spatial computational style used in ASIC hardware designs. By providing such a combination, the reconfigurable computing structure falls at the borders between pure “hardware” and “software” structures. In order to ease the design of such architectures a new plane

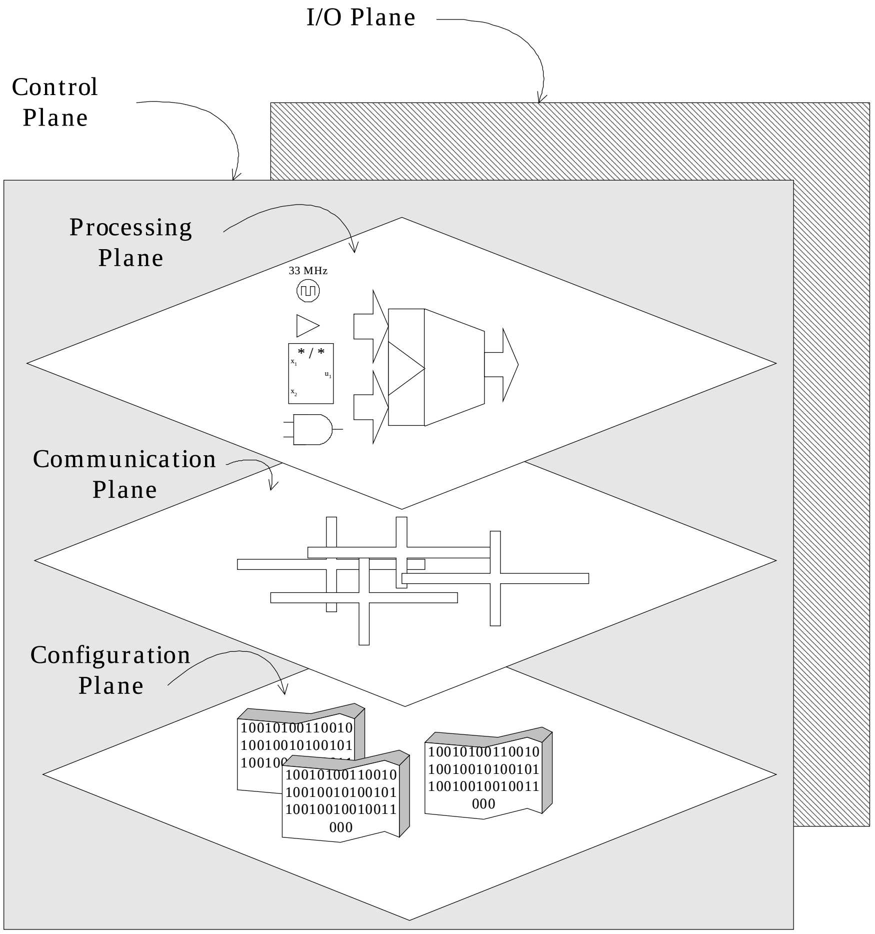


Figure 5-1: An abstract planar model for the dynamically reconfigurable architecture

architecture model has been developed. It is presented and illustrated in Figure 5-1. This model includes processing, communication, and configuration planes interacting with the I/O plane and is controlled by the global control plane.

The processing plane is an array of Processing Elements (PEs) that can be configured from the configuration plane to define discrete logic, registers, mathematical functions, and memory. For any process to be mapped to the processing plane it has to be translated first to the lower level functions. The process is then mapped to the communication network resources between functions, other processes, and/or to the I/O plane. These functions are then translated to primitive tasks. Tasks are defined as:

- Mathematical tasks such as multiplication, spreading, and addition.
- Logical tasks, such as comparison, shifting, logical functions (AND, OR, etc.)
- Data transfer tasks, such as RAM, FIFO, data in, data out.
- Control tasks, such as the start and end of a process, signaling and machine states.

The processing elements are able to perform any one or more of the first three tasks. The PEs are designed to execute all these tasks (and therefore all of the functions) in a very efficient manner.

The performance of any application mapped to the processing plane depends strongly upon the efficiency and speed of the communication plane. The communication plane contains the routing resources needed by the processing plane elements. Commercial FPGAs use static communication resources. Thus the routing resources are allocated at the time of mapping the process to the processing plane. These communication lines are then tied to the process and cannot be used even when the process is idle. Since computation time differs from one process to another and since the data rate (bits/s) also differ from one process to another, it is more efficient to use the routing resources for more than one process. If the communication behavior of a process is known prior to the mapping step, then it is possible to allocate the communication resources only at the required time(s) and then assign the same routing resources to other processes for the remaining time. Since this will result in configuration time overhead, a compromise between less routing resources and communication plane performance must be made.

The configuration plane's main task is to store the configuration bits and load them into the appropriate processing plane location in accordance with the specification of the control plane. This can be achieved rapidly by locating the configuration memory which holds the configuration codes near the targeted PE of the processing plane. Through the global view of the control plane, a new configuration is downloaded to the vacant PE whenever the PE sends a vacant signal to the control plane.

The data flow of the processing plane needs to be interfaced to the outside world. The configuration codes of the configuration plane must be downloaded from outside of the architecture. The I/O plane performs these tasks. Fast I/O units are required in order to overcome the fact that there is only a limited number of I/Os. Since the I/Os are used to input/output data bits and also to input configuration bits, then they must be fast enough to reduce the configuration-time overhead. This requirement of a fast I/O will be elevated when the architecture is integrated in a SoC. In this case the configuration bits are saved in the internal RAM and loaded directly into the architecture.

When designing the PEs, the number of concurrent tasks that can be executed on one PE at the same time has to be weighted against the amount of communication network resources required for those tasks. The number of inputs/ outputs of a PE is another major factor affecting the number of tasks per PE. These factors were carefully analyzed in this dissertation.

A well designed dynamically reconfigurable architecture must take advantage of the data flow nature of the mobile processing. Compared with control-dominated systems, data-flow systems are characterized by the flow of data that passes through one or more processing stages. These stages process the data and move the results on to the next stage. This is similar to vector processing, where a vector of data is processed concurrently in a bit-wise fashion. While DSPs are sequentially processing elements, reconfigurable architectures can perform such vector processsing in parallel. The results are large gains in performance while operating at much lower clock frequencies and less power dissipation.

For instance, to reach the same performance levels on a DSP compared to reconfigurable computing processors, DSPs are required to run at clock rates in excess of 400 MHz. Based on the clock rates and assuming the same power supply and fabrication technology, the savings in power consumption is more than 50% of the DSP's total power consumption. In addition, since these architectures are only programmed (configured) when needed, even more power will be saved.

Flexibility is well supported by DSPs. However, this type of flexibility is more than what is really needed for one selected area of application. The reconfigurable computing architecture is designed to only support the required flexibility within the target application. The ease of programming is still preserved through the proposed IP design and mapping methods. The redundant flexibility of DSP is traded for better performance in the DRAW architecture. The IP design methods would be very similar to programming a DSP. Therefore, a new set of algorithms (IP) can be designed and compiled for the target architecture.

5.3 Dynamically Reconfigurable and Parallel Array Architecture

DRAW consists of an array of parallel operating coarse-grained Dynamically Reconfigurable Processing Units (DRPUs). Each DRPU is designed to execute all of the required arithmetic data manipulations for data-flow oriented mobile applications, as well as support necessary control-flow oriented operations. The complete DRAW array architecture connects all DRPUs with reconfigurable local and global communication structures; see Figure 5-2. The architecture provides an efficient and fast dynamic reconfiguration of DRPUs and interconnection structures, reconfiguring parts of architecture during run-time, while other parts are active.

The decisions during the design of the architecture are based mainly on careful reviewing of the tailored application area requirements. We began the bottom-up design approach of DRAW by focusing on a list of the most complex and flexibility-demanding application parts of future mobile receivers, e.g., RAKE-receiving parts, filtering, searcher algorithms, turbo coding, and maximal ratio combining techniques. Based upon the set of required arithmetic and control-flow operations, the performance/power optimized structure of the DRPUs and local communication units called Communication Switching Units (CSUs) was devised.

As shown in Figure 5-2 the DRAW architecture consists of a scalable array of DRPUs that have 16-bit fast direct local connections between neighboring

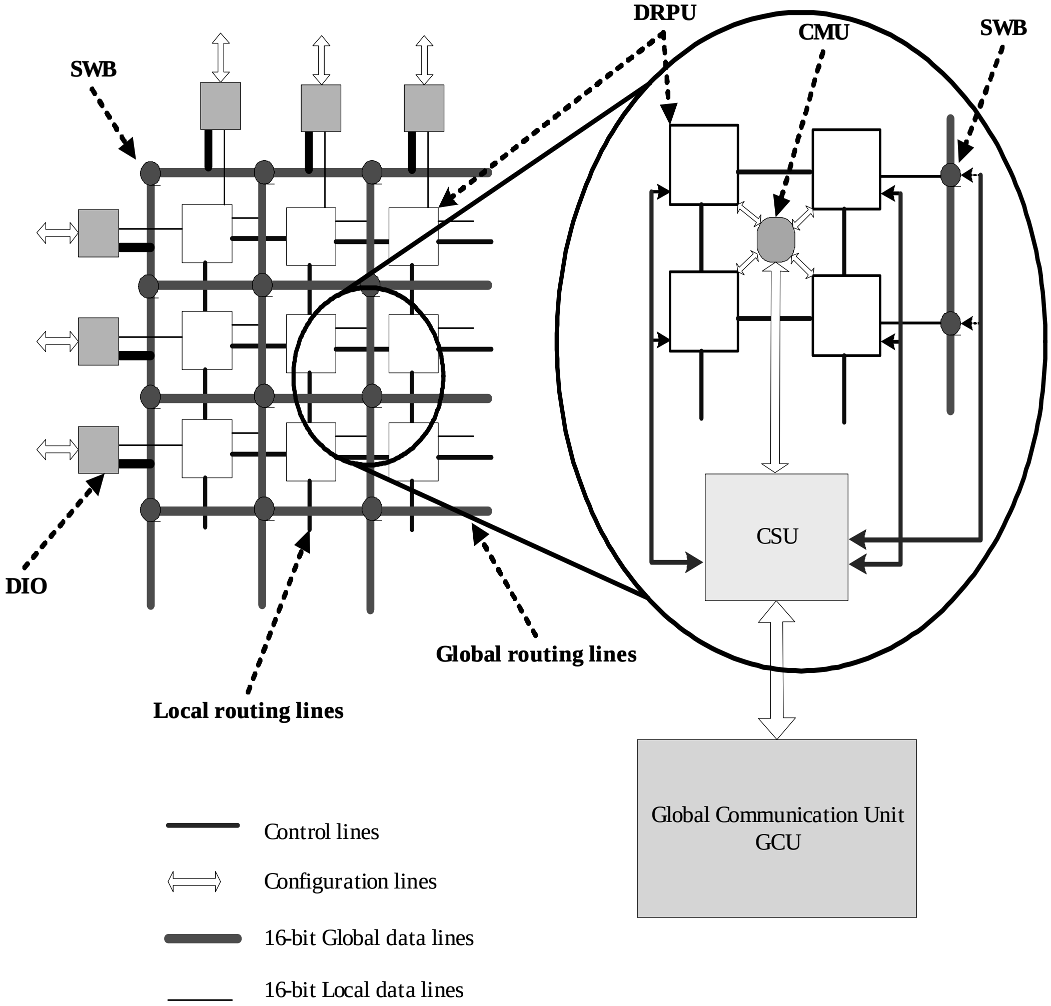


Figure 5-2: A block diagram of the DRAW architecture

DRPUs. Every four DRPU sub-array shares one Configuration Memory Unit (CMU). The CMU holds configuration data for performing fast dynamic reconfiguration for every four DRPUs and is controlled by the nearest CSU. Each CSU controls two CMUs and four global interconnecting Switching Boxes (SWB).

Dedicated I/O Units (DIOs) for fast and parallel transfers of the input/output data of DRAW are placed around the DRPU array. Each DIO can be connected either to one DRPU at the border of the DRAW architecture, and/or to DRPUs inside the array through the global interconnect lines. Every DIO is able to perform the internal local and global DRAW communication protocols, as well as the interfacing functionality to the other components of the SoC, e.g., the DSP, the micro controller, and the on-chip memories.

All CSUs communicate with one Global Communication Unit (GCU), which coordinates all dynamic reconfiguration steps of the DRPUs and the global interconnection structure. Moreover, the GCU controls the external communication with other hardware components of the SoC. The GCU triggers the responsible CSUs to initiate the controllers of the corresponding CMUs for loading a new configuration to the selected DRPUs. This fast transfer process of reconfiguration code is then completed by an efficient burst-mode protocol between the CMU- and the DRPU-controllers. The dynamic reconfiguration algorithm for various scenarios in future mobile communication systems is performed by the GCU. This algorithm decides which parts of the array have to be reconfigured with which particular configuration data which are stored either in the CMUs or in an external memory.

5.3.1 Regular and Simple Array Structure

The DRAW architecture is designed to be a target implementation for a higher level IP-based mapping tool. Making the processing elements and the routing resources of the architecture simple, reduces the efforts of the IP-mapper to reach an optimal mapping structure. To be able to integrate the reconfigurable architecture in a SoC, the architecture must be sizable, since different SoC solutions require different sizes of the architecture. Additionally, as new fabrication technology with smaller feature sizes become available, the architecture can be scaled down. Scaling the architecture will only require the scaling of one unit of the design, which is then repeated for the whole architecture.

5.3.2 Special Processing Units

Different wireless mobile receiver functions mandate different processing data-path widths. Although, most of the functions will require 8-bit or more data-path widths, some functions require only one-bit data-path processing. Implementing such a function on a data-path that is wider than a one-bit width is a waste of resources.

Three particular functions that don't require more than one-bit processing are spreading/despreading, code generation, and one-bit data coding. The spreading of a data stream involves multiplying data by the PN code. Despreading is done in a similar manner. Implementing this function in the 16-bit DRAP is not useful, since it will waste the DRAP resources.

For this reason, two special processing units are built in the DRPU to handle these special processing requirements. These units are a Configurable Linear Feedback Shift Register (CLFSR) and a Configurable Spreading Data Path (CSDP).

5.3.2.1 Configurable Linear Feedback Shift Register (CLFSR)

Code generation is one bit function in the receiver. In the third generation wireless mobile system receiver standard, PN and gold codes are used for cell and user scrambling. The PN and gold codes are generated by a linear feedback shift register. The linear feedback shift register is an N stage shift register, where the feedback bit is the result of XOR operation on selected bits from the register. The selection is based on the equivalent prime polynomial of the shift register. The Turbo coder consists of two three-stage linear feedback shift registers with an internal interleaving before the second register. The coding is done on the bit level of the incoming data.

A dedicated unit is available in the DRPU that can handle the code generation and can be used for turbo coding when combined with an interleaver. The CLFSR is shown in Figure 5-3. As shown in the figure, the CLFSR can be configured to be either a two or three stage shift register. The unit can be connected to the neighboring units to create longer shift registers. Additionally, the unit can be configured to either generate code or to encode a stream of bits. The polynomial of the shift register sets the configuration bits.

5.3.2.2 Configurable Spreading Data Path (CSDP)

One CSDP for execution of spread spectrum-based spreading tasks is designed and implemented in each DRPU. This CSDP unit can be used together with the add operation of a DRAP to efficiently implement a fast complex PN-code correlation. Such spreading and de-spreading operations are often required in the

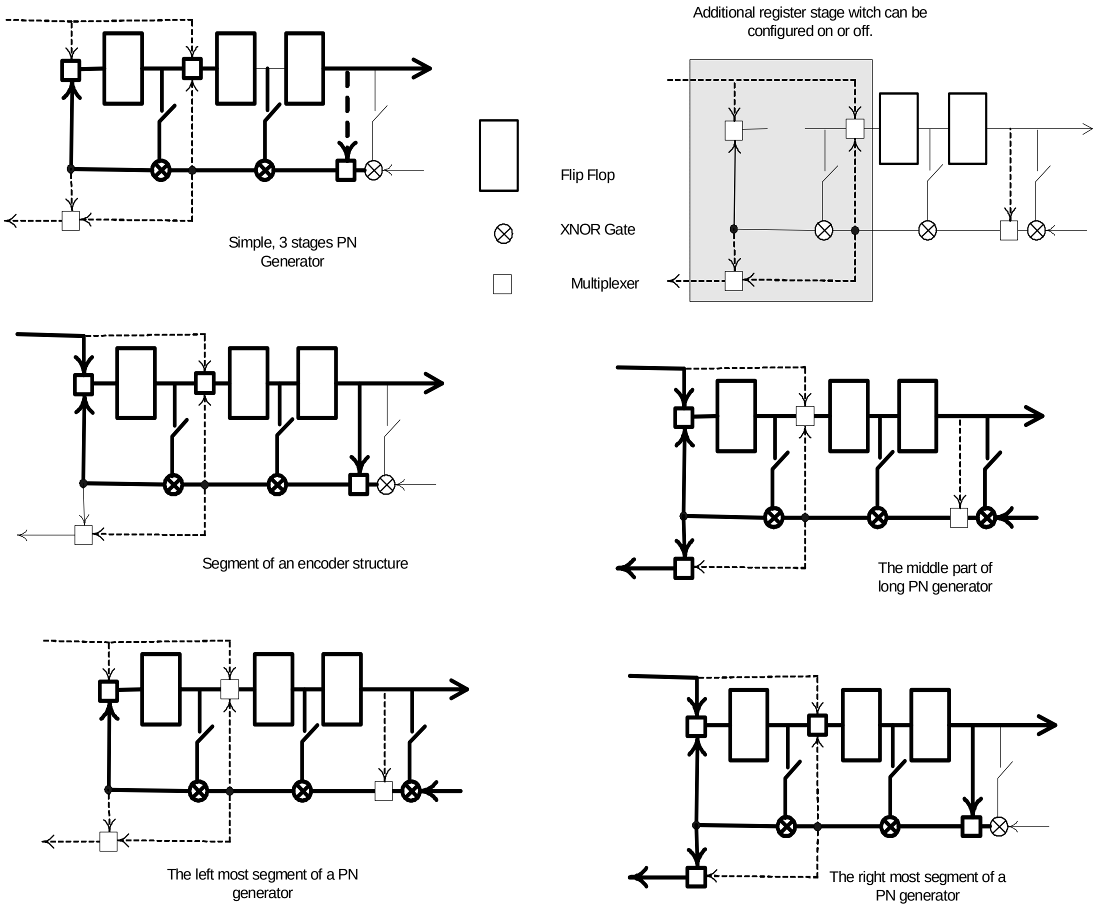
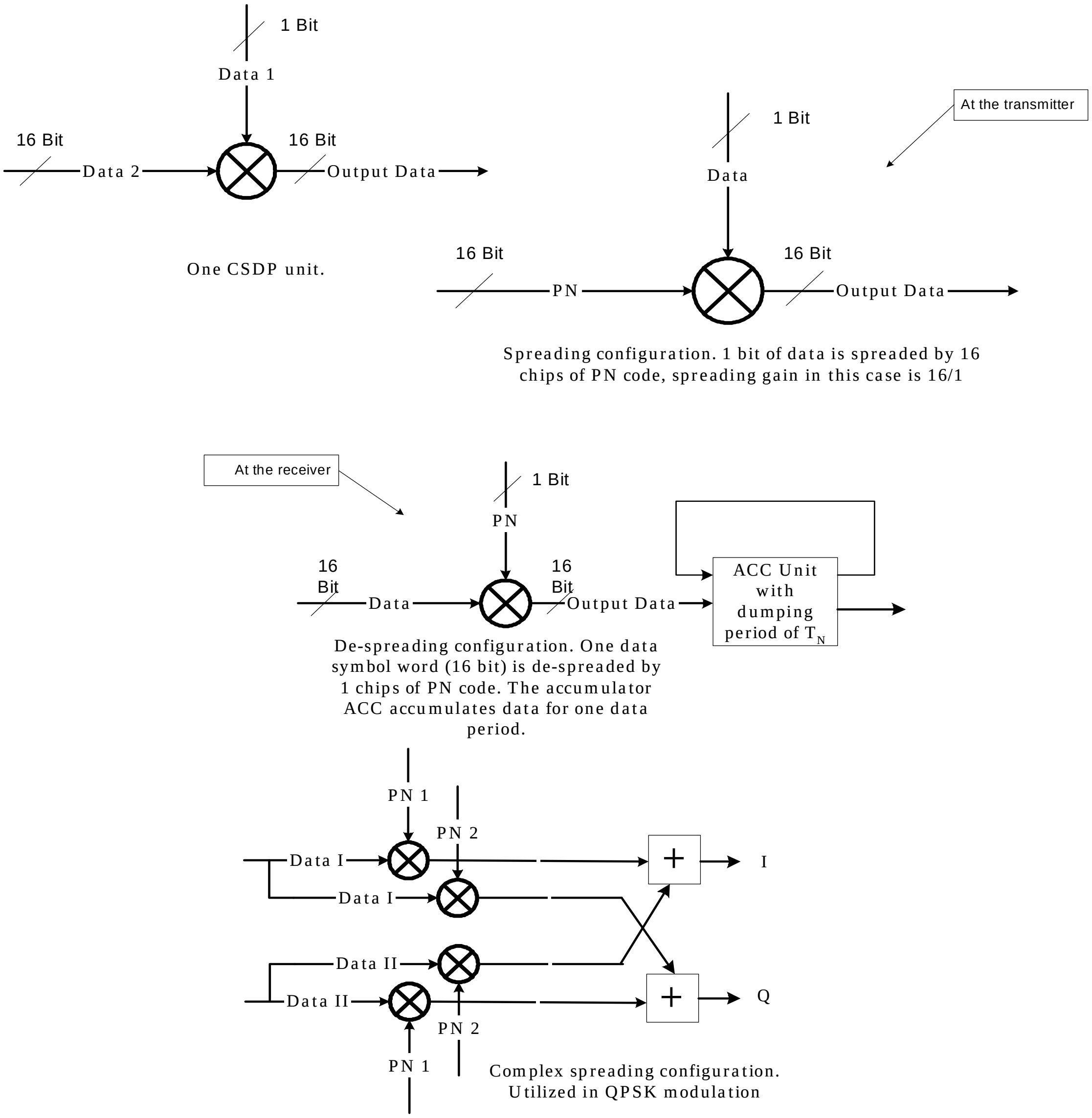

Additional register stage witch can be configured on or off.
Flip Flop
XNOR Gate
Multiplexer
Simple, 3 stages PN Generator
Segment of an encoder structure
The middle part of long PN generator
The left most segment of a PN generator
The right most segment of a PN generator

Figure 5-3: Different possible configurations of the configurable linear feedback shift register

Quadrature Phase Shift Keying (QPSK) modulated systems commonly found in mobile systems.

The CSDP can be configured into many different useful configurations. In addition, it can be configured with other components in the DRPU to implement de-spreading. Some possible configuration structures are shown in Figure 5-4.



One CSDP unit.

Spreading configuration. 1 bit of data is spreaded by 16 chips of PN code, spreading gain in this case is 16/1

De-spreading configuration. One data symbol word (16 bit) is de-spreaded by 1 chips of PN code. The accumulator ACC accumulates data for one data period.

Complex spreading configuration. Utilized in QPSK modulation

Figure 5-4: Some configuration structures of the CSDP

5.3.3 RAM and FIFO Storage

Many functions in the receiver need to store intermediate data during processing. The RAM unit in the RPU can be configured to act as either RAM or as a FIFO. When configured to act as a RAM, the data input, Read and Write addresses are all supplied through the input interface. The RAM writes the incoming data to the provided address whenever there is a new value on the data line. Additionally, the RAM sends the appropriate data selected by the read address whenever the address value changes. This style of implementing RAM eliminates the need to control the RAM to write or read.

When the RAM/FIFO is configured as a FIFO, it requires some control. The FIFO provides two flags to the control unit. The flags are FIFO-full and FIFO-empty. The control unit in the DRPU sends a hold signal to the sending DRPU when the FIFO is full and disables reading whenever the FIFO is empty.

5.3.4 Scale and Delay

Scaling is a very important operation for keeping the intermediate results values under the available maximum data path width. In the DRAW architecture, a scaling feature is built in the DRPU output interface. Scaling is accomplished by shifting the data to the right by a variable or fixed number of bits. The scaling factor is provided either through the configuration bits or through the input interface. When the scaling value is static, it is easier to set it up through the configuration bits, and when the scaling value depends on some variables, it is generated by another DRPU and then passed to the scaling DRPU.

5.3.5 Communication Network

For fast data processing, fast connection between the DRPUs is necessary. The communication between different DRPUs has two hierarchy levels. At the lower level, local communication between neighboring DRPUs provide a fast connection channel, and at the upper level, global communication channels can be utilized by the communicating DRPUs.

The communication between two DRPUs can be either deterministic or indeterministic. If the communication between two DRPUs is deterministic, then the knowledge of this communication pattern can be utilized at the design compilation level. The designer can implement the sequence of the SWB's configuration bits to follow the communication behavior. This type of communication should be utilized whenever available in the implemented process. Many processes of the 3G receiver can be deterministic, thus this feature would benefit the architecture.

For an example of the deterministic communication, assume two RAKE fingers implemented on four DRPUs and two DRPUs for each finger. Assume also a spreading gain of 64, which means that the dumping period of each finger is every 64 chips long. The example is illustrated in Figure 5-5. Since one RAKE finger is delayed by at least one chip then the two fingers will not use the communication channel at the same time. By configuring the connection points to switch on and off at the required time, the two fingers can share the communication channel without any conflicts.

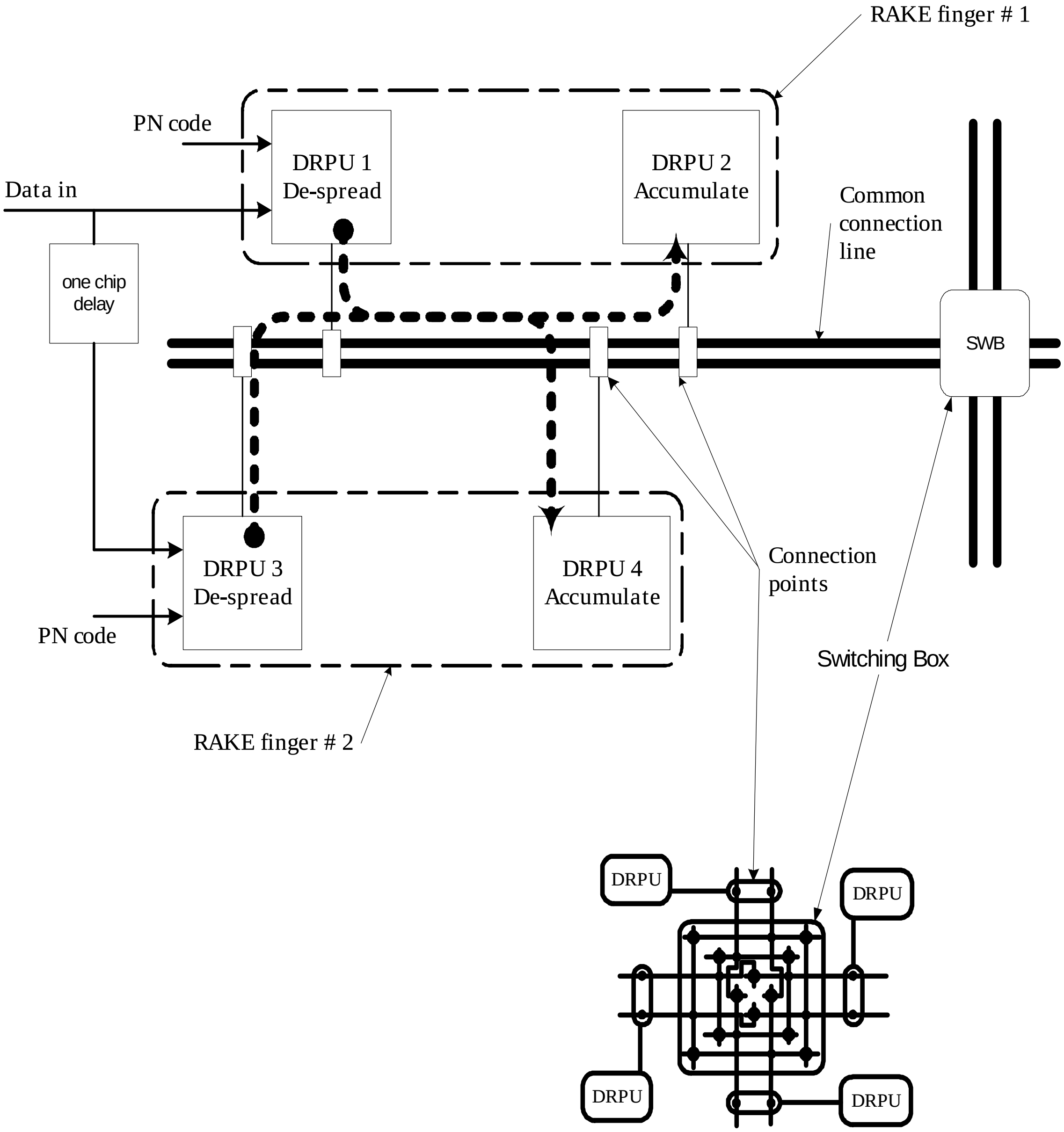


Figure 5-5: An example of a deterministic communication between four DRPUs

Unfortunately, not all communication between DRPUs is deterministic. When the communication between DRPUs is not deterministic, a start-hold mechanism is used to synchronize the communication between the communicating

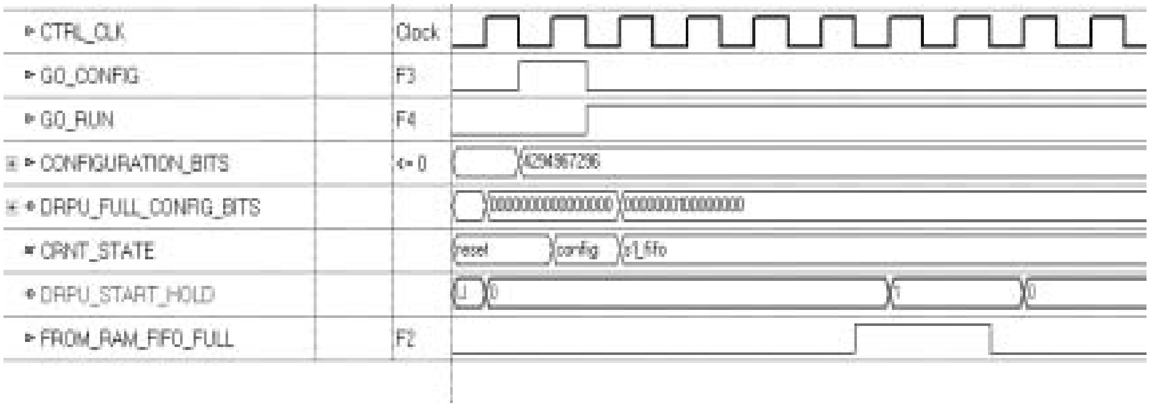


Figure 5-6: A VHDL simulation of the start-hold mechanism of the DRPU

DRPUs. Every DRPU communicating with its four neighbors uses the start-hold signal to indicate when it is ready to receive data.

The output of a VHDL simulation for the start-hold mechanism is shown in Figure 5-6. The figure shows an DRPU that is first configured as a FIFO, then while the DRPU is running, the FIFO becomes full and a FIFO full signal is asserted. Once the FIFO full signal is asserted, the DRPU controller asserts the start-hold signal. Since the start-hold signal is routed to the sending DRPU, the sender will go into hold. As the FIFO becomes empty the FIFO full signal is de-asserted and the DRPU controller de-asserts the start-hold signal.

5.3.6 Dedicated I/O

The input/output pins can be the limiting factor for some applications. We designed the core of the architecture to run at lower levels of clock signal, then, whenever an application requires a high throughput rate, the I/O pads must cope with this rate by distributing (or collecting) data to (from) one or more destinations (sources).

The DIO unit of the DRAW architecture is designed to provide either registered or un-registered input/output functionality. Additionally, the input/output unit drives up to four destinations/sources global lines as shown in Figure 5-7. The selection of the destination/source global lines is either done by the configuration bits or by another DRPU. The DRPU that is the nearest to the DIO can be used to select which global line is connected to the DIO.

5.3.7 Fast Dynamic Reconfiguration

To enable the DRPUs and SWBs to be quickly reconfigured, the configuration bits are loaded in the configuration memory unit prior to the reconfiguration process. This concept is similar to cashing the configuration bits before loading them into the DRPU or the SWB. A closer look at configuration parts from Figure 5-2 is shown in Figure 5-8. The CMU holds up to four configuration sets. The CMU is connected to four DRPUs. Each DRPU can read any one of the four configuration bits available in the CMU.

The communication switching unit (CSU) collects eight (8-bit words) configuration bits stream and then loads this 64-bit configuration into the lower 64 bit configuration register of the CMU. This process may be repeated until four configuration sets are loaded in the CMU. The CSU then receives control signals from the Global Communication Unit (GCU) which selects one of the four configurations residing in the CMU and also selects which DRPU is to be configured. The CSU complete the configuration process by loading the specified configuration set into the targeted DRPU.

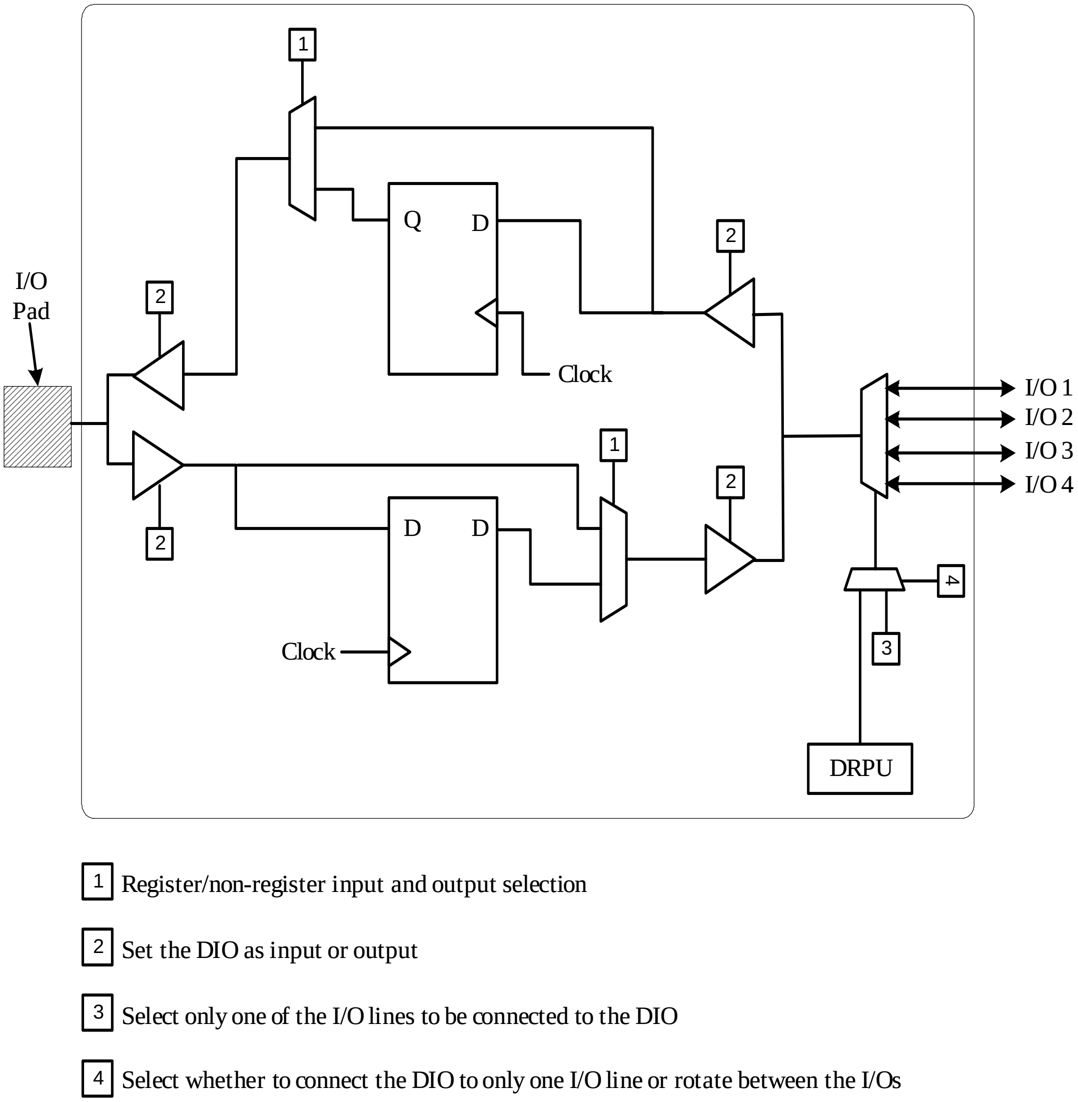

1
Q
D
2
I/O
Pad
2
Clock
I/O 1
I/O 2
I/O 3
I/O 4
1
2
D
D
2
4
Clock
3
DRPU
1 Register/non-register input and output selection
2 Set the DIO as input or output
3 Select only one of the I/O lines to be connected to the DIO
4 Select whether to connect the DIO to only one I/O line or rotate between the I/Os

Figure 5-7: A block diagram of the dedicated I/O unit

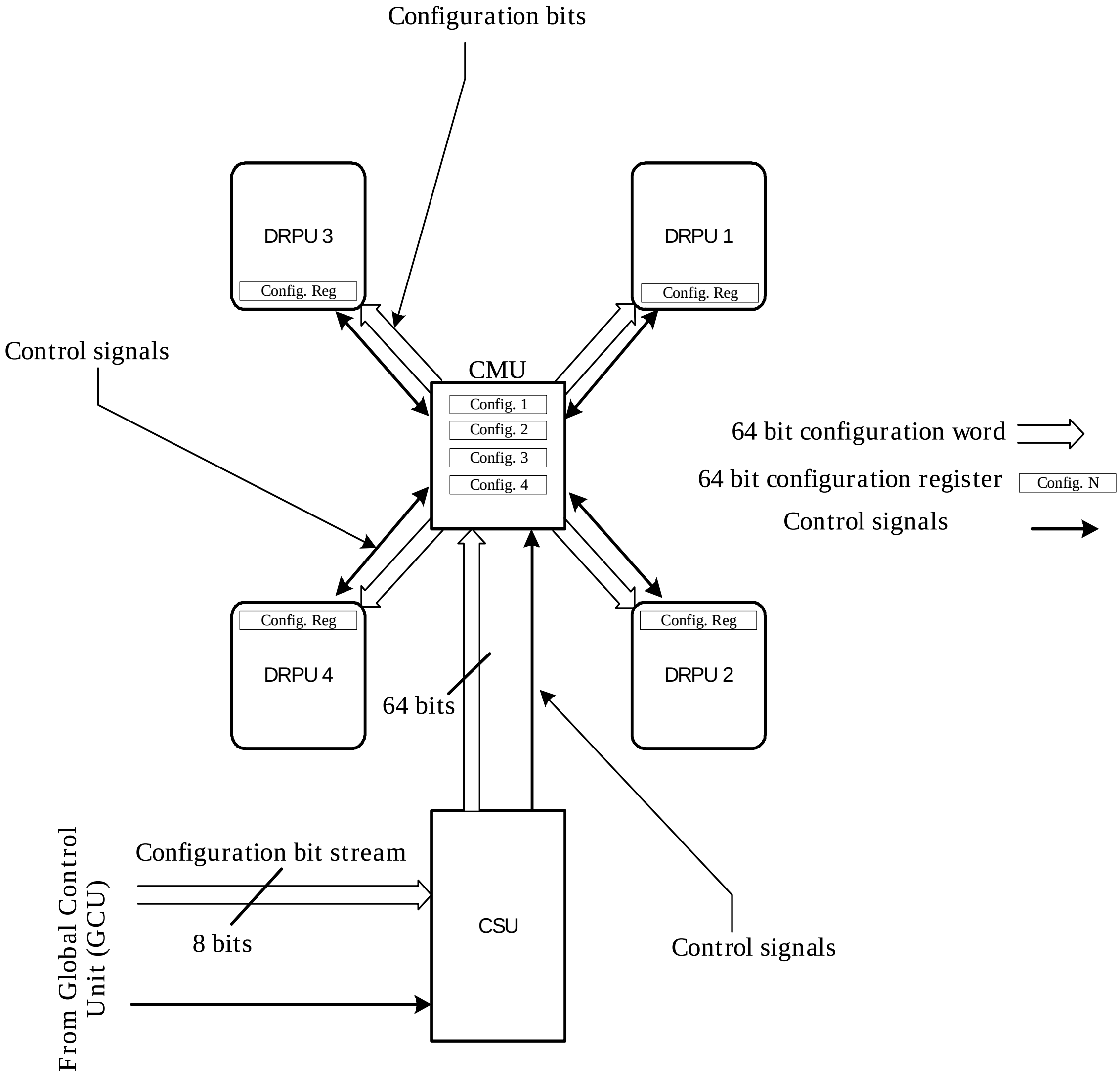


Figure 5-8: The configuration level structure for fast reconfiguration mechanism

Chapter 6

Dynamically Reconfigurable Architecture Design

6.1 Introduction

The goal of this chapter is to describe the design and the development of the DRAW architecture. The design of the DRAW architecture started by profiling the five most demanding operations of the baseband unit of the wireless mobile terminal. A description of the profiling steps will be presented. The hardware components of the architecture are described in detail in this chapter.

6.2 Design Goals

Flexibility is the primary design goal for the architecture. Flexibility is the ability of the architecture to execute a large number of functions from the wireless applications. Designing for flexibility is a complicated task. A simplification of the tasks is introduced by applying some limitations and assumptions. These assumptions will be declared when each unit design is described.

A high computation rate is the second design goal. The wireless mobile receiver requires a real time implementation. The design of the DRAW architecture is considered together with the timing of the data received in the 3G system standard.

Enabling the architecture to work with an IP-based mapping tool requires the architecture to be regular. Regularity is exploited in the processing plane and in the communication plane.

Having a fast dynamic reconfiguration is expected to produce the success of the architecture as an efficient implementation platform. The fast reconfiguration mechanism must be implemented on the upper two levels of the architecture that is on the processing and the communication planes.

6.3 Design Process Flow

Figure 6-1 shows the flow of the design process of the DRAW architecture. The design process began by simulating in MATLAB the most demanding functions the third generation receiver. This step involves reviewing in detail the 3GPP standard. Based on the standard, I wrote a set of MATLAB functions to implement the complete front-end receiver (see Appendix A).

The 3GPP standards have been published in over 3,000 documents which cover all aspects of the WCDMA system. The first step in reviewing the standard is to locate only the documents that are related to the implementation of the receiver. For the sake of simulating a complete communication system, the trans-

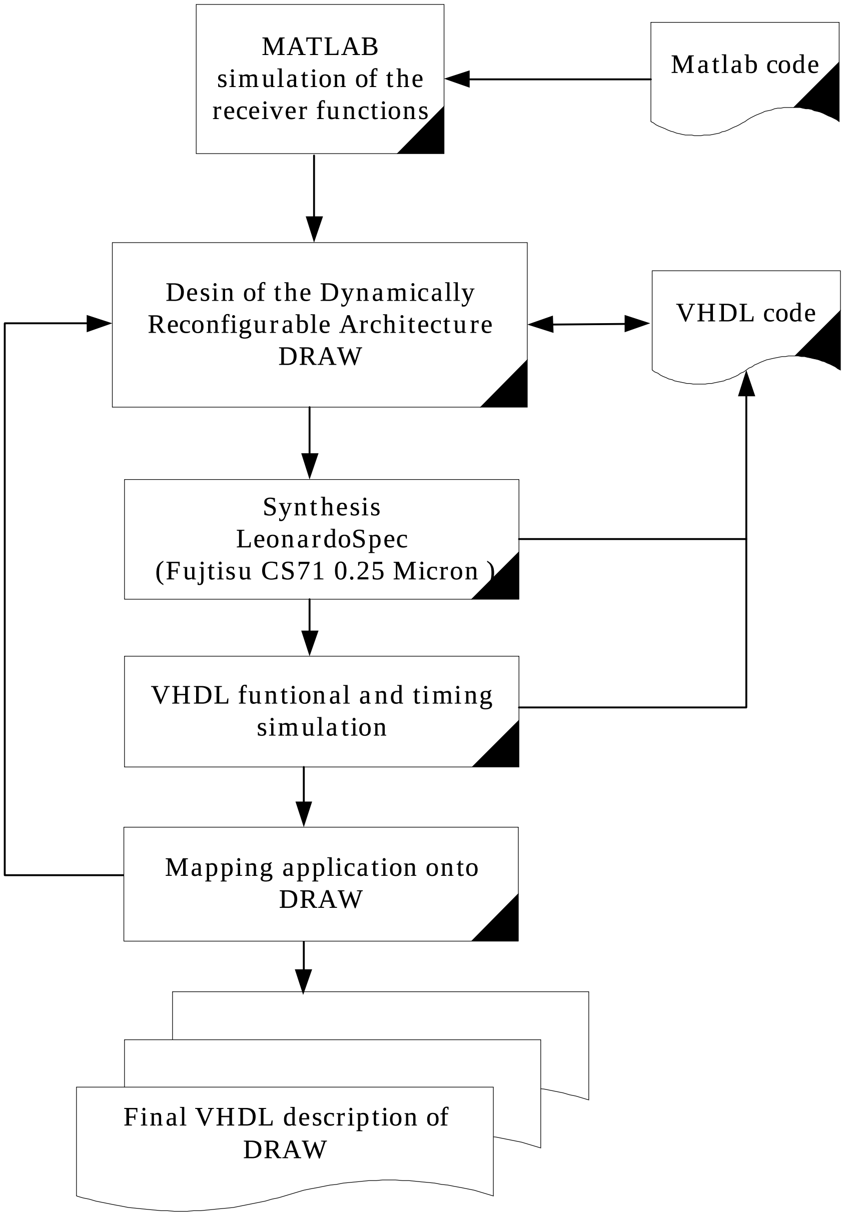

MATLAB simulation of the receiver functions
Matlab code
Desin of the Dynamically Reconfigurable Architecture DRAW
VHDL code
Synthesis LeonardoSpec (Fujtisu CS71 0.25 Micron)
VHDL funtional and timing simulation
Mapping application onto DRAW
Final VHDL description of DRAW

Figure 6-1: Design flow process of the DRAW architecture

mitter specifications are also needed, which requires locating the documents related to the transmitter. The 3GPP standard documentation can be obtained from the 3GPP web site *www.3gpp.org*.

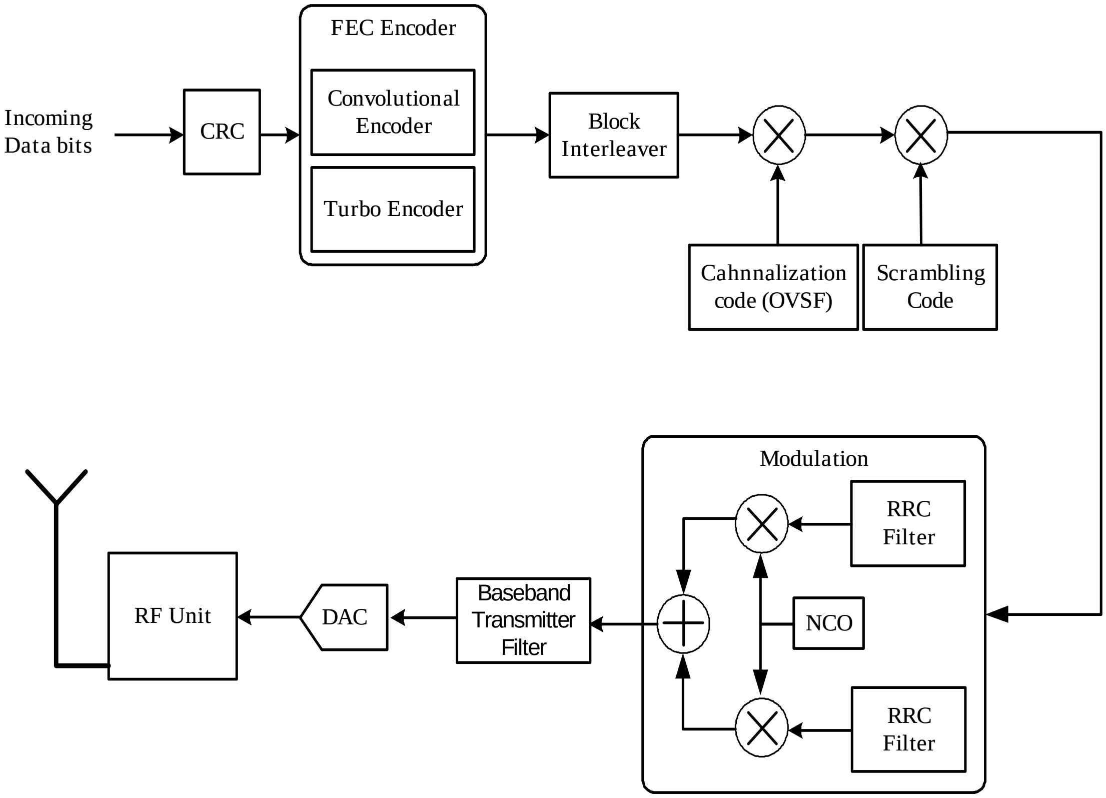


Figure 6-2: Block diagram of a transmitter which supports 3GPP standard

The transmitter architecture that supports the 3GPP standard is shown in Figure 6-2. According to the standard, the incoming user data bits are first augmented with Cyclic Redundancy Check (CRC) bits. The CRC bits are added for the purpose of error detection. The standard specifies four different polynomials for the CRC checking:

$$g_{CRC24}(D) = D^{24} + D^{23} + D^{6} + D^{5} + D + 1 \qquad \textbf{(6-1)}$$

$$g_{CRC16}(D) = D^{16} + D^{12} + D^{5} + 1 \quad \textbf{(6-2)}$$

$$g_{CRC12}(D) = D^{12} + D^{11} + D^{3} + D^{2} + D + 1 \quad \textbf{(6-3)}$$

$$g_{CRC8}(D) = D^{8} + D^{7} + D^{4} + D^{3} + D + 1 \quad \textbf{(6-4)}$$

Forward Error Correction (FEC) bits are then added. The FEC bits are either generated by a convolutional or turbo encoder. Determining wether to use a convolutional or turbo encoder depends upon the type of service that is required. For example, for a high quality service with a BER of 10^{-6} a turbo coding is used for the FEC. If a service requires a BER of 10^{-3} then a convolutional coding is used. For the turbo encoder the standard specification are:

- A block length of 40 to 5114 bits long.
- A Parallel Concatenated Convolutional Code (PCCC) with an 8-state constitute encoder and an internal interleaver.
- An unpunctured rate of 1/3 or a punctured rate of 1/2.

Whereas for a convolutional encoder the standard specifications are:

- The downlink’s convolutional encoder constraint length is K= 9 with rates of 1/2 or 1/3.
- For the uplink K= 9 and rate = 1/3.

A block interleaver is then applied to the data to spread the burst errors incurred in the communication channel. Orthogonal Variable Spreading Factor (OVSF) codes specific to every channel are then used to spread the data. The OVSF codes are a Walsh code that is generated from the Hadmard structure:

$$H_0 = \begin{bmatrix}1\end{bmatrix} \qquad \textbf{(6-5)}$$

$$H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix} \qquad where \qquad (n \geq 1) \qquad \textbf{(6-6)}$$

The spreaded data is then scrambled with a Gold code that is specific to the communication cell. The main advantage of using the scrambling code is that it gives the receiver the ability to recover all the different paths of the same signal. This is because the scrambling codes reduce the auto-correlation between the different time delayed versions of the transmitted signal. The scrambling codes are specified in the 3GPP standard as follows:

- A Gold code segment of 38,400 chips long from a 2^{18}-1 chip long Gold code. The Gold code generator is shown in Figure 6-3
- 512 different scrambling codes. The 512 codes are grouped into 32 groups of 16 codes each in order to conduct faster cell search.

The data is then modulated by first shaping the data with a Root Raised Cosine (RRC) filter that has a roll off factor of 0.22. The data is then modulated to the carrier frequency and filtered by the baseband filter before being converted to analog and sent to the RF unit.

A receiver block diagram that supports a 3GPP standard is shown in Figure 6-4. The demodulation is performed in a fashion similar to the modulation in the transmission process. A Multi-path diversity RAKE receiver or other suitable receiver structure with maximum combining is specified in the 3GPP standard.

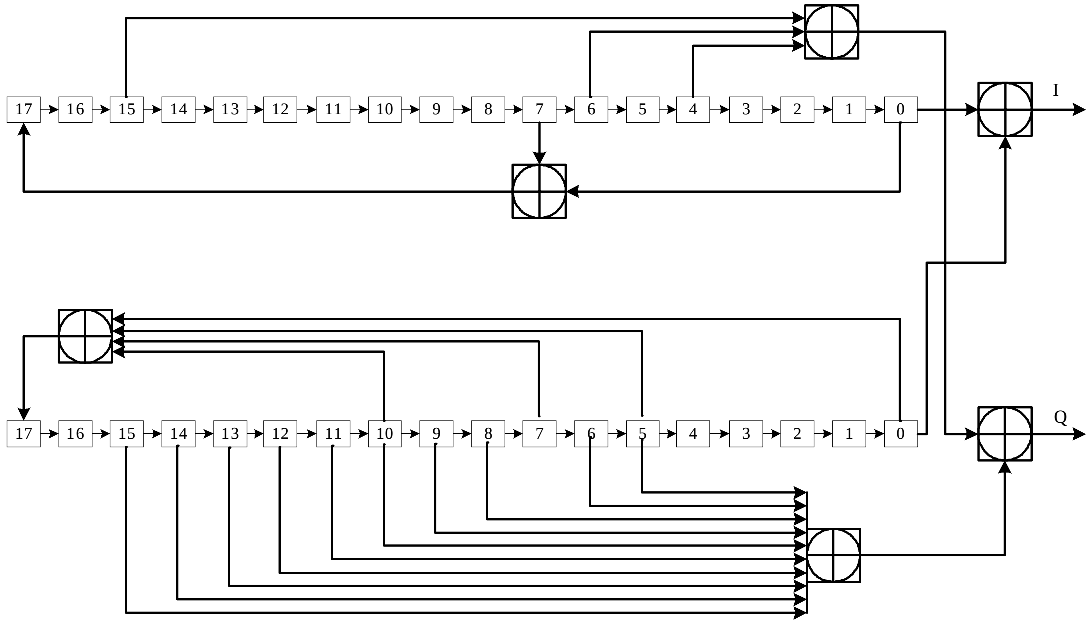


Figure 6-3: 3GPP downlink scrambling code generator

Despreading of the received signal can be implemented using the despreading circuit shown in Figure 6-5 [33]. In the 3GPP standard, data modulation is QPSK in the downlink as seen in Table 2-3 in Chapter 2 on page 21.

The multipath estimation block estimates two variables for every path in the channel. The two variables are the delay and the value of the amplitude attenuation. Traditionally, the implementation of the multipath estimation unit is done in two steps. First the candidate paths are located, then a Delay-Locked Loop (DLL) is used to track the path. The first step in the implementation utilizes a sliding correlators that calculates one delay tap power per dumping period. The more parallel correlators are available, the faster the incoming data can be scanned. Since this implementation requires the use of a DLL, a control unit is needed to control the DLL to keep track of the paths. This makes the implementing of the

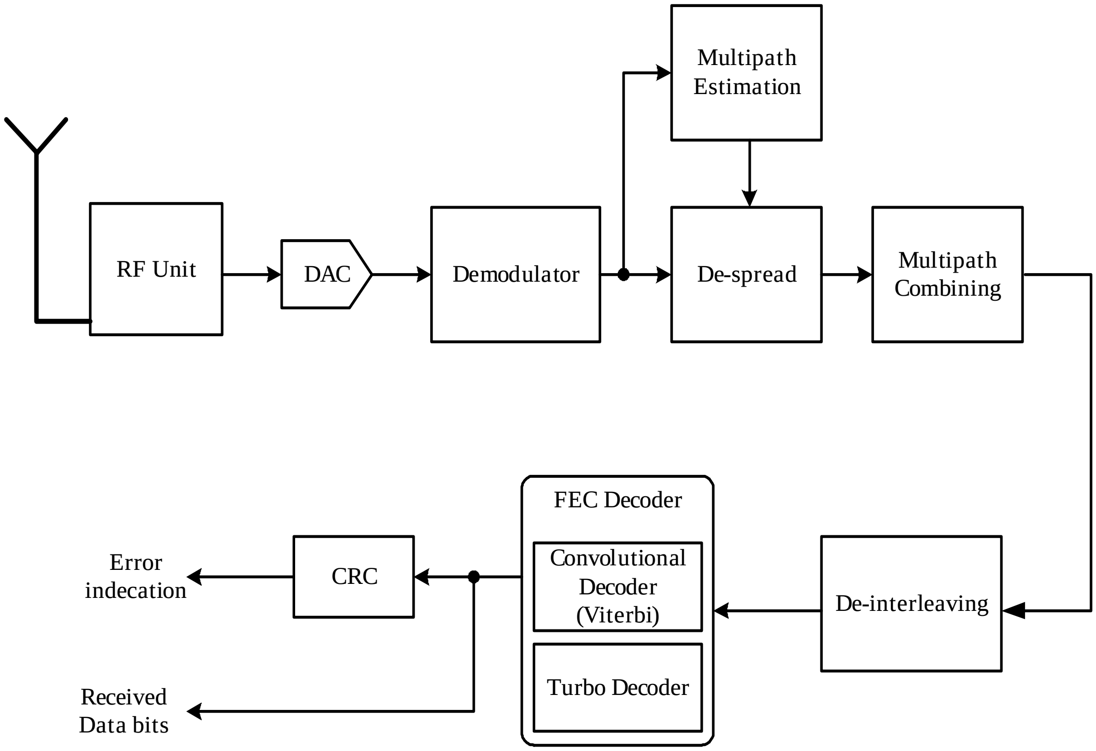
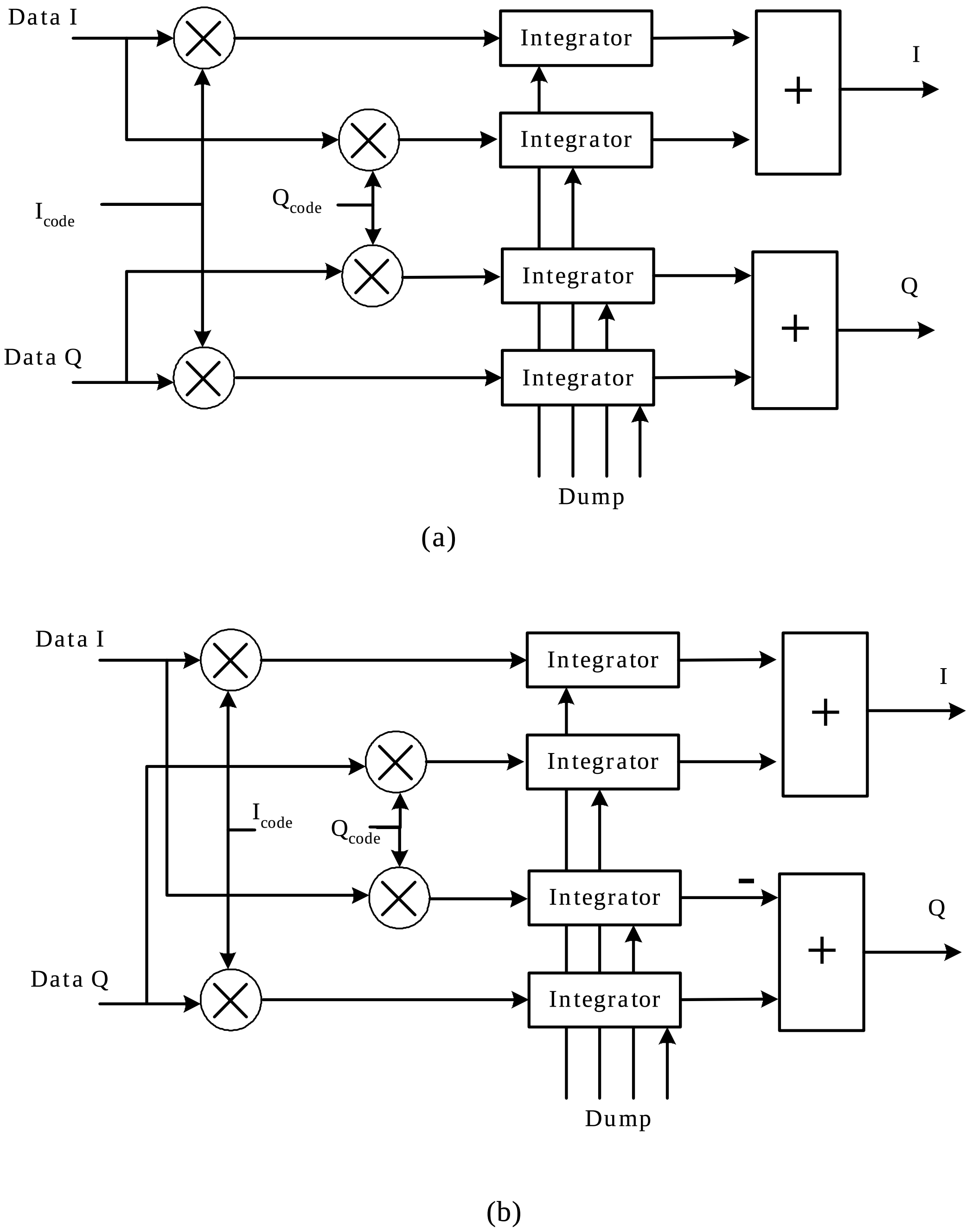


Figure 6-4: Block diagram of a receiver baseband which supports 3GPP standard

multipath estimation undesirable. A better approach is to use a matched filter. The matched filter correlates the incoming data with the user code in different delay phases. Thus the matched filter gives a new tap value every dumping period. The advantage of the matched filter over the DLL approach is the elimination of the control unit that was needed for the DLL.

The target of a multipath combining unit is to combine different RAKE fingers outputs into one signal with the SNR higher than that of the individual signals from the RAKE fingers. The Maximal Ratio Combiner (MRC) method provides the best performance for this purpose. The MRC combines the different signals according to their SNR.



(a)

(b)

Figure 6-5: Despreading circuit for QPSK chip modulated signal. (a) Dual-channel despreading and (b) complex despreading

The MATLAB implementation and simulation of the complete baseband receiver is reported in [27]. A simulation of the RAKE receiver and the turbo decoder are reported in [61] and [71] respectively.

A statistical profile of the function is extracted from the MATLAB simulation. This profile contains information about the operation type, number of operations, interconnections, and I/O requirements of the most demanding functions.

Based on the MATLAB profiling and guided by the design goals, the architecture components in VHDL were designed (see Appendix B). The VHDL codes generated are synthesizeable behavioral descriptions. Once a complete set of VHDL codes were written, I ran a functional simulation. The synthesis was done on the Leonardo Spectrum synthesis tool targeting Fujitsu CS71 standard cell ASIC library. The Fujitsu CS71 is a five-metal layer 0.25 μm CMOS fabrication process which offers up to 10 million gates per chip. In case the functional or the timing simulation were not correct, the design was refined and re-simulated. The next step was to map the targeted function of the receiver. If the required performance of these functions is reached, then this would be the final version of the architecture. Otherwise more refinement must be necessary until all of the requirements are met.

6.4 Operation Profile

A description of the MATLAB simulation results of the most demanding functions of the third generation receiver will be given in this section. The five most demanding functions of the wireless mobile receiver are:

- FIR Filter
- RAKE receiver
- Maximal Ratio combining
- Turbo coder
- Searcher

A MATLAB code was written for each function (see Appendix A). An organizational diagram of the MATLAB simulation functions is shown in Figure 6-6. The MATLAB code of all the functions on the figure are given in Appendix A. Statistical information regarding the number and type of operations, and the interconnection information were extracted.

The simulation of the MATLAB code was performed on one RF slot data. Figure 6-7 Shows the layout of one slot of the WCDMA standard. One slot data was used as the processing data for the MATLAB simulations. Figure 6-7 shows the spreading factor is set to the minimum. In the 3GPP standard, the spreading factor ranges from 4 to 512. For the highest data rate in a channel (maximum bits per slot) the lowest spreaded is applied. In the simulation process, the highest data rate communication channel is assumed. The data rate of the channel shown in Figure 6-7 can be calculated as follows: the 3gpp standard states that a maximum of 1,280 bits can be packed in one slot. One frame lasts for 10 ms. One frame is constructed from 15 slots. The data (1,280 bits) are spreaded by a user specific code with a spreading gain of 4, i.e. each bit is multiplied by four chips of the user code.

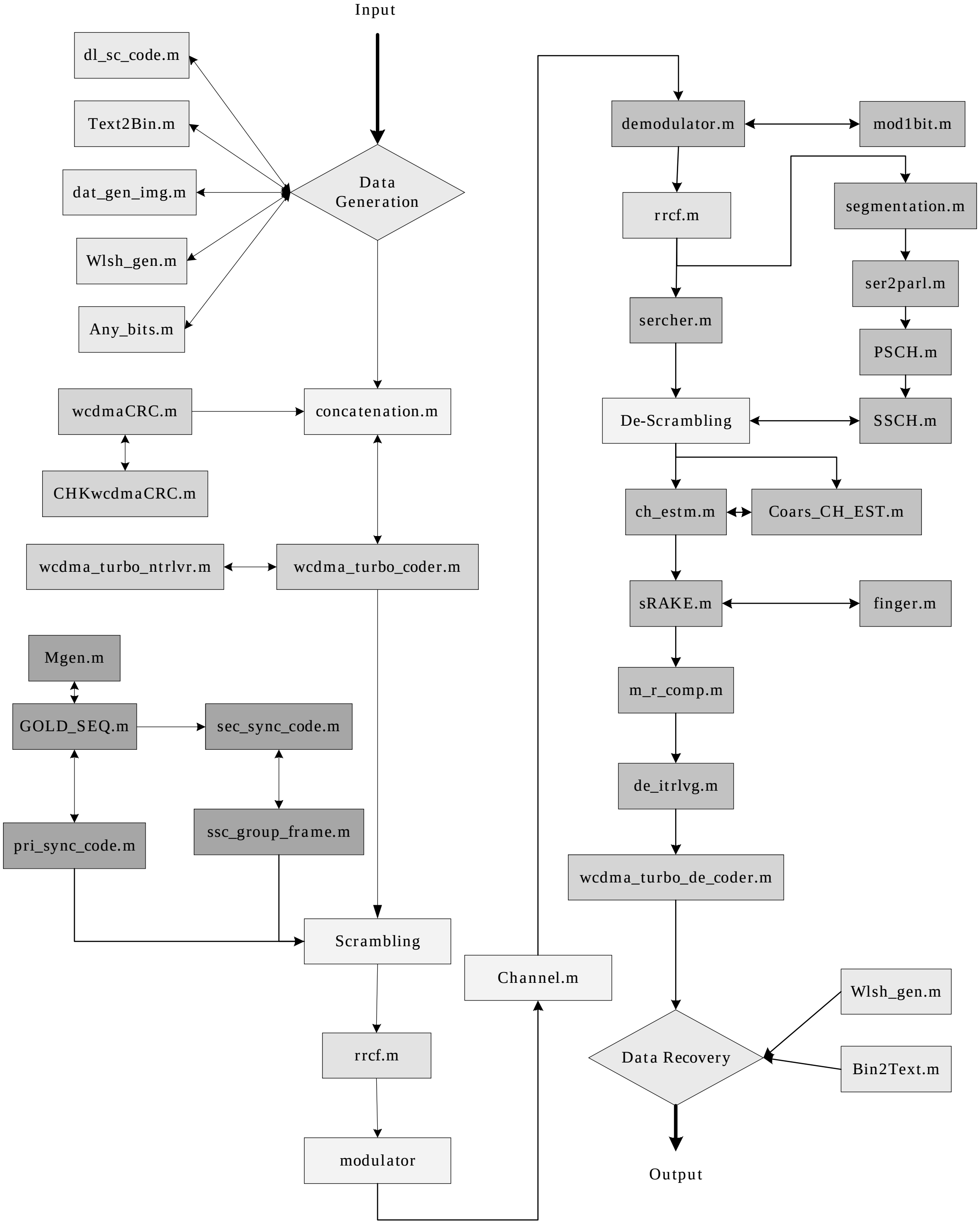
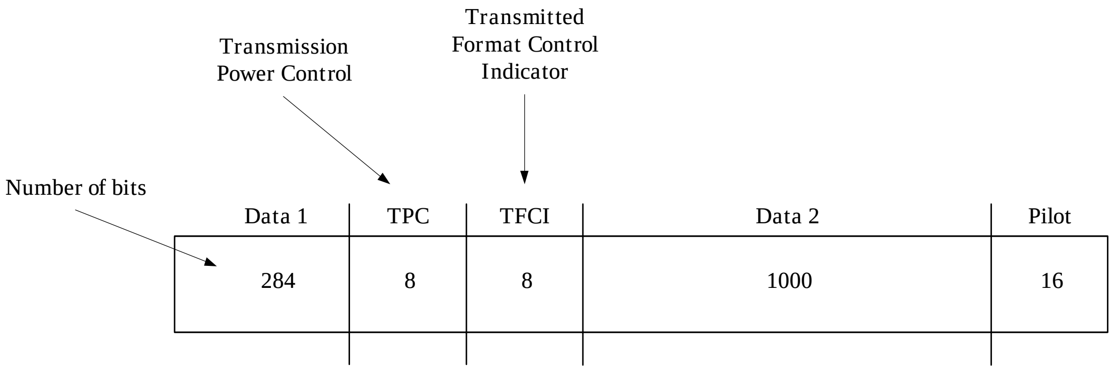


Figure 6-6: Matlab simulation environment functions



Total bits = 1,280
Spreading factor = 4
Total number of chips = 1,280 x 4 = 5,120 chips/slot
Channel rate is 1,280 x 15 slots = 1,920 Kbps

Figure 6-7: Slot structure of 3GPP standard for WCDMA

This results in 1280*4 = 5120 Chips per slot. So the data rate of the channel is 5120 Chips/slot * 15 slots / 10 ms =1.920 Mchip/s.

Figure 6-8 shows a count of the number of operations versus the operation type in the five different applications. The assumption made for the RAKE, maximal ratio combining, searcher, and FIR filter is that they operate on the chip rate. While the turbo decoder operate on the symbol rate. Additionally, the RAKE receiver was assumed to operate with four fingers.

Based on the statistics the ADD/SUB operation is the dominant operation. RAKE receiver, MRC, and FIR algorithms are the ones that contribute to the large number of ADD/SUB operations. Further understanding of the result would be gained by summing every operation for all applications.

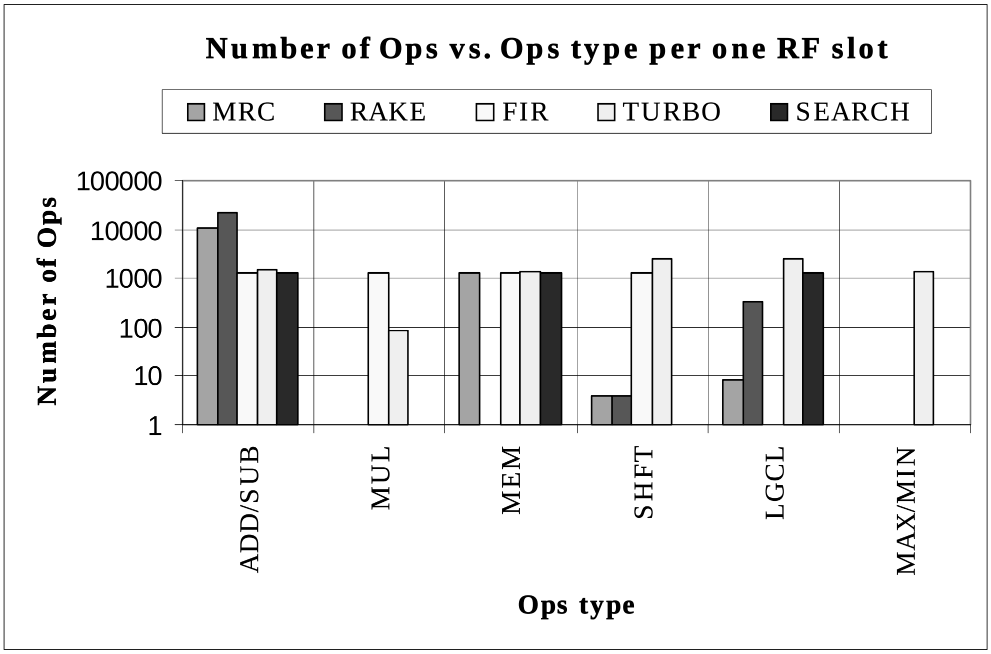


Figure 6-8: Number of operations vs. operation type for one RF slot (log scale)

Table 6-9 shows the percentage of each operation to the total number of operations. As shown, ADD/SUB operations are approximately 70% of all the total operations. Therefore, a fast and efficient implementation of ADD/SUB operations is necessary. The memory and shift operations makes nearly 17% of the total operations. The memory operation covers the RAM and FIFO storage type, and the shift operation covers the arithmetic and logicl shift operations. As a result, a RAM unit must be implemented that can also be configured to act as a FIFO and a fast barrel shifter that can be configured for arithmetic or logicl shift operations.

The design of the routing structure depends on the communication behavior of the applications. The number of connections between nodes of the algorithms for

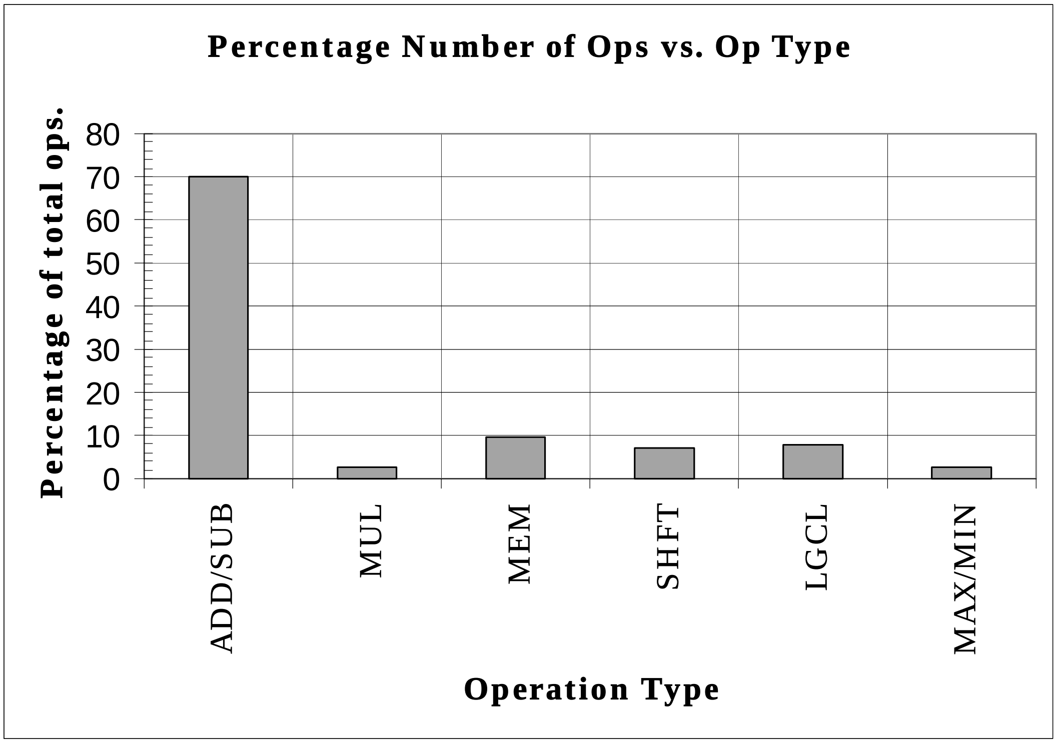


Figure 6-9: Percentage of the total number of operations for each operation

all five applications was counted. The notation of 1 to 1 to represent one node connected to one node, 1 to 2 was used to represent one node connecting to two nodes and so on. Figure 6-10 shows the total number of interconnections versus the interconnection type. Figure 6-11 shows a percentage of the total number of interconnections versus interconnection type. The figures show that the local connections dominate in all the applications. This locality is also related to the granularity of the execution elements, i.e. it is related to the size of the node. In this study the node was selected to perform the same operation that can be performed by the

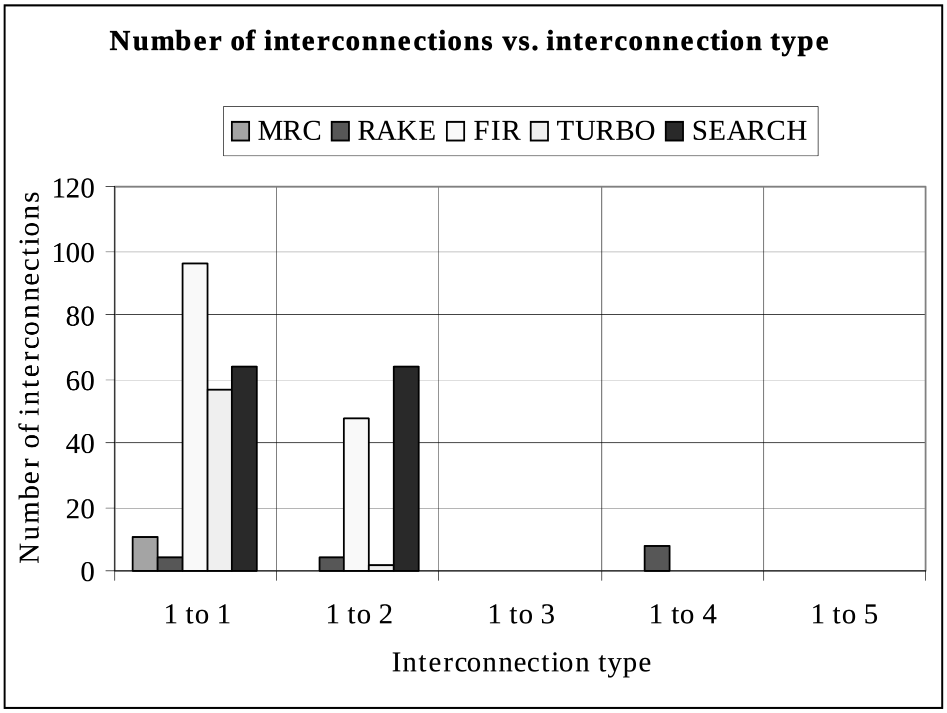


Figure 6-10: Number of interconnections of each application versus the types of connection

DRPU. Selecting the node to be primitive will result in a large number of long interconnection fan out.

6.5 Hardware Structure of the DRAW Architecture

This section presents detailed description of the hardware parts of the DRAW architecture. The DRPU and its internal hardware design is presented first. The structure of all the internal DRPU components are illustrated. The configuration bits of the DRPU (and subsequently its internal components) are explained afterword.

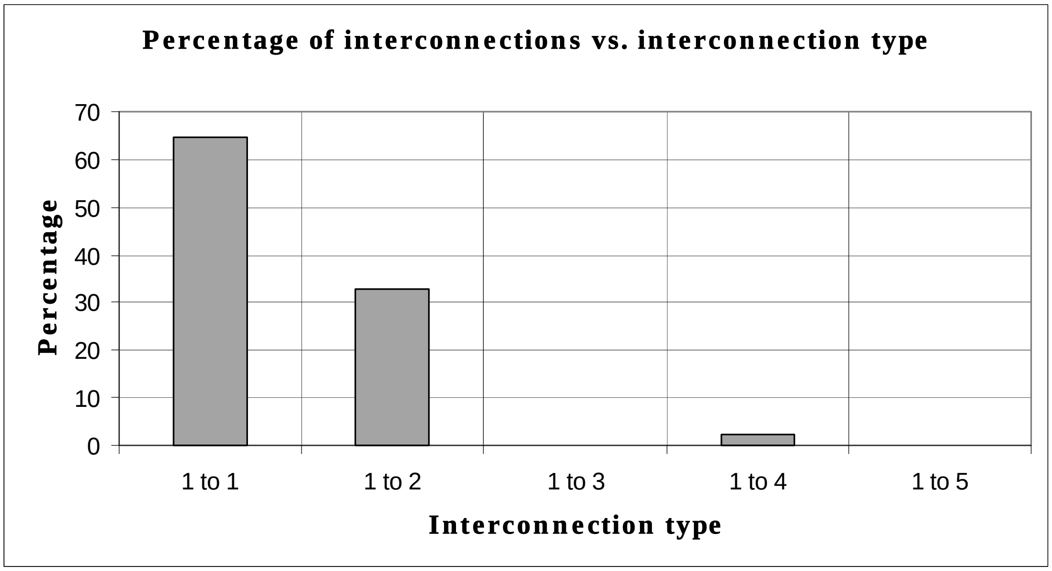


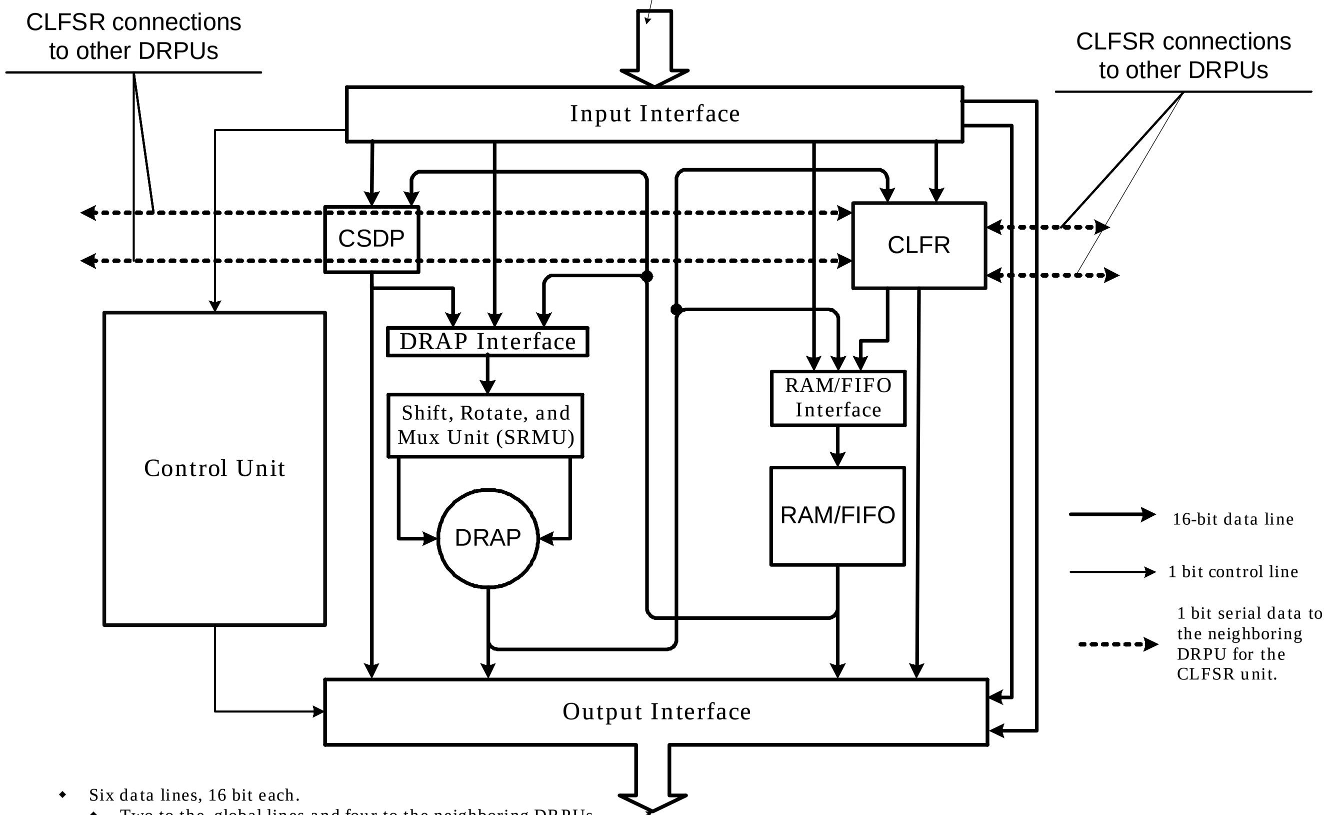
Figure 6-11: Percentage of interconnections of each application versus the type of connection

6.5.1 Dynamically Reconfigurable Processing Unit (DRPU)

The DRPUs are the major hardware components of the DRAW architecture for executing the arithmetic data operations. The DRPUs perform the required coarse-grained (16-bit) integer operations. This is in contrast to the Configurable Logic Blocks (CLBs) of commercially available fine-grained and general purpose FPGA-chips, which operate on the one-bit level [6]. As shown in Figure 6-12 each DRPU consists of the following components:

- One 16-bit Dynamically Reconfigurable Arithmetic Processing unit (DRAP),
- One Configurable Spreading Data Path (CSDP),

- Six input data lines, 16 bit each
  - Two from the global lines and four from local DRPU neighbors.
- Four Start_Hold control signals from neighboring DRPUs
- Four CARY signals from neighboring DRPUs


CLFSR connections to other DRPUs
Input Interface
CLFSR connections to other DRPUs
CSDP
CLFR
DRAP Interface
Shift, Rotate, and Mux Unit (SRMU)
DRAP
RAM/FIFO Interface
RAM/FIFO
Control Unit
Output Interface
16-bit data line
1 bit control line
1 bit serial data to the neighboring DRPU for the CLFSR unit.

- Six data lines, 16 bit each.
  - Two to the global lines and four to the neighboring DRPUs
- Four Start_Hold control signals to neighboring DRPUs. 1 bit each.
- One CARY signals to neighboring DRPUs. 1 bit.

Figure 6-12: Dynamically Reconfigurable Processing Unit DRPU

- One Configurable linear Feedback Shift Register (CLFSR),
- One DRPU controller,
- One dual port RAM/FIFO, and
- Two I/O interfaces.

The DRPU communicates with its nearest four neighbors through fast connection lines. A one-bit control signal START/HOLD is exchanged between neighboring DRPUs for control purposes. Each DRPU sends a DONE signal to its CSU whenever it completes its required task.

Figure 6-13 shows the top level block symbol and the signal connections of the DRPU. The synthesizeable VHDL code is provided in Appendix B.

6.5.1.1 Dynamically Reconfigurable Arithmetic Processing unit (DRAP)

Each DRAP can perform all arithmetic 16-bit operations identified in the above mentioned application of mobile communication systems. Table 6-1 is a list of the supported operations by the DRAP unit.

The block diagram of the DRAP is shown in Figure 6-14. The DRAP can perform the typical arithmetic operations. The DRAP is equipped with a fast barrel shifter (the VHDL code of the barrel shifter is shown in Appendix B on page 204), and a comparison unit. The barrel shifter can perform logical or arithmetic shifts and it can also be used as a typical shift register. The output of the DRAP is a 32 bit line. The output can be scaled down by the output interface.

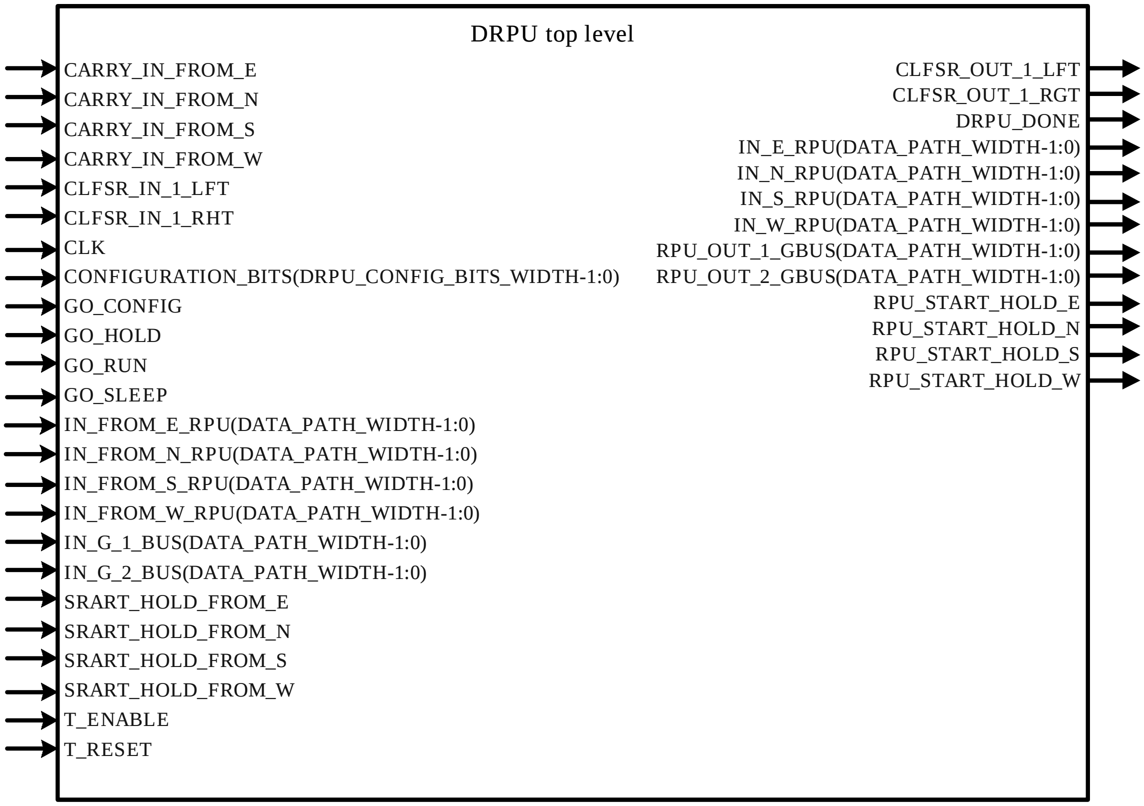
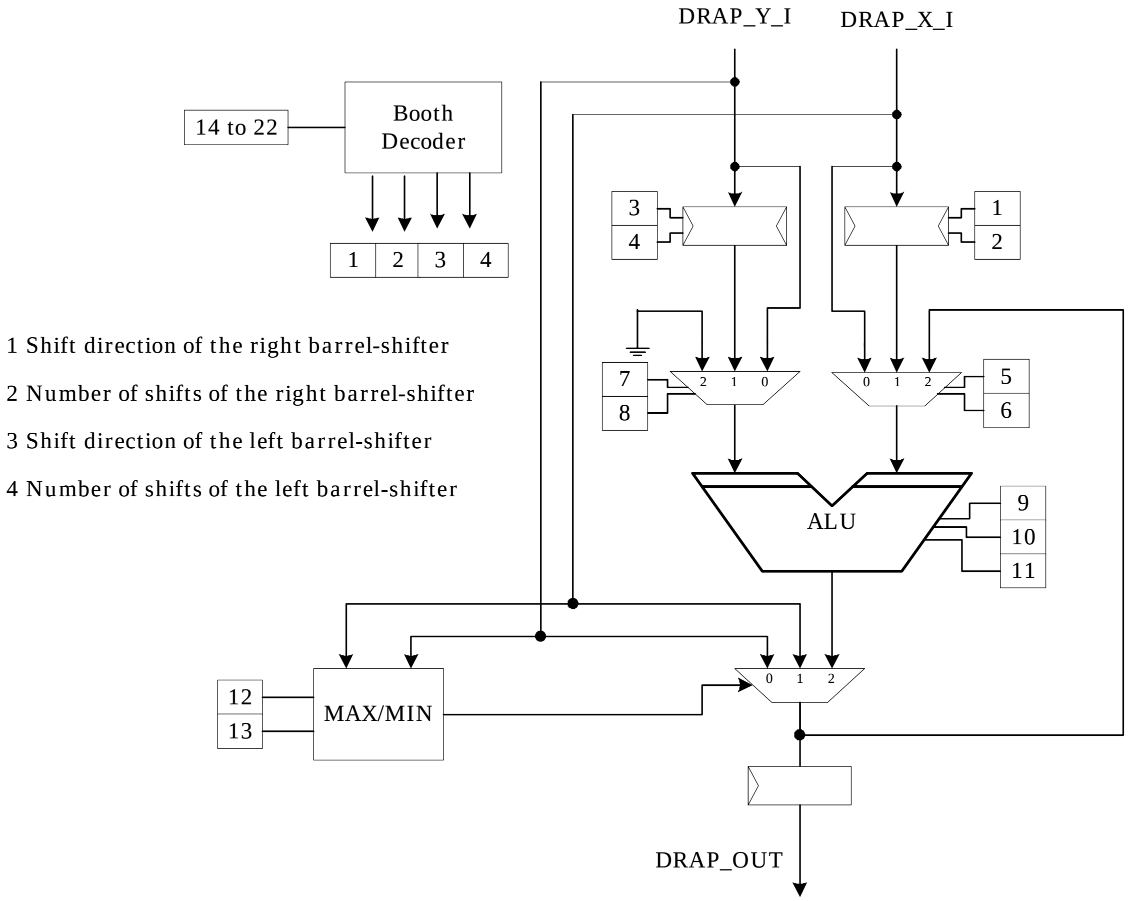

DRPU top level
CARRY_IN_FROM_E
CARRY_IN_FROM_N
CARRY_IN_FROM_S
CARRY_IN_FROM_W
CLFSR_IN_1_LFT
CLFSR_IN_1_RHT
CLK
CONFIGURATION_BITS(DRPU_CONFIG_BITS_WIDTH-1:0)
GO_CONFIG
GO_HOLD
GO_RUN
GO_SLEEP
IN_FROM_E_RPU(DATA_PATH_WIDTH-1:0)
IN_FROM_N_RPU(DATA_PATH_WIDTH-1:0)
IN_FROM_S_RPU(DATA_PATH_WIDTH-1:0)
IN_FROM_W_RPU(DATA_PATH_WIDTH-1:0)
IN_G_1_BUS(DATA_PATH_WIDTH-1:0)
IN_G_2_BUS(DATA_PATH_WIDTH-1:0)
SRART_HOLD_FROM_E
SRART_HOLD_FROM_N
SRART_HOLD_FROM_S
SRART_HOLD_FROM_W
T_ENABLE
T_RESET
CLFSR_OUT_1_LFT
CLFSR_OUT_1_RGT
DRPU_DONE
IN_E_RPU(DATA_PATH_WIDTH-1:0)
IN_N_RPU(DATA_PATH_WIDTH-1:0)
IN_S_RPU(DATA_PATH_WIDTH-1:0)
IN_W_RPU(DATA_PATH_WIDTH-1:0)
RPU_OUT_1_GBUS(DATA_PATH_WIDTH-1:0)
RPU_OUT_2_GBUS(DATA_PATH_WIDTH-1:0)
RPU_START_HOLD_E
RPU_START_HOLD_N
RPU_START_HOLD_S
RPU_START_HOLD_W

Figure 6-13: Dynamically Reconfigurable Processing Unit (DRPU) top level symbol

An example of the barrel shifter operations and configurations is shown in Table 6-2. The configuration string is ordered as follows [Direction Rotate/Shift Logical/Arithmetic Number_of_Shifts]. In the logical shift operation, the shifted bits location is filled with value ‘0’, while in arithmetic shift operation the location of the shifted bits is filled with the left or right most value depending on the shifting direction. The value x in the configuration string bits is a “don’t care” logic value.



1 Shift direction of the right barrel-shifter

2 Number of shifts of the right barrel-shifter

3 Shift direction of the left barrel-shifter

4 Number of shifts of the left barrel-shifter

Figure 6-14: Dynamically Reconfigurable Arithmetic Processing Unit (DRAP)

6.5.1.2 The DRPU Controller

The DRPU-controller is responsible for guiding all data manipulations and transfers inside the DRPU. Moreover, the DRPU-controller performs together with the CMU and its controller the fast dynamic reconfiguration of the DRPU. The finite state machine diagram of the DRPU controller is shown in Figure 6-15. The DRPU defaults to the *RESET* state at the start of the DRPU. From the *Reset* state

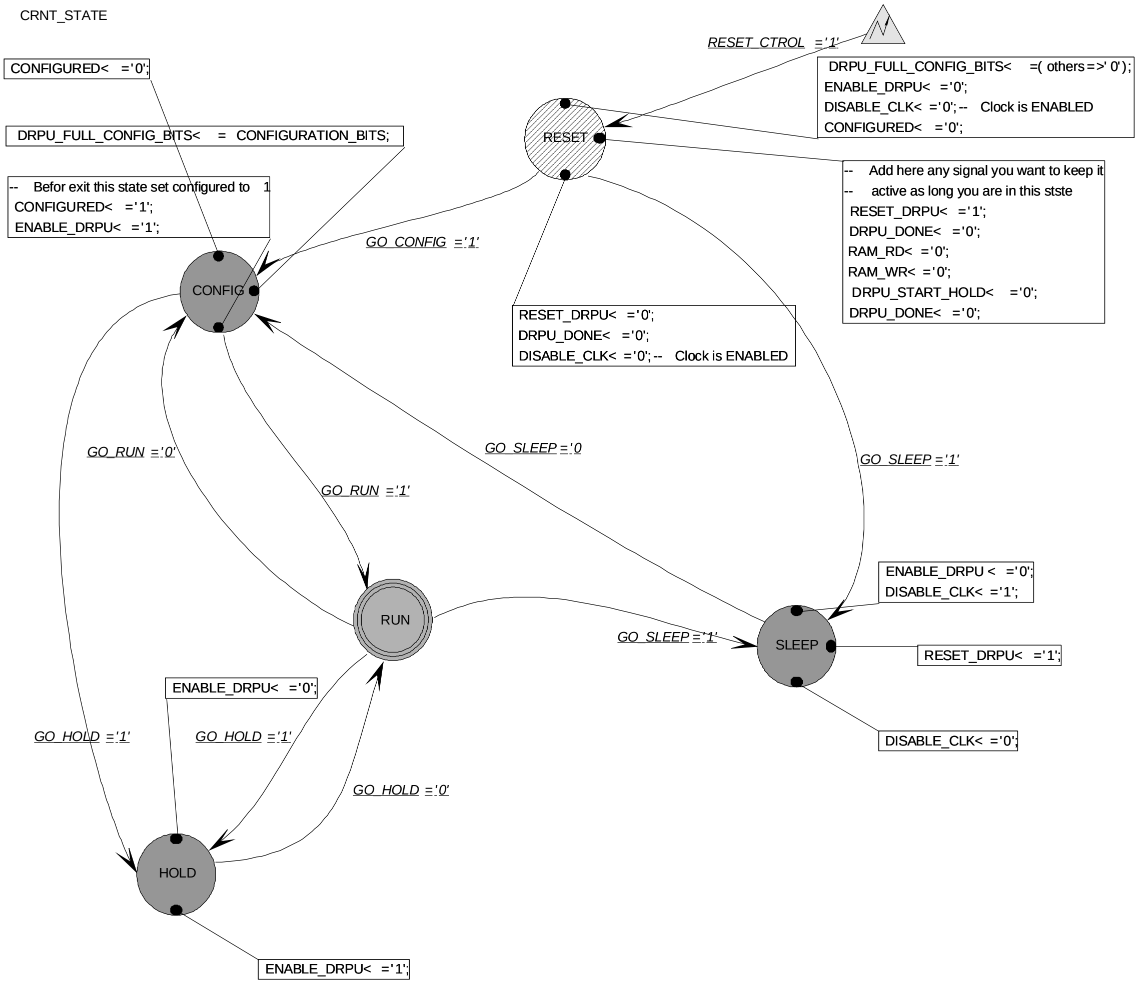


Figure 6-15: FSM diagram of the DRPU controller

the DRPU is either configured in the *CONFIG* state or sent to a sleeping mode in the *SLEEP* state. The DRPU is loaded with new configuration bits when the FSM is in the *CONFIG* state. After completing the loading of a new configuration bits, the FSM starts the execution of the configured operation in the *RUN* state. If the GO_HOLD signal is asserted by another DRPU through the Start-Hold mechanism, then the FSM goes to the *HOLD* state until the GO_HOLD is de-asserted. Since the DRAP is designed with its own controller, and since the other compo-

Table 6-1. A list of DRAP operations

#	Operation	Description
1	MUL	2’s complement multiplication
2	ADD	2’s complement addition
3	SUB	2’s complement subtraction
4	SHIFT	Logic & arithmetic shift
5	AND	Bit-wise AND
6	NAND	Bit-wise NAND
7	OR	Bit-wise OR
8	NOR	Bit-wise NOR
9	XOR	Bit-wise XOR
10	XNOR	Bit-wise XNOR
11	NOT	Bit-wise NOT
12	MAX	Maximum
13	MIN	Minimum

Table 6-2. An example of the barrel shifter operation.

Input vector	Operation	Configuration	Output Vector
[1001101001001011]	Rotate right by 7 bits	[1 0 x 111]	[1001011100110100]
[1001101001001011]	Rotate left by 3 bits	[0 0 x 011]	[1101001001011100]
[1101101101100011]	Shift Right Logical by 5 bits	[1 1 0 101]	[0000011011011011]
[1101101101100011]	Shift left Logical by 4 bits	[0 1 0 100]	[1011011000110000]
[1001101001110100]	Shift Right Arithmetic by 2 bits	[1 1 1 010]	[1110011010011101]
[1001101001110100]	Shift left Arithmetic by 6 bits	[1 1 1 110]	[1001110100000000]

nents in the DRPU requires minimal control, the *RUN* state task is to only control the RAM/FIFO and implement the Start-Hold mechanism.

6.5.1.3 The RAM/FIFO Unit

The 8 words by 16-bit dual port RAM within each DRPU can be used as a look-up-table (LUT). The RAM/FIFO can be used as normal memory or if necessary it can be used as a first-in/first-out (FIFO) memory, e.g., for buffering intermediate results.

The block diagram of the RAM/FIFO unit along with its signal interface is shown in Figure 6-16. When the RAM/FIFO unit is configured as a FIFO it follows the FIFO behavior as shown in Figure 6-17.

6.5.1.4 The Configurable Spreading Data Path (CSDP)

The Configurable Spreading Data Path (CSDP) unit (refer to Figure 6-12) is implemented in every DRPU. It performs fast executions of CDMA-based spreading tasks. The unit can be either used with the two adders in the DRPU to perform a complex correlation function found in QPSK modulation, or it can perform one- or two-channel normal correlations, as found in the Binary Phase Shift Keying

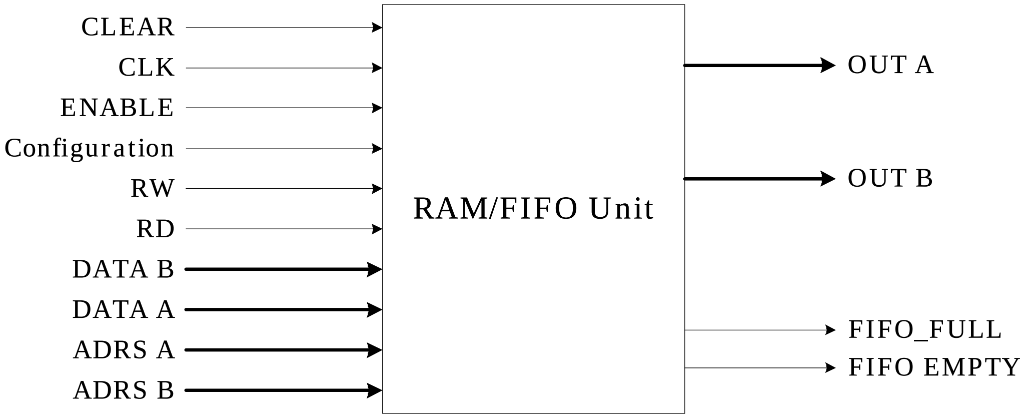


Figure 6-16: A block diagram of the RAM/FIFO unit showing its signal interface

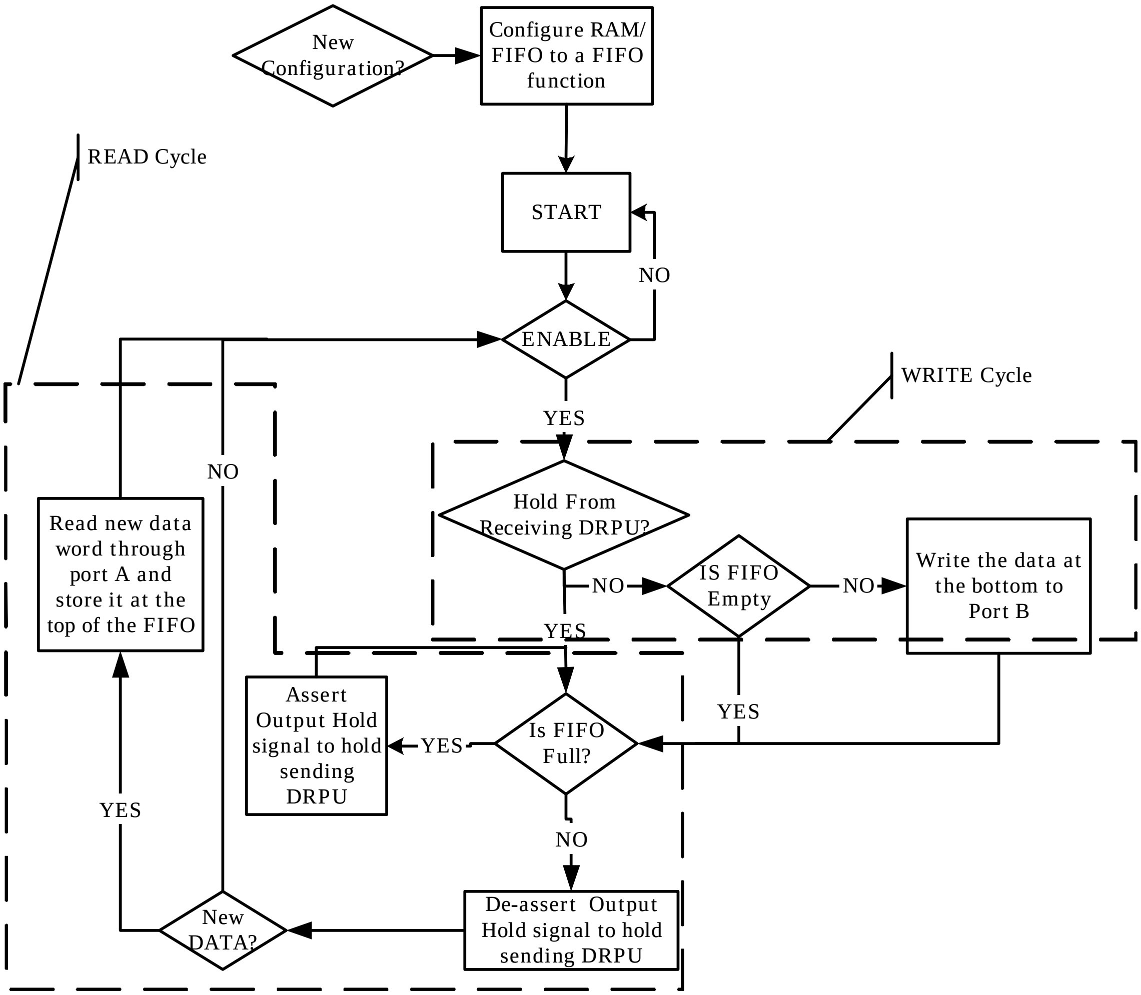

New Configuration?
Configure RAM/ FIFO to a FIFO function
READ Cycle
START
NO
ENABLE
WRITE Cycle
YES
NO
Hold From Receiving DRPU?
Read new data word through port A and store it at the top of the FIFO
NO
IS FIFO Empty
NO
Write the data at the bottom to Port B
YES
Assert Output Hold signal to hold sending DRPU
YES
Is FIFO Full?
YES
YES
NO
New DATA?
De-assert Output Hold signal to hold sending DRPU

Figure 6-17: FIFO control flow diagram

(BPSK) modulation [70]. It is mainly designed to perform a complex correlation operation for QPSK-schemes on 8-bit data words with serial codes (e.g. PN-codes) for any number of iterations N. The CSDP can also be utilized in many other functions, e.g., despreading, synchronization, etc. The inputs to the unit can come from outside the DRPU or from the local memory.

6.5.1.5 DRPU I/O interfaces.

The routing of the signals in, out and inside the DRPU is organized by a set of interfaces. At the input there is the DRPU input interface. Table 6-3 lists the input possible configurations of the DRPU input interface for the data lines. While

Table 6-3. Configuration table of the DRPU input interface

Configuration Number	DRPU_IN_1	DRPU_IN_2	DRPU_IN_3	CARY_1	CARY_2
0	N	S	W	N	S
1	N	S	E	N	S
2	N	S	G1	N	S
3	N	S	G2	N	S
4	N	W	E	N	W
5	N	W	G1	N	W
6	N	W	G2	N	W
7	N	E	G1	N	E
8	N	E	G2	N	E
9	N	G1	G2	N	E
10	S	W	E	S	W
11	S	W	G1	S	W
12	S	W	G2	S	W
13	S	E	G1	S	E
14	S	E	G2	S	E
15	S	G1	G2	S	S
16	W	E	G1	W	E
17	W	E	G2	W	E
18	W	G1	G2	W	W
19	E	G1	G2	E	E

Figure 6-18 illustrates the way by which an input data line are selected by the interface. The selection of the data lines and carry signals for the DRAP is illustrated as an example. As shown in the figure, the process of selecting the data lines to be fed to the DRAP is done at two levels. At the input interface the data lines are reduced to three out of six and the carry signals are reduced to two lines out of four. Then at the DRAP interface two data lines are selected out of the three lines and one carry line is selected out of the two carry lines.

6.5.2 Fast Inter-DRPU local and Global Communication Mechanism

Each DRPU is locally connected to its four neighbors (North, East, South, and West) through an 8-bit fast direct connection line. It can be connected to the global lines through a SRAM-based switching box, as shown in Figure 6-19.

The global interconnect lines are implemented with two 16-bit lines, which run to the neighboring SWBs. As shown in Figure 6-20 each global line coming from one direction can be routed to any of the other directions. Each SWB consists of 20 switching-points implemented by 20 SRAM-controlled pass transistors. All combinations of these one-line connections are possible as long as there is no resource conflict.

6.5.3 DRPU Configuration Bits

One complete DRPU configuration word consists of 64 bits. The organization of such configuration bits is shown in Table 6-4.

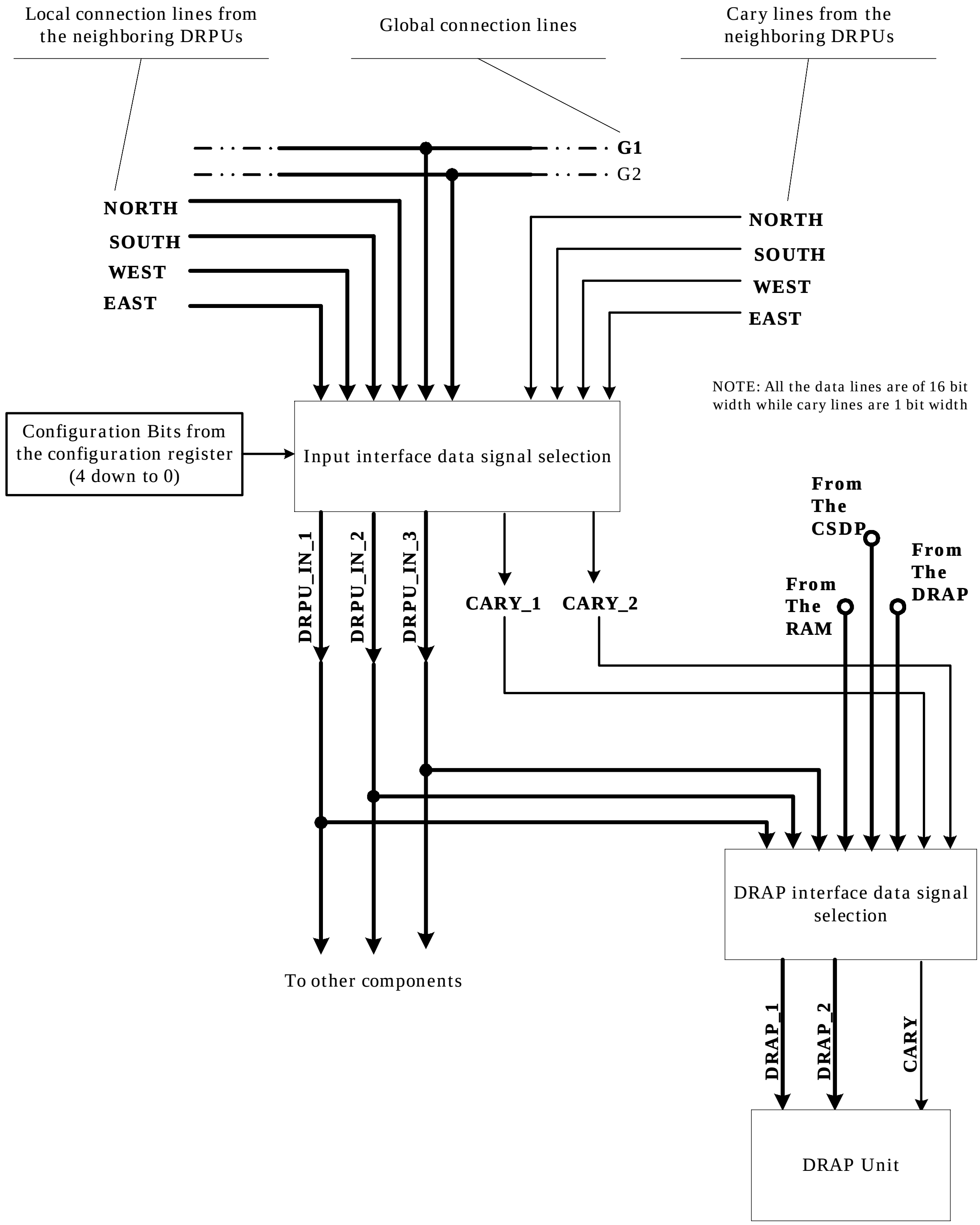

Local connection lines from the neighboring DRPUs
Global connection lines
Cary lines from the neighboring DRPUs
G1
G2
NORTH
SOUTH
WEST
EAST
NORTH
SOUTH
WEST
EAST
NOTE: All the data lines are of 16 bit width while cary lines are 1 bit width
Configuration Bits from the configuration register (4 down to 0)
Input interface data signal selection
From The CSDP
From The DRAP
From The RAM
DRPU_IN_1
DRPU_IN_2
DRPU_IN_3
CARY_1
CARY_2
DRAP interface data signal selection
To other components
DRAP_1
DRAP_2
CARY
DRAP Unit

Figure 6-18: Input and DRAP interfaces selection diagram of the data lines

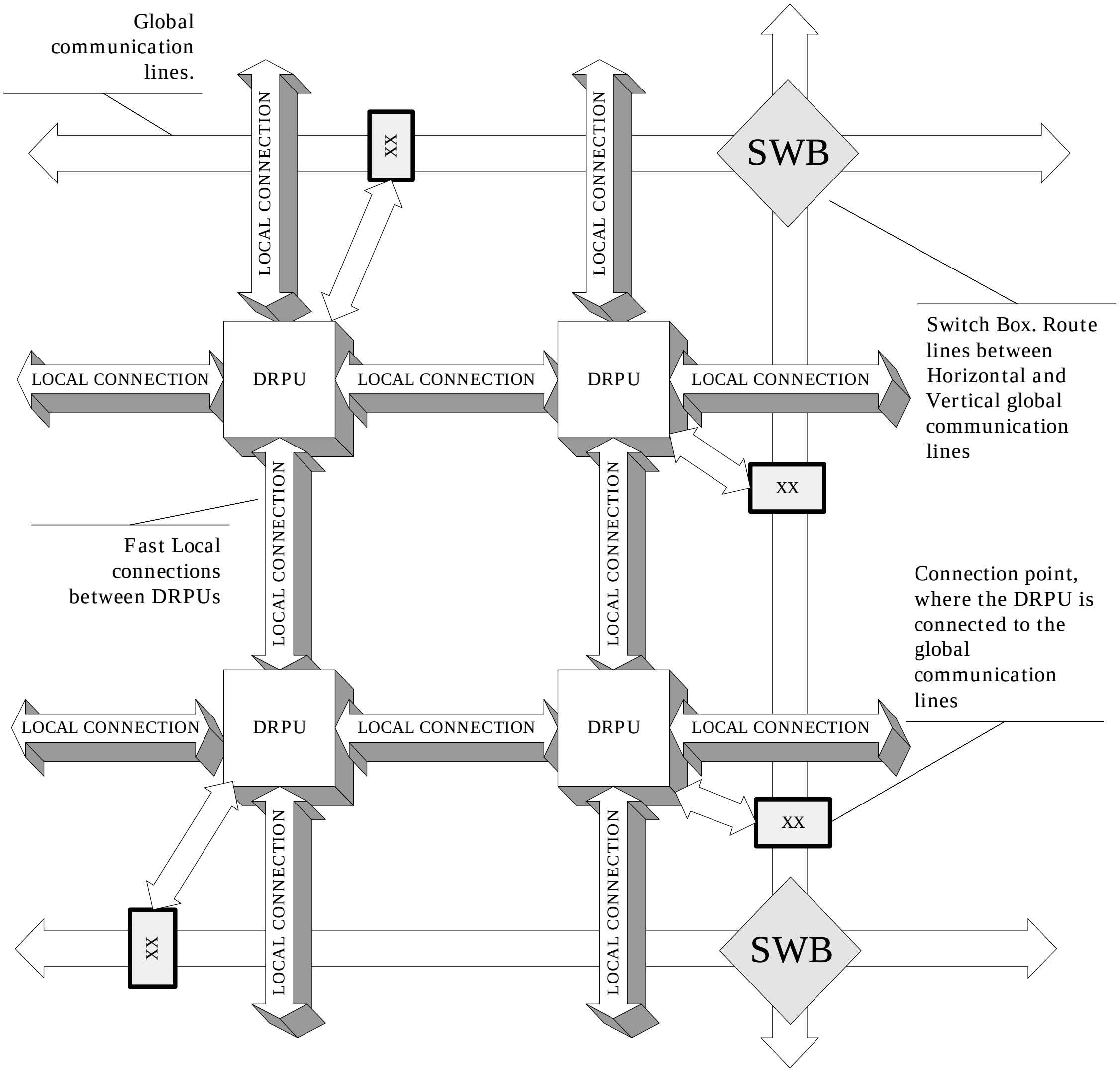
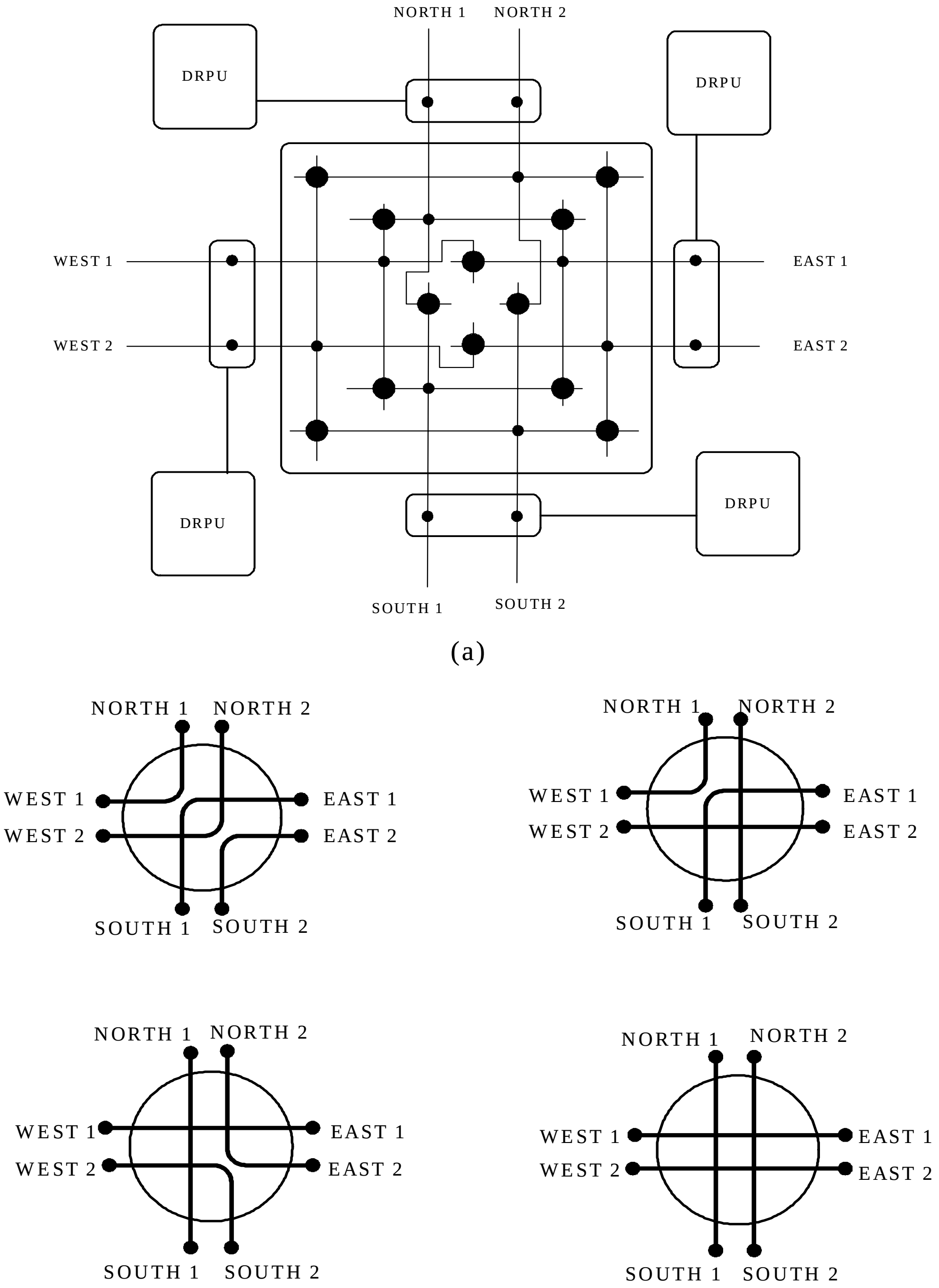


Figure 6-19: DRPU Communication structure

The configuration bits are transmitted during configuration from the global communication unit to the CSU in 8 bytes. This method is selected to reduce the number of lines required to distribute the configuration throughout the architecture.



(a)

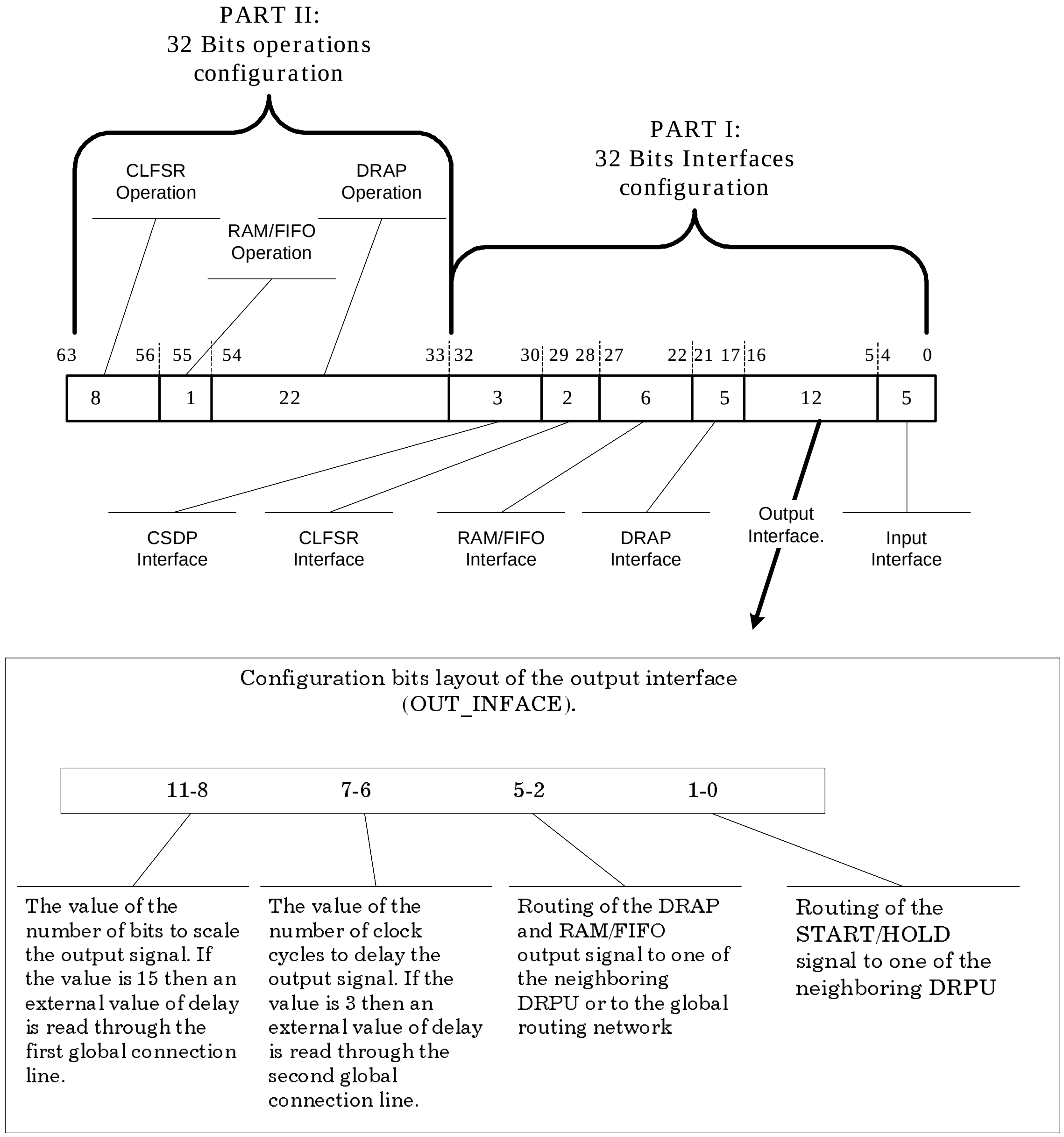
(b)

Figure 6-20: A connection topology for the SWB: (a) Complete connection lines. (b)

Table 6-4. DRPU configuration bits structure

Unit Name	Number of Bits	Configuration Bits
Input Interface	5	0 to 4
Output Interface	12	5 to 16
DRAP Interface	5	17 to 21
RAM_FIFO Interface	6	22 to 27
CLFSR Interface	2	28 to 29
CSDP Interface	3	30 to 32
RAP Operations	22	33 to 54
RAM_FIFO Operations	1	55
CLFSR Operations	8	56 to 63
TOTAL BITS	64	0 to 63

The configuration bits of the DRPU can be organized in many ways. A structure that groups the configuration bits into two minor groups was chosen for the design. The configuration bit set of one DRPU is shown in Figure 6-21. Also shown in the figure is the expansion of the configuration of the output interface. The first part of the configuration bits contains the configuration bits for the routing paths inside the DRPU, i.e. the configuration of the interfaces of the DRPUs internal components. The second group contains the configuration bits for the operations of the DRPU. This layout was selected since for any application mapped to the architecture, there is either more dynamics in routing over operation or vice versa. Such scheme will reduce the amount of configuration bits to be moved between the DRPU and the CSU. This reduction is accomplished by observing the behavior of the application mapped to the architecture, and then updating either the routing bits or the operation bits depending on the need to update.



The value of the number of bits to scale the output signal. If the value is 15 then an external value of delay is read through the first global connection line.

The value of the number of clock cycles to delay the output signal. If the value is 3 then an external value of delay is read through the second global connection line.

Routing of the DRAP and RAM/FIFO output signal to one of the neighboring DRPU or to the global routing network

Routing of the START/HOLD signal to one of the neighboring DRPU

Figure 6-21: The configuration bits structure of one DRPU (the configuration of the output interface is detailed as an example)

6.6 Area and Performance Figures

The complete DRPU architecture was synthesized using a Fujitsu CS71 standard cell ASIC library. The Fujitsu CS71 is 0.25 μm CMOS technology with up to 5 metal layers. The synthesis was done using Leonardo Spectrum synthesis tool. Figure 6-22 shows the output of the synthesis tool for the DRPU. This schematic can be converted into a physical VLSI layout diagram in the future for the purpose of fabrication. The synthesis timing and area reports are summarized next.

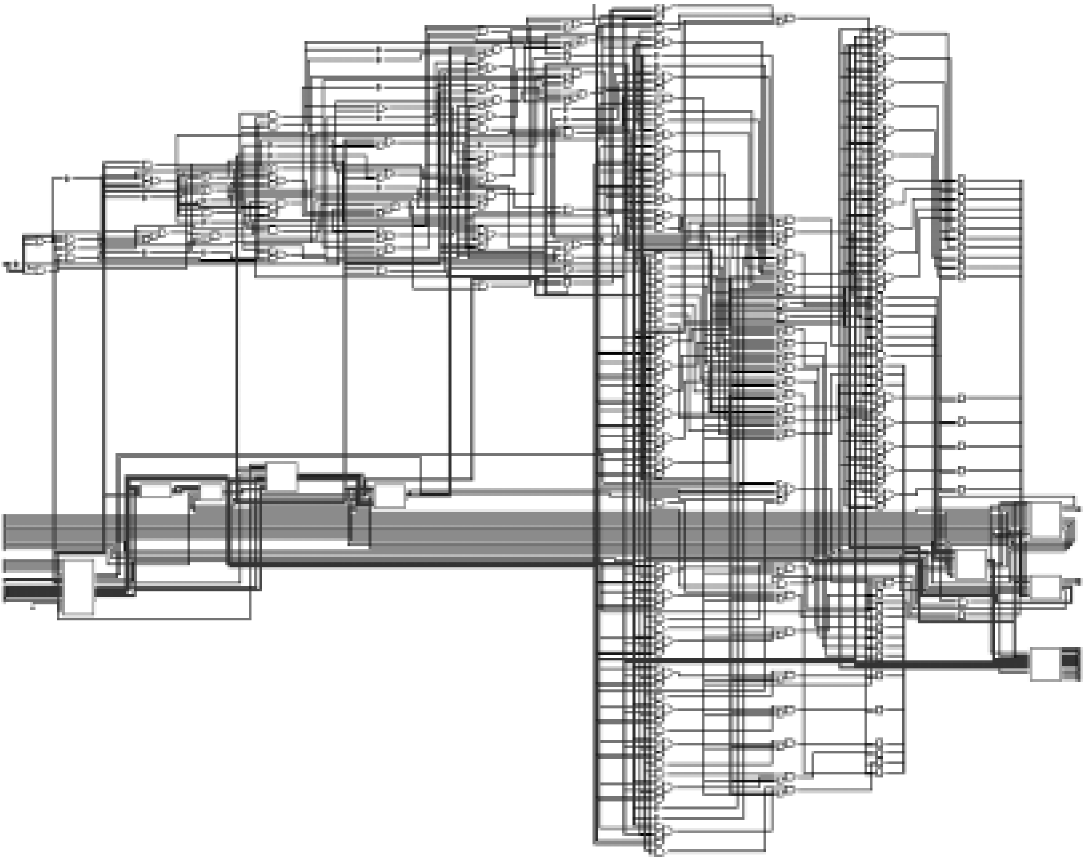


Figure 6-22: The schematic diagram of the synthesized DRPU using Leonardo Spectrum

An area requirement of one DRPU of the design is shown in Table 6-5. Table 6-6 shows the area consumed by one unit design. One unit design is unit of the architecture that when repeated a certain number of times can create the complete architecture. The unit design consists of one DRPU, one fourth of the CMU, four

Table 6-5. Area consumed by each component of the DRPU

Unit	Number of gates	Area in Microns square
CLFSR	913	41,998
DRAP	2,767	127,282
DRAP Interface	401	18,446
RAM_FIFO Interface	296	13,616
RAM_FIFO	3,182	146,372
INPUT Interface	738	33,948
OUTPUT Interface	2,328	107,088
DRPU Controller	723	33,258
CSDP	1,121	51,566
TOTAL Area of DRPU	12,469	573,574

Table 6-6. Area consumed by one unit design

Unit	Number of gates	Area in Microns square
DRPU	12,469	573,574
CMU/4	4353/4 = 1089	50,094
DIO	109	5,014
CSU	1597	73,4**62**
SWB	1239	56,994
TOTAL area of design unit	16,503	759,138

and half SWB, and one DIO. Assuming a chip area of $11 \times 11\ mm^2$ based on MOSIS 0.25 available technologies for research, we can have up to 159 units of design. With this estimation, a 10X10 array of DRPUs can be easily fabricated on the chip.

The preliminary performance numbers of the architecture are presented based on the timing simulation after synthesis. The synthesis was done with the maximum optimization for area and speed. The longest delay of all of the components is 11.36 ns of the DRAP unit. This was expected since the DRAP is the arithmetic engine of the DRPU. This time delay corresponds to a maximum clock of the DRAP 88.0 MHz. As a result the system clock was set to 60 MHz to take in the parasitic effects.

Based on the system clock of 60 Mhz the performance of the different operations in the DRPU are listed in Table 6-7

Table 6-7. Performance of the different operations of the DRPU

#	Operation	Number of cycles	Operation rate
1	Multiplication	3	20 Mops
2	Addition/Subtraction	1	60 Mops
3	Barrel Shifting	1	60 Mops
4	Logical (AND,OR,NOT)	1	60 Mops
5	MAC	3	20 Mops
6	Scaling	1	60 Mops

Chapter 7

Mapping Examples

7.1 Introduction

The mapping of an application into DRAW involves the complete understanding of the architecture's built-in capabilities. It also requires the understanding of the application to be mapped on the algorithmic level, and the ability to modify the algorithm to take advantage of the architecture's built-in features. The architecture is designed to be simple to model for the sake of automating the mapping process. The simplicity comes from the regularity of the architecture in addition to its simple routing structure. This chapter will present some examples of the mapping process for two applications: a gold code generator and a finite impulse response (FIR) filter. These two applications dominate the hardware of the baseband processing unit.

A symbolic representation of the DRPU suitable for a mapping description is shown inFigure 7-1. An example of the use of the DRPU symbolic representation showing the DRPU configured into different operations and connections is shown in Figure 7-2.

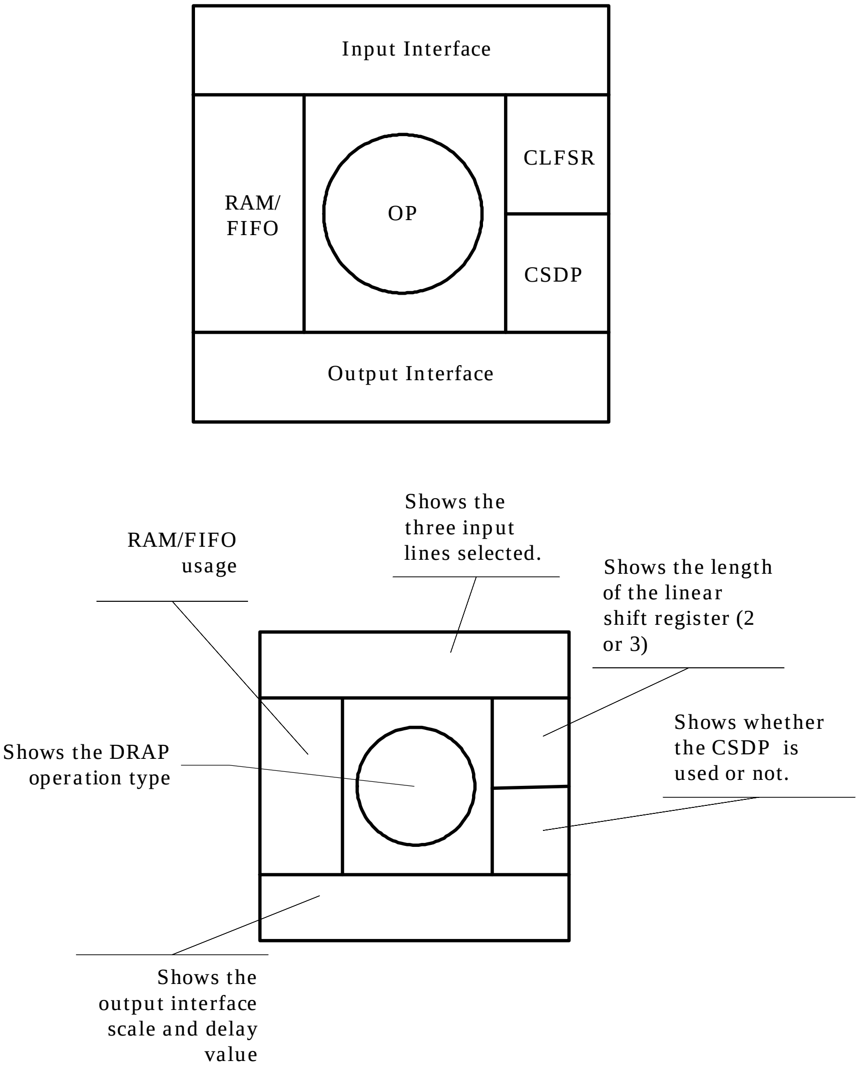

Input Interface
RAM/ FIFO
OP
CLFSR
CSDP
Output Interface
RAM/FIFO usage
Shows the three input lines selected.
Shows the length of the linear shift register (2 or 3)
Shows whether the CSDP is used or not.
Shows the DRAP operation type
Shows the output interface scale and delay value

Figure 7-1: Symbolic representation of the DRPU to show the general configuration

7.2 Mapping Gold-Code Generator

Different types of codes are used in the baseband processing unit. Most of the codes are forms of Pseudorandom Noise (PN) codes. PN codes can be generated

EXAMPLE: Configuration of a DRPU

The input interface is configured to select the 1st Global line, the North DRPU output, and the West DRPU output.

The CLFSR is utilized as 3 stages LFSR. The output of the CLFSR is routed to the next CLFSR in the right DRPU neighbor.

The DRAP is configured to do a Multiplication operation.

The output Interface is configured to scale the results by 2 right shifts and no delay.

The other components of the DRPU are not used in this example.

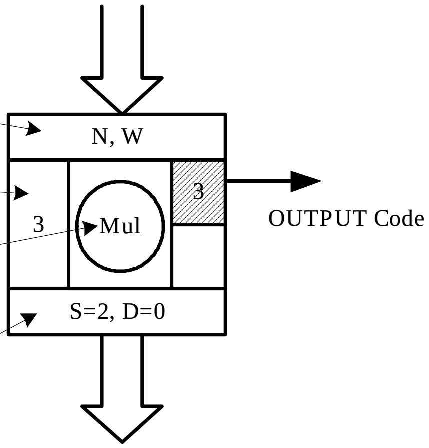


Figure 7-2: An example of the use of the DRPU symbolic representation

using one or more linear feedback shift registers. M-sequence codes are simple PN codes. M-sequence codes of a length of 2^N-1 can be generated using a linear feedback shift register of length N based on prime polynomial order N as shown in Figure 7-3. Gold codes are generated by XOR-ing two M-sequences. This section shows the mapping of a three-stage M-sequence generator onto a single DRPU. By combining two 5-stages M-sequence generator while XOR-ing their outputs one gets a Gold code generator. The codes are generated in a one-bit stream. The code is then grouped in a 16-bit word and sent to the output of the code generation unit. This example shows how generations of other codes can be easily done by altering the configurations of the configurable linear feedback shift register units.

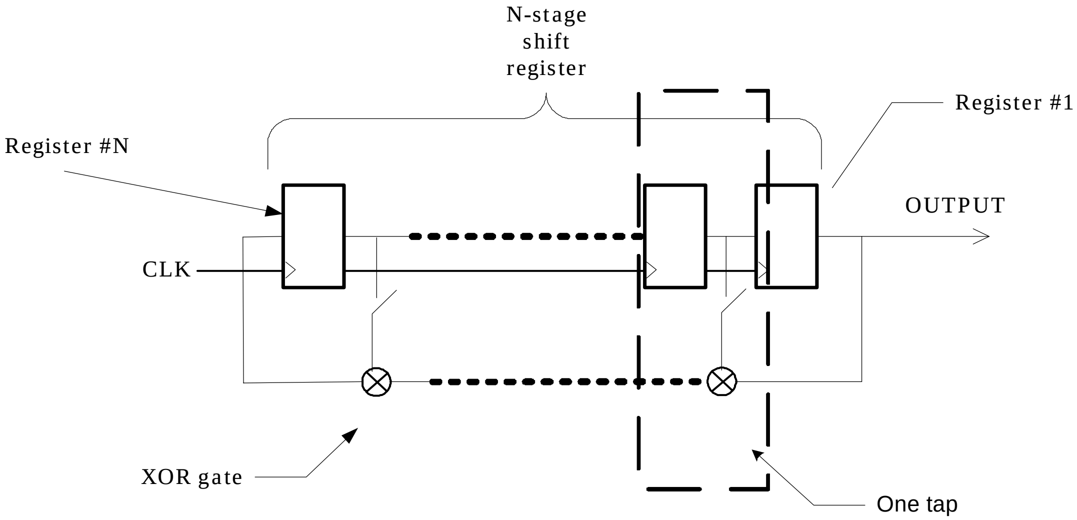


Figure 7-3: N-stages linear feedback shift register

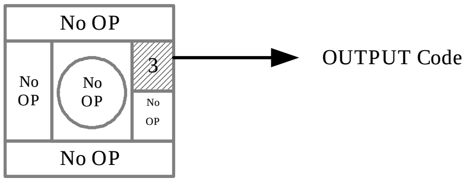


Figure 7-5: A symbolic representation of one DRPU configured as a three-stage LFSR

7.2.1 M-sequence Generator Using One DRPU

Figure 7-4 shows a MATLAB code fragment of a general M-sequence code generator. It was used to generate a simple seven chips long M-sequence code of the polynomial $1+X^2$ which is represented as [1 0 1] in MATLAB. The generated code is shown in Figure 7-4.

```
%M-sequence spreading code generator
function [M_code, reg_state]=Mgen(poly_vector, num_chips,
inti_loading)
%M_code is the generated M-sequence code of length num_chip
%reg_state returned the last loading of the reg.
%poly_vector is the characteristic polynomial. For example
[1 0 1 1] represent the polynomial 1+x^2+x^3.
%inti_loading must not be all zeros.

reg_state=inti_loading;
for i=1:num_chips
   temp_reg=poly_vector.*reg_state;
mul=mul+length(temp_reg);
   temp_xor=temp_reg(length(poly_vector));
           for j=length(poly_vector)-1:-1:1
                 temp_xor=xor(temp_reg(j),temp_xor);
           end
         %temp_xor=rem(sum(temp_reg),2);
         fed_bit=temp_xor;
         M_code(i)= not(reg_state(3));
        reg_state=[fed_bit, reg_state(1:length(poly_vector)-
1) ];


      end
```

```
MCODE= Mgen([1 0 1],10,[1 1 1])

MCODE = 0     0     0     1     0     1     1     0     0     0
```

Figure 7-4: Matlab code fragment of M-sequence generator

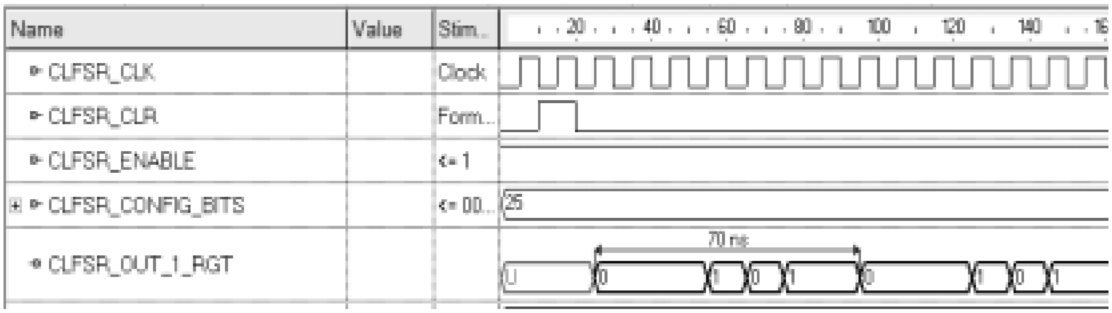


Figure 7-6: CLFSR output for a three-stage M-sequence with polynomial $1+X^2$

This simple code generator can be mapped onto one DRPU. The CLFSR can be configured to map three stages LFSR as shown symbolically in Figure 7-5. Figure 7-6 shows the output of the CLFSR when configured as a three-stage linear feedback shift register. As seen in the figure, the code lasts for seven cycles and then repeats again. As expected, the output code matches the Matlab generated code.

7.2.2 M-sequence Generator Using Two DRPUs (5-Stage LFSR)

Based on three-stage code generator, five stage LFSR that spans over two DRPUs is designed. The five stages are divided into two parts with two and three stages respectively. The first part will be mapped onto the CLFSR of the first DRPU. The CLFSR will be configured as two stages and at the same time as the left segment of a LFSR. That means the CLFSR is connected to its neighboring DRPU which is mapping the second three stages part.

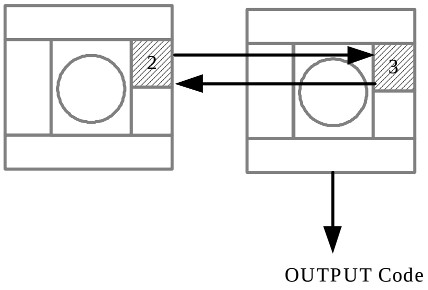


Figure 7-7: The symbolic representation of two DRPU configured as a five-stage LFSR

Figure 7-7 shows the symbolic representation of the final mapping of the five-stage M-sequence code. Note that the output is driven through the output interface without delay or scaling.

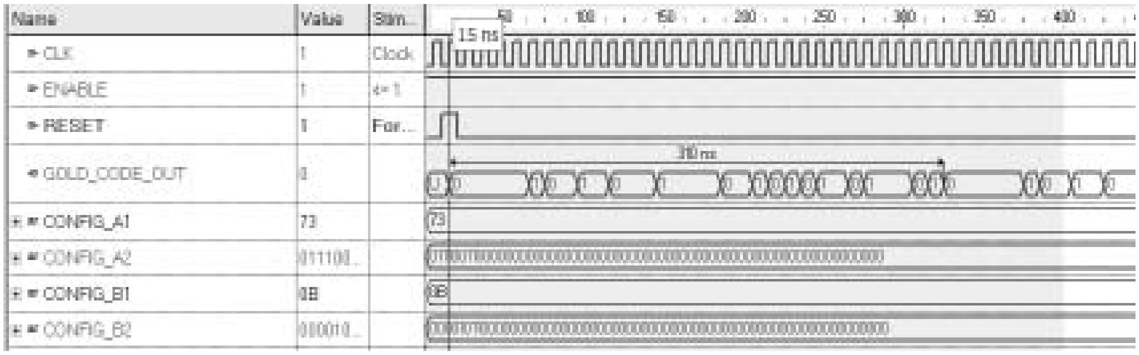
Figure 7-8 shows the Matlab output of the M-gen function for the five-stage LFSR implementing the polynomial $1+x+x^2+x^4+x^5$. When mapping this polynomial into DRAW, the polynomial is first partitioned into two parts to fit into two DRPUs. The first part that is mapped into DRPU1 is $1+x+x^2$ and the second part is x^4+x^5. The second part is normalized to $1+x^4$. Figure 7-9 shows the output of the five-stage LFSR mapped onto two DRPUs. As can be seen the generated code matches the code generated from the Matlab simulation.

7.2.3 Gold code Generator

Gold codes are widely used in the WCDMA receiver as scrambling codes. The M-sequence is generated by a LFSR as shown in the previous section. In this section two M-sequence generators were concatenated by XOR-ing their outputs.

```
>> Mgen([1 1 0 1 1],31)
ans =

  Columns 1 through 22

     0     0     0     0     0     1     0     0     1     1
0     0     0     1     1     1     1     0     0     1     0     1

 Columns 23 through 31

     0     1     1     0     1     1     1     0     1
```

Figure 7-8: Matlab output of the M-gen function generated for the polynomial $1+x+x^2+x^4+x^5$



Name	Value	Stim...
CLK	1	Clock
ENABLE	1	<= 1
RESET	1	For...
GOLD_CODE_OUT	0	
CONFIG_A1	73	
CONFIG_A2	011100...	
CONFIG_B1	0B	
CONFIG_B2	000010...	

Figure 7-9: The output of an M-sequence generator for the polynomial $1+x+x^2+x^4+x^5$ mapped onto two DRPUs

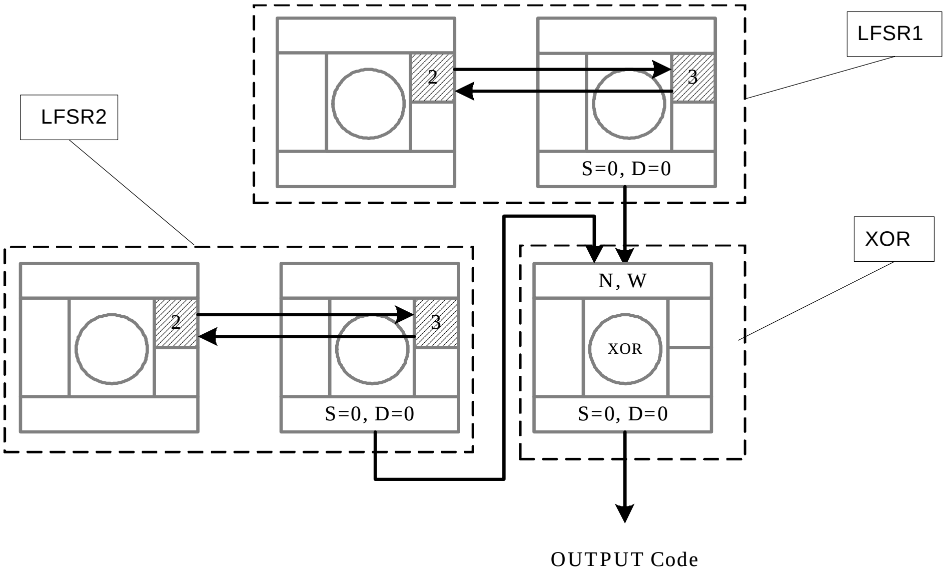


Figure 7-10: Symbolic representation of Gold code generator mapped onto five DRPUs

The hardware mapping of the gold code generator costs five DRPUs. Four DRPUs are used to generate the two M-sequences, the fifth DRPU is performing the XOR operation. The symbolic representation of such mapping is shown in Figure 7-10. The Matlab code that simulates the Gold code generation is shown in Figure 7-11. A simulation of the hardware mapping results is shown in Figure 7-12. These two figures show that the generated codes are identical.

Implementation of the Gold code generator onto a Xilinx Virtex chip is a function of number of taps. Table 7-1 shows a comparison between DRAW and Virtex implementation. Note that for Virtex as the number of taps increases more Virtex slices are required. Although this is true for DRAW, the coarse datapath (16 bits) of DRAW and the fast local connections between DRPUs eliminate the need

```
MATLAB Code of Gold code gen
function [OUT_CODE]=GOLD_SEQ(LENGTH, POLY_1, POLY_2)

%We will use the mgen function
%function [M_code, reg_state]=Mgen(poly_vector, num_chips)

CODE1=Mgen(POLY_1,LENGTH)
CODE2=Mgen(POLY_2,LENGTH)

OUT_CODE=xor(CODE1,CODE2)
```

```
>> GOLD_SEQ(31,[0 1 0 0 1],[1 1 0 1 1])
ans =

 Columns 1 through 20
     0     0     0     0     0     0     1     0     1     0
0     1     1     1     0     0     0     1     0     0

 Columns 21 through 31

     0     0     0     1     1     1     1     1     0     1     0
```

Figure 7-11: MATLAB Code of Gold code generator function and the output run for the two polynomials $1+x^2+x^5$ and $1+x+x^2+x^4+x^5$

for global routing lines. More routing overhead is incurred in Virtex as the number of taps increases due to the fine-grain PEs of the Virtex architecture.

Table 7-1. Implementation cost of 8-stages 4 taps on DRAW and Virtex

	DRAW	Virtex
Number of PEs	3 DRPUs	6.5 Slices (3.25 CLBs)
Number of Global Routing lines Required	0	3 lines
Number of Clock cycles to Reconfigure	2	over 30

7.3 Mapping an FIR filter

Finite Impulse Response filters are essential in the baseband processing. FIR filters are implemented using multipliers, adders, and delay elements. Figure 7-13 shows a typical implementation of the FIR filter. In this structure, the incoming data passes through a tapped delay line. The data symbols are multiplied by the filter coefficients before adding the results in every cycle to generate the output

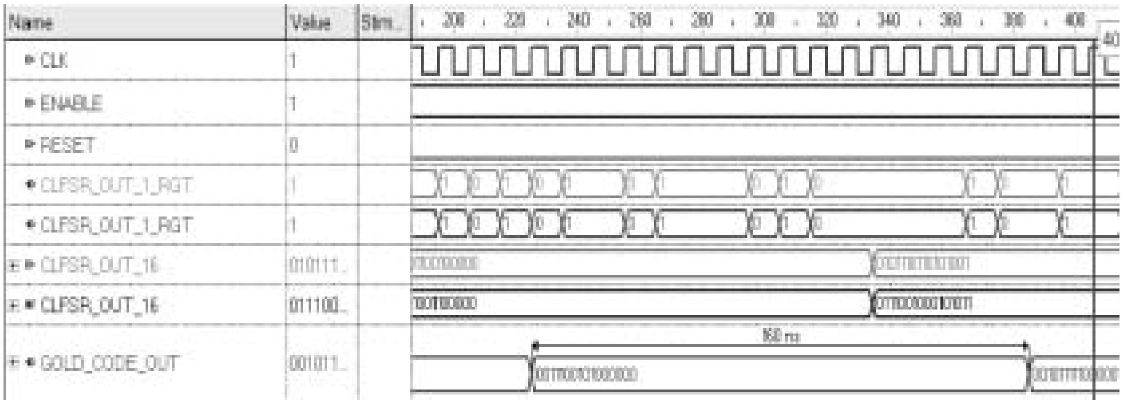


Figure 7-12: The output of the mapped Gold code generator onto five DRPUs

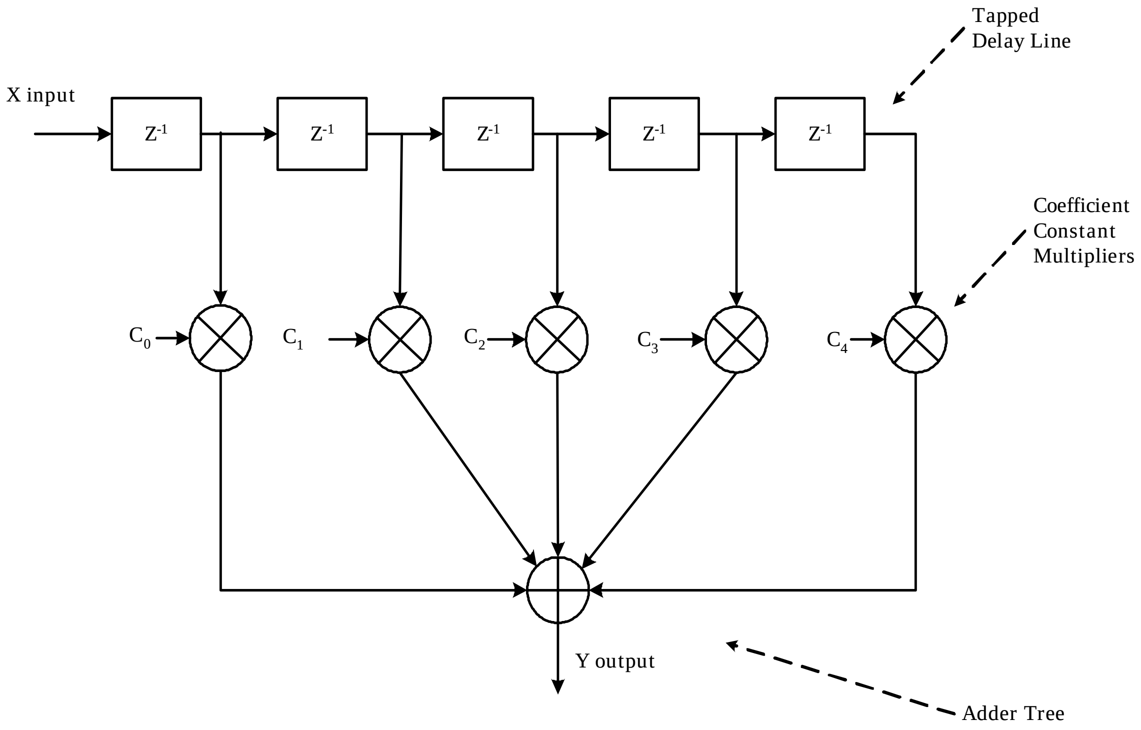


Figure 7-13: Typical implementation of FIR filter

filtered data. An alternative implementation structure is shown in Figure 7-14. This structure is called the FIR transposed form. In this structure the input data is applied to all the taps in parallel. The output of each tap is added to the previous tap. A one delay unit is needed between one tap and the other.

This structure (Figure 7-14) is very compatible with the structure of the DRAW. The parallel application of data to all taps can be easily done through the use of global lines that extended through the chip length. The adder in every DRPU passes the carry signal to the neighboring DRPU. This eliminates the need for extra routing of the results.

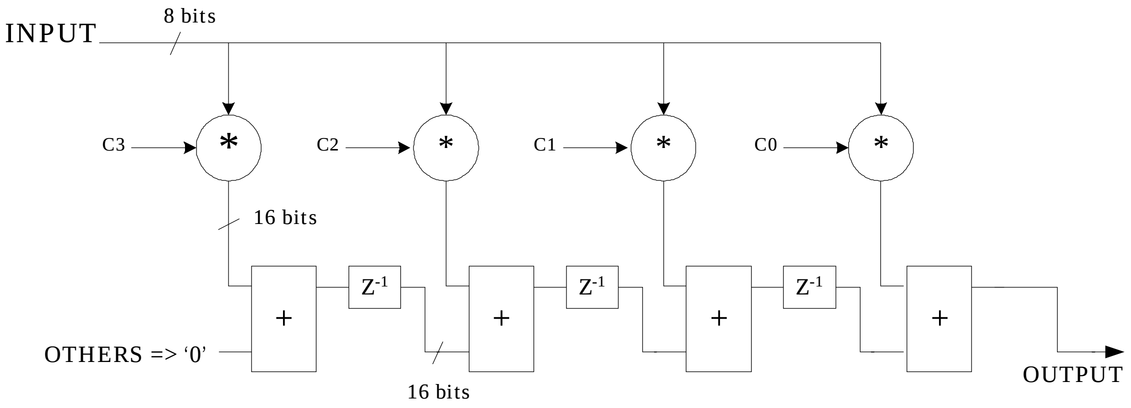


Figure 7-14: Transposed FIR filter implementation

Figure 7-15 is a Matlab code for a transposed FIR filter structure. A four tap example was simulated using the Matlab FIR function. The output for a stream of data is shown at the bottom of Figure 7-15.

The same FIR structure is mapped onto six DRPUs and symbolically shown in Figure 7-16. The simulation waveform of the mapped FIR onto six DRPUs is shown in Figure 7-17.

Based on the above mapping examples, the mapping process requires a detailed knowledge of the application algorithm and the DRAW resources. Since DRAW is designed for 3G and future mobile wireless applications, the complexity and size of the algorithms is expected to grow. Hence, the mapping process needs to be automated. Automating the mapping step will speed up the implementation and provide more than one implementation strategies for each algorithm. This is left for future research.

```
function [Y]=FIR(X,L,H)

Y(1)=0;

TT_ADD=[0 0 0 0];
X_reg=zeros(1,L);

for i=1:length(X)
    for j=1:length(H);
        X_reg(j)=X(i);
    end;
    TT=X_reg.*H;
    TT_SHIFT=[0 TT_ADD(1:L-1)];
    TT_ADD=TT+TT_SHIFT;

    Y(i+1)=TT_ADD(4);


end;
```

```
>> FIR([1 2 3 4 5 6 7 8 9 10 11 12 13 14],4,[4 3 2 1])

ans =

     0     1     4    10    20    30    40    50    60    70
80    90   100   110   120
```

Figure 7-15: A Matlab code of an FIR filter

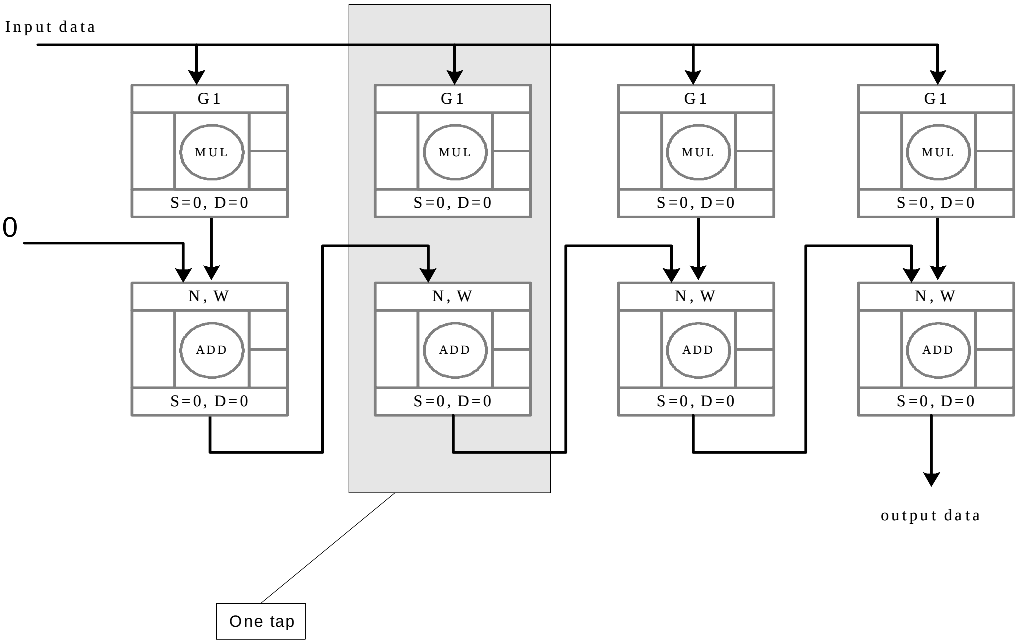
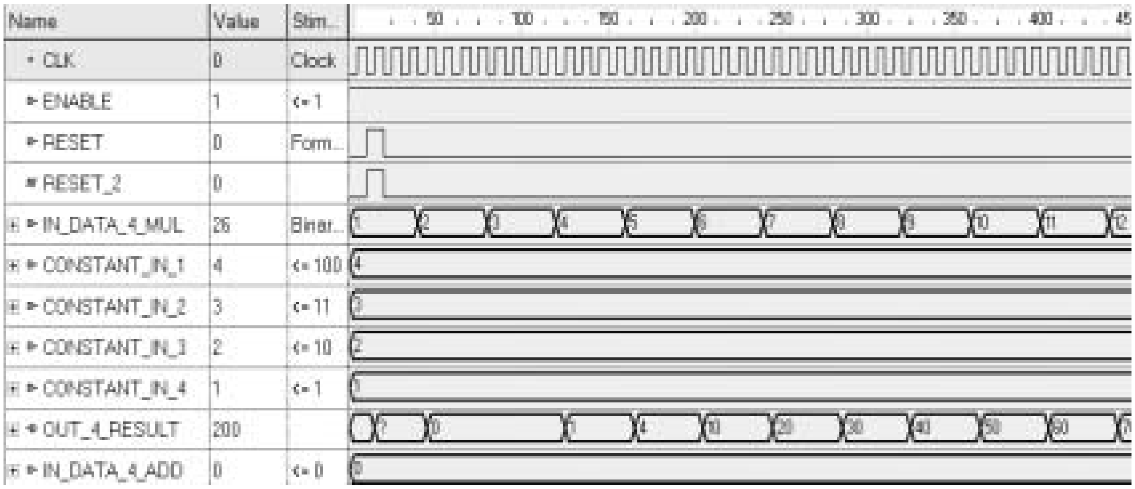


Figure 7-16: Symbolic representation of the mapping of 4-taps FIR filter onto DRAW



Name	Value	Stim...
CLK	0	Clock
ENABLE	1	<= 1
RESET	0	Form...
RESET_2	0	
IN_DATA_4_MUL	26	Binar...
CONSTANT_IN_1	4	<= 100
CONSTANT_IN_2	3	<= 11
CONSTANT_IN_3	2	<= 10
CONSTANT_IN_4	1	<= 1
OUT_4_RESULT	200	
IN_DATA_4_ADD	0	<= 0

Figure 7-17: Simulation waveform of a four-tap FIR filter mapped onto six DRPUs

Chapter 8

Conclusion and Recommendations

8.1 Conclusion

The result of this work was the development of a dynamically reconfigurable architecture specially designed and tailored toward the third generation wireless mobile systems. The architecture can be integrated with other components to form a configurable block in a system on a chip solution. The dynamic and fast reconfiguration of the architecture and it's highly parallel execution makes the dynamically reconfigurable architecture a promising solution for the future wireless mobile systems.

The dynamic configurable computing capability of the architecture has many advantages. A smaller physical size is possible as is a longer battery life since the needed applications are mapped to the architecture only when they are required.

The dynamic configuration process is extremely fast compared to the commercial FPGAs. The processing element can be configured/reconfigured in one clock cycle, i.e. 12 ns. This was accomplished by storing four configuration contexts (configuration bit set) next to the processing element. The four configuration contexts are cached while the processing element is running. This is accomplished by dedicating a routing bus network and not relying on the data routing network to transfer the configuration bits. One configuration set of the processing element is 64-bits long. It is partitioned into two sections of 32-bits each. The first part holds the configuration for the internal routing of the signal in the processing element. The second part holds the configuration of the operations of the internal components of the processing element. The configuration bits are routed in 8-bit segments. The 8-bit routing lines reduce the area requirement of the configuration routing network. The segmenting of the configuration set eliminates the need to transmit a complete configuration set if the new configuration set is similar to the previous configuration set.

The architecture is built around fast execution processing elements. The processing elements are repeated in a regular fashion to create a regular array processing structure. This regular structure is the ideal target for an IP-based method that will be developed later. The processing element contains a special ALU/Multiplier, dictated processing units for wireless applications, and a configurable storage component. The ALU is specifically designed to provide fast addition and subtraction operations, since these two operations dominate 70% of the total operations the baseband processing unit of the wireless application. A fast modified barrel shifter is available in the ALU. The modified barrel shifter is capable of rotating, logical shifts, and arithmetic shifts. A Booth multiplication procedure is then added to the ALU to provide the processing element with multiplication capability. The processing element contains special processing units. These special units are a configurable linear feedback shift register, and a configurable spreading data path. Implementing spreading/de-spreading, and code generation on a 16-bit ALU is a waste of resources. The two special components (CSDP and CLFSR) can efficiently implement such operations.

The implementation of the wireless mobile baseband unit requires the use of FIFO and RAM as storage elements. The processing unit has a configurable component that can be utilized as an FIFO or RAM.

Scaling and delaying the output signal during the execution time is necessary. The output of the processing element is a configurable scaling and delay unit. The unit can be configured to delay and/or scale the output signal or pass the signal without any delay. The amount of scale and delay are either provided by the configuration bits or can come from other active processing elements. The calculation of the delay and/or scale values by other running processing elements facilitate the requirement of a dynamic delaying and/or scaling.

The architecture is completely designed and documented in synthesizeable VHDL code. The architecture was synthesized using an 0.25 μm standard cell ASIC library. The performance of the synthesized architecture is sufficient to run the most demanding applications of the third generation mobile receiver. As an example, a turbo decoder which is compliant to the 3GPP standard was designed and mapped onto DRAW [71].

8.2 Recommendations for Future Work

Manual mapping is a limiting point to the ability to run many applications on the architecture. The manual method is long, difficult, and not optimized. A CAD tool that would automate the mapping process would be very useful.

As the area of dynamic reconfigurable computing matures, a need to design a dynamic reconfigurable architecture for each area of application will arise. Thereby providing a strong motivation to automate the generation of the specification of such architecture for each area.

When designing the architecture for the wireless mobile systems, power is of primary importance. Further optimization of the architecture is required to reduce the power consumption even further. Designing for power methodology is the next logical step in this project.

The architecture can be an ideal prototyping platform (especially in an academic community) for the wireless application in particular and for all communication systems in general. For that purpose, the architecture needs to be fabricated and integrated with other components as a PCI board. This will create an implementation bed for classroom experiments and research.

Appendix A

WCDMA Matlab Codes

This appendix contains parts of the MATLAB codes used to simulate the UMTS WCDMA wireless mobile system. The results of simulation are then used to design the DRAW architecture.

The complete set of codes are available at the following web address, password is granted upon request. ftp://www.ent.ohiou.edu/webcad/DRAW/matlab_codes/

1 Data Generation.

The purpose of this function is to generate binary data from a text or an image. The text is first converted to its binary ASCII code. The binary data is then segmented to fit one radio transmission frame. A spreading, filtering and QPSK modulation are then performed on the data stream.

```
clear;
%read a text
fid=fopen('text.txt','rt');
Txt=fscanf(fid,'%c');
fclose(fid);
```

```
%Convert txt to binary;
TL=length(Txt);
WW=txt2bin(Txt) ;
%Flip S so that the first data bit is in the right most position
%S=fliplr(S);
MM=bin2txt(WW);
%Convert to 1,-1 values.
%y=-2*x+1
%x=(y-1)/-2
S=-2*WW+1;
check=bin2txt((S-1)/-2);


%the K parameter determines how many bits are in one frame:
%num_bits_per_frame=10*2^K.
%the Spreading factor SF is related to K as: SF=512/2^K.
%Select spreading factor: min=4 (K=7 ==> Data rate is max=1920kbps)
%max=512 (k=0 ==> data rate is min=15kbps)

%let us select a Walsh code of length 4 for max data rate.
SF=4;

%generate a Walsh code.
wlsh_code=wlsh_gen(4,2);
wlsh_code=wlsh_code*(-2)+1;

%According to 3GPP standards, one slot (10ms/15 = .666ms) layout is as
follows:
% |--Data1--|--TPC--|--TFCI--|--Data2--|--pilot--|
% |  248    |   8   |   8    |   1000  |    16   |
%total bits =1280, SF=4 ==>num_chips=1280*4=5120chips/slot
%channel rate is 1280*15slot=1920 kbps

%to form a slot then a frame we need to break our data stream into
%%284-1000-284-1000... corresponding to Data1 and Data2.

%the other parts (TPC,TFCI,Pilot) of the slot will be filled with random
%binary values for the time being.

S=S(1:18720);
%construct the frame
j=1;
un_spd_FRAME=[];slot=[];
for i=1:15
```

```
   slot=[S(j:j+247),any_bits(8), any_bits(8),S(j+248:j+1247), 
any_bits(16)];
j=j+1248;

un_spd_FRAME=[un_spd_FRAME,slot];
end

%split the data to I and Q, then spread, conjugate the Q branch. and
then add.
I_brnch=[];
Q_brnch=[];

for i=1:2:19199
   I_brnch=[I_brnch,un_spd_FRAME(i)];
   Q_brnch=[Q_brnch,un_spd_FRAME(i+1)];
end

%SF=4 ==>num_chips=1280*15*4=76800chips/slot
%Spread the data;
spd_I=wlsh_code'*I_brnch; %(4*1)*(1*19200/2)=4*19200/2.=4*9600 matrix
spd_Q=wlsh_code'*Q_brnch;

%reshape the data to a stream of bits
[M,N]=size(spd_I);
L=M*N;
spd_I=reshape(spd_I,1,L); % (4*9600)=38400 chips long array
spd_Q=reshape(spd_Q,1,L); %

%Multiply Q branch by j.
spd_Q=spd_Q*-1;


%Scramble the frame by multiplying with a cell specific down link
%Scrambling code of length 38400. Let us select code number 500
[I_code,Q_code]=dl_sc_code(500);

%change it to bipolar;
I_code=I_code*(-2)+1;
Q_code=Q_code*(-2)+1;

sc_I_brnch=spd_I.*I_code;
sc_Q_brnch=spd_Q.*Q_code;

%create the final stream by complex addition. i.e. create a stream of
```

```
%38400 symbols, each symbol is 2bits (I+jQ);
sc_frame=[];
for i=1:2:38400
   sc_frame(i)=sc_I_brnch(i);
   sc_frame(i+1)=sc_Q_brnch(i);
end;


%QPSK modulation:
%Filter the I and Q branches. RRC Filter.
%[NUM_s, DEN_s] = RCOSINE(1, 4, 'sqrt',  0.22);
I_Fltd=(RCOSFLT(sc_I_brnch, 1, 4, 'sqrt', 0.22))'; %Array was 38400,
four times
Q_Fltd=(RCOSFLT(sc_Q_brnch, 1, 4, 'sqrt', 0.22))';% up sampled = 153600
chips.


%check point
check2=(RCOSFLT(I_Fltd, 1, 4, 'sqrt', 0.22))';
check2=[check2, zeros(1,8)];
check3=reshape(check2,16,length(check2)/16);
check4=sum(check3(:,:));
%their is a delay of 3 chips in the filter by default so
check5=check4(4:length(check4));
%change back to -1 1 format so
check6=round(check5);
check7=check6./(abs(check6));
if (check7(1:38397)==sc_I_brnch(1:38397))
    fprintf('correct');
end;
%end check point.


%multiply by the sin and cos of the carrier freq. fo=2050Mhz.
%38400 chips per 10 ms frame. So the data rate is 3.84Mcps.
%chip duration is 10ms/38400.  t=0 to 10ms. step 10/38400ms.
%t=0:(10e-3/153599):10e-3;
%save temp.mat I_Fltd Q_Fltd;
%clear;
%load temp.mat;
%>> t=0:3e-14:4.608e-9;
%>> y=cos(2*pi*2000e6*t);
%>> plot(t,y)
```

```
%one complete wave at this freq 2050e6
% takes 1/20.5e6 in 10msec. and we must over sample by 4.
%in addition we need to evaluate the multiplication only at >>
t1=0:(10e-1/(4*2050e5)):10e-3;
t1=0:3e-14:4.608e-9;
t=t1(1:153600);
I_Fltd=I_Fltd(1:153600);
Q_Fltd=Q_Fltd(1:153600);
I_mod=I_Fltd.*(cos(2*pi*2050e6*t));
Q_mod=Q_Fltd.*(-sin(2*pi*2050e6*t));


%feed the frame bits into the channel.
[chi,chq]= CHNL(I_mod,Q_mod,4,5);
```

2 Code Generation

The following functions generate different types of codes which are needed in the transmission and receiving of the data.

```
%DL Scrambling code Generator
%there are 2^18-1=262,143 codes. not all of them are used.
%generated code length is 38400 chips.
%input is the code number n, which correspond to Primary code when
%n=16*i i=0,...511. and to a secondary code when n=16*i+k k=1...15
%so only 8191 codes are used, of those 512 are primary (codes number
%0,16,32,48,64,...8176) and 7679 secondary codes (codes number 1,2,3
%,4,..15,17,18,...31,33,34,...8191).
%note: these are scrambling codes not synchronization codes.
function [INFO,I_code,Q_code]=dl_sc_code(n)

%if n>8191
 %  fprintf('generated code is an alternative code n> 8191 \n')
%end

%if (rem(n,16)==0&n<8191)
 %  fprintf('code is Primary \n');
%else if (rem(n,16)~=0&n<8191)
```

```
 %  fprintf('code is secondary \n');
%end
%end

%x M seq shift reg x(1)=1 others x(i)=0 i=2,..18.
%y Mseq shift reg y(i)=1

for i=1:18
   x(i)=0;
   y(i)=1;
end
x(1)=1;

for j=1:3840
x_fed_back_bit=rem((x(1)+x(8)),2);
y_fed_back_bit=rem((y(1)+y(6)+y(8)+y(11)),2);

Q_y=rem((sum(y(9:16))+y(5)+y(6)),2);
Q_x=rem((x(5)+x(7)+x(16)),2);

I_code(j)=xor(x(1),y(1));
Q_code(j)=xor(Q_y,Q_x);

x=[x(2:18),x_fed_back_bit];
y=[y(2:18),y_fed_back_bit];


end


-----------------------
-----------------------
%Primary Sync Code.
% this is a fixed code for all the system, and of length 256 chips.
function [INFO,PSC_I,PSC_Q]=pri_sync_code



a=[0 0 0 0 0 0 1 1 0 1 0 1 0 1 1 0 ];
PSC_I=[a a a not(a) not(a) a not(a) not(a) a a a not(a) a not(a) a a ];
PSC_Q=PSC_I;
```

```
-----------------------
-----------------------
%Spreading Code generator, code type M-seq.
function [M_code]=Mgen(poly_vector, num_chips)


%M_code is the generated M code of length num_chip
%reg_state returns the last loading of the reg, so we can continue generating the code later.
%poly_vector is the characteristic Polynomial, [1 0 1 1] is 1+x^2+x^3.
%inti_loading must not be all zeros.
reg_state=ones(1,length(poly_vector))

for i=1:num_chips
   temp_reg=poly_vector.*reg_state;
   temp_xor=temp_reg(length(poly_vector))
            for j=length(poly_vector)-1:-1:1
                   temp_xor=xor(temp_reg(j),temp_xor)
            end

         fed_bit=temp_xor
         M_code(i)= not(reg_state(length(poly_vector)))
         reg_state=[fed_bit, reg_state(1:length(poly_vector)-1) ]


      end
------------------
-------------------
%Code Allocation of SSC.
%this function will produce the group of SSC code to be mapped to one frame
%based on the selected group. see 3GPP 25-213
%input Group number (1-64), output a 15 SSCs each of length 256. in a frame of
%length 38400 chips, i.e. each code fall in the first 256 of each slot
%(2560 chip long).the remaining of the slot is filled with zeros.
function [INFO,SSCs_frame]=ssc_group_frame(group_num)


GRPS=[
1     1     2     8     9     10    15    8     10    16    2     7     15    7     16;
1     1     5     16    7     3     14    16    3     10    5     12    14    12    10;
```

```
1      2      1      15     5      5      12     16     6      11     2      16
11     15     12;
1      2      3      1      8      6      5      2      5      8      4      4
6      3      7;
1      2      16     6      6      11     15     5      12     1      15     12
16     11     2;
1      3      4      7      4      1      5      5      3      6      2      8
7      6      8;
1      4      11     3      4      10     9      2      11     2      10     12
12     9      3;
1      5      6      6      14     9      10     2      13     9      2      5
14     1      13;
1      6      10     10     4      11     7      13     16     11     13     6
4      1      16;
1      6      13     2      14     2      6      5      5      13     10     9
1      14     10;
1      7      8      5      7      2      4      3      8      3      2      6
6      4      5;
1      7      10     9      16     7      9      15     1      8      16     8
15     2      2;
1      8      12     9      9      4      13     16     5      1      13     5
12     4      8;
1      8      14     10     14     1      15     15     8      5      11     4
10     5      4;
1      9      2      15     15     16     10     7      8      1      10     8
2      16     9;
1      9      15     6      16     2      13     14     10     11     7      4
5      12     3;
1      10     9      11     15     7      6      4      16     5      2      12
13     3      14;
1      11     14     4      13     2      9      10     12     16     8      5
3      15     6;
1      12     12     13     14     7      2      8      14     2      1      13
11     8      11;
1      12     15     5      4      14     3      16     7      8      6      2
10     11     13;
1      15     4      3      7      6      10     13     12     5      14     16
8      2      11;
1      16     3      12     11     9      13     5      8      2      14     7
4      10     15;
2      2      5      10     16     11     3      10     11     8      5      13
3      13     8;
2      2      12     3      15     5      8      3      5      14     12     9
8      9      14;
2      3      6      16     12     16     3      13     13     6      7      9
2      12     7;
2      3      8      2      9      15     14     3      14     9      5      5
15     8      12;
```

```
2     4     7     9     5     4     9     11    2     14    5     14
11    16    16;
2     4     13    12    12    7     15    10    5     2     15    5
13    7     4;
2     5     9     9     3     12    8     14    15    12    14    5
3     2     15;
2     5     11    7     2     11    9     4     16    7     16    9
14    14    4;
2     6     2     13    3     3     12    9     7     16    6     9
16    13    12;
2     6     9     7     7     16    13    3     12    2     13    12
9     16    6;
2     7     12    15    2     12    4     10    13    15    13    4
5     5     10;
2     7     14    16    5     9     2     9     16    11    11    5
7     4     14;
2     8     5     12    5     2     14    14    8     15    3     9
12    15    9;
2     9     13    4     2     13    8     11    6     4     6     8
15    15    11;
2     10    3     2     13    16    8     10    8     13    11    11
16    3     5;
2     11    15    3     11    6     14    10    15    10    6     7
7     14    3;
2     16    4     5     16    14    7     11    4     11    14    9
9     7     5;
3     3     4     6     11    12    13    6     12    14    4     5
13    5     14;
3     3     6     5     16    9     15    5     9     10    6     4
15    4     10;
3     4     5     14    4     6     12    13    5     13    6     11
11    12    14;
3     4     9     16    10    4     16    15    3     5     10    5
15    6     6;
3     4     16    10    5     10    4     9     9     16    15    6
3     5     15;
3     5     12    11    14    5     11    13    3     6     14    6
13    4     4;
3     6     4     10    6     5     9     15    4     15    5     16
16    9     10;
3     7     8     8     16    11    12    4     15    11    4     7
16    3     15;
3     7     16    11    4     15    3     15    11    12    12    4
7     8     16;
3     8     7     15    4     8     15    12    3     16    4     16
12    11    11;
3     8     15    4     16    4     8     7     7     15    12    11
3     16    12;
```

```
3     10    10    15    16    5     4     6     16    4     3     15
9     6     9;
3     13    11    5     4     12    4     11    6     6     5     3
14    13    12;
3     14    7     9     14    10    13    8     7     8     10    4
4     13    9;
5     5     8     14    16    13    6     14    13    7     8     15
6     15    7;
5     6     11    7     10    8     5     8     7     12    12    10
6     9     11;
5     6     13    8     13    5     7     7     6     16    14    15
8     16    15;
5     7     9     10    7     11    6     12    9     12    11    8
8     6     10;
5     9     6     8     10    9     8     12    5     11    10    11
12    7     7;
5     10    10    12    8     11    9     7     8     9     5     12
6     7     6;
5     10    12    6     5     12    8     9     7     6     7     8
11    11    9;
5     13    15    15    14    8     6     7     16    8     7     13
14    5     16;
9     10    13    10    11    15    15    9     16    12    14    13
16    14    11;
9     11    12    15    12    9     13    13    11    14    10    16
15    14    16;
9     12    10    15    13    14    9     14    15    11    11    13
12    16    10;];

for j=1:2304
   Z(j)=0;
   end
ssc_frame=[];
for i=1:15
   [null,temp]=sec_sync_code(GRPS(group_num,i));
   ssc_frame=[ssc_frame, temp,Z];
end




-----------------------
------------------------
%Walsh code generator
%code_length is also called the SF.

function [INFO,wlsh_code]=wlsh_gen (code_length,code_num)
```

```
H=[0];
for i=1:log2(code_length)
   H=[H H; H not(H)];
end
wlsh_code=H(code_num,:);


---------------------
-----------------------
```

3 RAKE Receiver

```
function [INFO,Y]=sRAKE(X,PN,num_fings, intg_prd)
%spread and integrate
%X is the data victor. X length is a intg_prd * frame length.
%so that when we integrate over the period intg_prd we get one frame.



Y=[];
out=[];
for i=1:num_fings
   out(i,:)=X.*PN;
PN=[0, PN(1:length(PN)-1)];
temp1=1;
temp2=intg_prd;

   for j=1:(length(X)/intg_prd)
      Y(i,j)=sum(out(i,temp1:temp2));
      temp1=temp2+1;
      temp2=temp2+intg_prd;

   end;
end;
--------------------
--------------------
```

4 Searcher

```
function [INFO,LOC]=sercher(X,PN)


%search the input sequence for a match of the PN code segment. when a
match is found
%the function return the location of the first bit of the code.
%EXAMPLE:
X=-2*round(rand(1,200))+1; PN=X(23:87);
XD=[X,X];
PEAK=[];
LN=length(PN); %assuming that PN is shorter than X
thr=90/100*LN;  %if 90% of the PN matches X then it is a peak.

for i=1:length(X)
    i;
    Peak=sum(XD(i:LN+i-1).*PN);
    if Peak > thr
        PEAK=[PEAK, Peak];
    end;
end;


LOC=PEAK;
```

5 Channel Estimation

```
function [INFO,delay,
peak_value]=Coars_CH_EST(Data,SC_CODE,Num_of_paths,T_PRCNT,Window_wid
th)

Data=round(rand(1,38400/400))*(-2)+1;
%SC_CODE=dl_sc_code(floor(rand*100));
load dl_scCODE;
SC_CODE=SCCODE;
%truncate the code to the required length
SC_CODE=SC_CODE(1:38400/400); ;
SC_CODE=SC_CODE*(-2)+1;
```

```
Num_of_paths=3;
T_PRCNT=30;
window_width=38400/400;

%Spread the data
Data_sent=SC_CODE.*Data;


%generate the paths
Data_path=[];Data_all_paths=zeros(1,196);delay_locations=[];Data_spre
ad_noise=[];

for k=1:Num_of_paths
    delay=floor(100*rand);
    delay_locations=[delay_locations, delay];
    Data_path=[(((round(rand(1,delay)))*(-2))+1) ,  Data_sent ,
(((round(rand(1,100-delay)))*(-2))+1)];

    %Add AWGN
    Noise=zeros(1,196);%round(rand(1,100+(38400/400)));
    div=div+1;
    Data_spread_noise=Data_path+Noise;

    %Create Data matrix
    Data_all_paths=[Data_all_paths ; Data_spread_noise];
end;

%Add the paths to gather (At the receiver)
Received_Data=sum(Data_all_paths(:,:));

Peaks=[];Data_spread=[Data_sent, (round(rand(1,100))*(-2)+1)];

for j=1:96
    Corr=Data_spread.*Received_Data;
    Sum_Corr=sum(Corr(j:j+95));
    Peaks=[Peaks, Sum_Corr];
    %Shift Data_spread
    Data_spread=[(((round(rand(1,j)))*(-2))+1) , Data_sent,
(((round(rand(1,100-j)))*(-2))+1)];

end;

  %stem(Peaks);
```

6 Maximal Ratio Combining

```
function [INFO,R]=m_r_comp(X_mlty_paths,num_fings)

%X_mlty_paths is the matrix containing the multipaths signals.
%number of rows must equal number of fingers. it is equivalent to the
output
% of sRAKE function Y.
%R recovered signal.
R=[];
power=ones(num_fings,length(X_mlty_paths(1,:)));
for i=1:num_fings
   power(i)=sum(X_mlty_paths(i,:).*X_mlty_paths(i,:));
end

for i=1:num_fings
   X_mlty_paths(i,:)=X_mlty_paths(i,:)*power(i)/max(power);
end


R=sign(sum(X_mlty_paths));
```

Appendix B

DRAW Architecture VHDL Codes

This appendix contains parts of the VHDL codes of the DRAW architecture. All the codes are synthesizesable. The first part of this appendix contains the DRPU top level code which shows the entities of all the components inside the DRPU. The second section contains the entities of the other components outside the DRPU. These components are: the Communication and Switching Unit (CSU), the Configuration Memory Unit (CMU), the Dedicated I/O (DIO), and the Switching Box (SWB) model.

The complete set of codes are available at the following web address, password is granted upon request. ftp://www.ent.ohiou.edu/webcad/DRAW/vhdl_codes/

1 DRPU Top Level

```
----------------------------------------------------------------
----------------------------------------------------------------
--
--  Project            : DRAW
--  File name          : DRPU_TOP_LEVEL.vhd
--  Title              : The Dynamically Reconfigurable Processing
Unit (DRPU)
--  Description        : The processing unit of the DRAW architecture
--                     :
--  Design Libray      :
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
---------------------------------------------------------------------
--   Revisions   :
-- Date     Author      Revision         Comments
-- 4/15/02  A. Alsolaim   Rev 0
--
---------------------------------------------------------------------
---------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;


entity DRPU_TOP_LEVEL is
      generic ( DRPU_CONFIG_BITS_WIDTH: integer:=64;
            DATA_PATH_WIDTH: integer:=16; RAM_ADRS_WIDTH: integer:= 3
);
      port (
            -- Global inputs
            -- Configuration bits input port is shown under "inputs from
CSU"
            CLK: in std_logic;
            CARRY_IN_FROM_E:in std_logic;
            CARRY_IN_FROM_N:in std_logic;
            CARRY_IN_FROM_S:in std_logic;
            CARRY_IN_FROM_W:in std_logic;


            SRART_HOLD_FROM_E: in std_logic;
            SRART_HOLD_FROM_N: in std_logic;
            SRART_HOLD_FROM_S: in std_logic;
            SRART_HOLD_FROM_W: in std_logic;

            --Data lines from global communication channels
            IN_G_1_BUS: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');
```

```
            IN_G_2_BUS: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');

            -- Data lines from neighboring RPU
            IN_FROM_E_RPU: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0):=(others=>'0');
            IN_FROM_N_RPU: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0):=(others=>'0');
            IN_FROM_S_RPU: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0):=(others=>'0');
            IN_FROM_W_RPU: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0):=(others=>'0');
            --- End inputs to inface
            -- External direct Inputs and outputs to/from CLFSR
            CLFSR_IN_1_LFT: in std_logic;
            CLFSR_IN_1_RHT: in std_logic;

            CLFSR_OUT_1_LFT:out std_logic;
            CLFSR_OUT_1_RGT: out std_logic;
            --END inputs/outputs to CLFR
            -- Inputs/outputs from/to the CSU (to the DRPU controller)
            CONFIGURATION_BITS: in STD_LOGIC_VECTOR
(DRPU_CONFIG_BITS_WIDTH-1 downto 0):=(others=>'0');
            GO_CONFIG: in STD_LOGIC;
            GO_HOLD: in STD_LOGIC;
            GO_RUN: in STD_LOGIC;
            GO_SLEEP: in STD_LOGIC;
            DRPU_DONE: out STD_LOGIC;

            -- outputs from the OUTINFACE
            RPU_OUT_1_GBUS : out std_logic_vector ( DATA_PATH_WIDTH-1
downto 0):=(others=>'0');
            RPU_OUT_2_GBUS : out std_logic_vector ( DATA_PATH_WIDTH-1
downto 0):=(others=>'0');

            IN_E_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');
            IN_N_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');
            IN_S_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');
            IN_W_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');

            RPU_START_HOLD_E: out std_logic; --'1' start, '0' hold
            RPU_START_HOLD_N: out std_logic; --'1' start, '0' hold
            RPU_START_HOLD_S: out std_logic; --'1' start, '0' hold
            RPU_START_HOLD_W: out std_logic --'1' start, '0' hold
            );
end DRPU_TOP_LEVEL;


Architecture BEHAV of DRPU_TOP_LEVEL is

-- Configuration layout table:
--Unit Name      Number of BitsConfiguration Bits
--
--Input Interface5                 0 to 4
--Output Interface12               5 to 16
--DRAP Interface5                  17 to 21
--RAM_FIFO Interface6              22 to 27
--CLFSR Interface2                 28 to 29
--CSDP Interface3                  30 to 32
--RAP Operations22                 33 to 54
--RAM_FIFO Operations1             55
--CLFSR Operations8                56 to 63
--TOTAL BITS           64          0 to 63

Component  AND_GATE
      port (
            I_IN0 : in STD_LOGIC;
            I_IN1 : in STD_LOGIC;
            O_OUT : out STD_LOGIC
      );
end component;

component CLFSR
      Generic(CLFSR_CONFIG_BITS_WIDTH: integer :=8;DATA_PATH_WIDTH:
integer:=16);
      port(
            CLFSR_CLK: in std_logic; --Clock
            CLFSR_CLR: in std_logic; --Clear
            CLFSR_ENABLE: in std_logic; -- Enable

            CLFSR_CONFIG_BITS: in
std_logic_vector(CLFSR_CONFIG_BITS_WIDTH-1 downto 0);
            CLFSR_IN_16: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0);
            CLFSR_IN_1_LFT: in std_logic;
            CLFSR_IN_1_RHT: in std_logic;
            CLFSR_OUT_16: out  std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            CLFSR_OUT_1_LFT:out std_logic;
            CLFSR_OUT_1_RGT: out std_logic;
            NEW_DATA_ARVD_FROM_CLFSR: out std_logic

            );
end component;

component CLFSR_INFACE
      generic (CLFSR_INFACE_CONFIG_BITS_WIDTH: integer:=2;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------

            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_2: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_3: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);

            CLFSR_INFACE_CONFIG_BITS: in std_logic_vector
(CLFSR_INFACE_CONFIG_BITS_WIDTH-1 downto 0);
            ---------------- OUTPUT SIGNALS-----------------------

            -- data Out lines. PN code in 16 bits word every 16 cycles
            CLFSR_INPUT: out std_logic_vector(DATA_PATH_WIDTH-1 downto
0)
            );

end component;

component SPRD_INFACE
      generic (SPRD_INTR_FC_CONFIG_BITS_WIDTH: integer:=3;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------

            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
```

```
            RPU_IN_FACE_2: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_3: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);

            SPRD_INTR_FC_CONFIG_BITS: in std_logic_vector
(SPRD_INTR_FC_CONFIG_BITS_WIDTH-1 downto 0);
            ---------------- OUTPUT SIGNALS-----------------------

            -- data Out lines, Data, and PN
            SPRD_DATA: out std_logic_vector(DATA_PATH_WIDTH-1 downto
0);
            SPRD_PN: out std_logic_vector(DATA_PATH_WIDTH-1 downto 0)

            );

end component;

component SPRD_UNIT
      generic(DATA_PATH_WIDTH : integer:=16);
      port (
            SPRD_CLK : in STD_LOGIC;  --Clock
            SPRD_ENABLE : in STD_LOGIC;  --Enable
            SPRD_RESET: in std_logic;
            DATA_IN : in STD_LOGIC_VECTOR (DATA_PATH_WIDTH-1 downto 0);
            PN1: in std_logic_vector (DATA_PATH_WIDTH-1 downto 0); --
PN1 is converted inside the CSDP
            --unit to serial stream PN1_SRL.
            SPRD_OUT : out STD_LOGIC_VECTOR (DATA_PATH_WIDTH-1 downto
0)

            );
end component ;


component  IN_FACE
      generic (IN_FACE_CONFIG_BITS_WIDTH: integer:=5;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------
            --Control signals
            IN_FACE_CONFIG_BITS: in
std_logic_vector(IN_FACE_CONFIG_BITS_WIDTH-1 downto 0);
            INFACE_RESET: in std_logic;  --Active high
            CLK: in std_logic;
            CARRY_IN_FROM_N:in std_logic;
            CARRY_IN_FROM_S:in std_logic;
            CARRY_IN_FROM_E:in std_logic;
            CARRY_IN_FROM_W:in std_logic;


            SRART_HOLD_FROM_N: in std_logic;
            SRART_HOLD_FROM_S: in std_logic;
            SRART_HOLD_FROM_W: in std_logic;
            SRART_HOLD_FROM_E: in std_logic;

            --Data lines from global communication channels
            IN_G_1_BUS: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0);
            IN_G_2_BUS: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0);

            -- Data lines from neighboring RPU
            IN_N_RPU: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
            IN_S_RPU: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
            IN_W_RPU: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
            IN_E_RPU: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);


            ---------------- OUTPUT SIGNALS-----------------------
            --Done signals from neighboring RPU to local control

            RPU_START_HOLD_FROM_X: out std_logic_vector(3 downto 0); -
-START from NSWE respectively

            RPU_CARRY_IN_1:out std_logic;
            RPU_CARRY_IN_2:out std_logic;

            -- Data lines selected from neighboring RPU and global going
to the RAPS
            RPU_IN_FACE_1: inout std_logic_vector( DATA_PATH_WIDTH-1
downto 0); --changed to inout to enable the new_data process to read
them.
            RPU_IN_FACE_2: inout std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_3: inout std_logic_vector( DATA_PATH_WIDTH-1
downto 0);

            NEW_DATA_ARVD_1: out std_logic:='0';
            NEW_DATA_ARVD_2: out std_logic:='0';
            NEW_DATA_ARVD_3: out std_logic:='0'

            );
end component;

component  OUT_FACE
      generic ( OUT_FACE_CONFIG_BITS_WIDTH: integer:=12;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------
            --Control signals
            TO_OTHER_RPU_START :in std_logic;

            OUT_FACE_CLK:in std_logic;

            RAP_1_OUT   : in std_logic_vector(DATA_PATH_WIDTH-1 downto
0);
            RAM_A_OUT   : in std_logic_vector(DATA_PATH_WIDTH-1 downto
0);
            --          RAM_1_B_OUT : in
std_logic_vector(DATA_PATH_WIDTH-1 downto 0);

                  --FROM CLFSR and CSDP
            CLFSR_OUT_16: in  std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            SPRD_OUT : in STD_LOGIC_VECTOR (DATA_PATH_WIDTH-1 downto 0)
;
            --line directly from input
            RPU_IN_FACE_1: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_2: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);

            -- Configuration bits;
            OUT_FACE_CONFIG_BITS: in
std_logic_vector(OUT_FACE_CONFIG_BITS_WIDTH-1 downto 0);

            -------------------------- OUTPUT SIGNALS ---------------
            RPU_OUT_1_GBUS : out std_logic_vector ( DATA_PATH_WIDTH-1
downto 0);
            RPU_OUT_2_GBUS : out std_logic_vector ( DATA_PATH_WIDTH-1
downto 0);
```

```
            IN_N_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto
0);
            IN_S_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto
0);
            IN_W_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto
0);
            IN_E_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto
0);

            RPU_START_HOLD_E: out std_logic; --'1' start, '0' hold
            RPU_START_HOLD_W: out std_logic; --'1' start, '0' hold
            RPU_START_HOLD_N: out std_logic; --'1' start, '0' hold
            RPU_START_HOLD_S: out std_logic --'1' start, '0' hold

            );
end component;


component  RAP_INTR_FC
      generic (RAP_INTR_FC_CONFIG_BITS_WIDTH: integer:=5;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------

            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_2: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_3: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);


            RPU_CARRY_IN_1:in std_logic;
            RPU_CARRY_IN_2:in std_logic;

            -- data lines from the RAP
            FROM_RAP: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);

            -- data lines from RAMs
            FROM_RAM_A: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0);

            --data from SPRD unit
            SPRD_OUT : in STD_LOGIC_VECTOR (DATA_PATH_WIDTH-1 downto
0);

            RAP_INTR_FC_CONFIG_BITS: in std_logic_vector
(RAP_INTR_FC_CONFIG_BITS_WIDTH-1 downto 0);
            ---------------- OUTPUT SIGNALS-----------------------

            -- data lines, X, and Y
            RAP_X_IN: out std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
            RAP_Y_IN: out std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
            RAP_CRY_IN: out std_logic
            );
end component;

component RAM_FIFO
            generic (DATA_PATH_WIDTH     :natural := 16
;RAM_CONFIG_BITS_WIDTH: integer :=1;--RAM width
            RAM_ADRS_WIDTH  :natural := 3--RAM depth =
2^RAM_ADRS_WIDTH.
            );
      port (
            RAM_A_IN : in std_logic_vector (DATA_PATH_WIDTH-1 downto
0);
            --RAM_B_IN : in std_logic_vector (DATA_PATH_WIDTH-1 downto
0);

            RAM_A_ADRS: in std_logic_vector (RAM_ADRS_WIDTH-1 downto
0);
            RAM_B_ADRS : in std_logic_vector (RAM_ADRS_WIDTH-1 downto
0); -- Port B is only used for reading.

            RAM_WR: in std_logic;-- 0 no op, 1 write. --Port A
            RAM_RD: in std_logic;-- 0 no op, 1 READ.   --Port A
            RAM_ENABLE: in std_logic;  -- synchronous, '1' enabled.
            RAM_CLEAR: in std_logic; -- synchronous Clear
            RAM_CONFIG_BITS:instd_logic:='0';
            --'0' for RAM behavior and
            -- '1' for FIFO behavior

            RAM_CLK: in std_logic;

            FROM_RAM_FIFO_FULL: out std_logic; --'1' for full
            FROM_RAM_FIFO_EMPTY: out std_logic; --'1' for empty
            FROM_RAM_A_OUT :out std_logic_vector (DATA_PATH_WIDTH-1
downto 0)
```

```
            --    FROM_RAM_B_OUT :out std_logic_vector
(DATA_PATH_WIDTH-1 downto 0)
            );

end component;

component  RAM_FIFO_FACE
      generic (RAM_FIFO_FACE_CONFIG_BITS_WIDTH: integer:=6;
            DATA_PATH_WIDTH: integer:=16; RAM_ADRS_WIDTH:integer:=3);
      port (
            ---------------- INPUT SIGNALS-----------------------
            RAM_FIFO_INFACE_CLK:in std_logic;

            --Data lines from RAP
            RAP_OUT: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_2: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_3: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            -- Data from the CLFSR
            CLFSR_OUT_16: in STD_LOGIC_VECTOR (DATA_PATH_WIDTH-1 downto
0);

            NEW_DATA_ARVD_1: in std_logic;
            NEW_DATA_ARVD_2: in std_logic;
            NEW_DATA_ARVD_3: in std_logic;
            NEW_DATA_ARVD_FROM_RAP: in std_logic; -- the Done_runing
signal out of the RAP
            NEW_DATA_ARVD_FROM_CLFSR: in std_logic;
            --configuration bits
            RAM_FACE_CONFIG_BITS: in
std_logic_vector(RAM_FIFO_FACE_CONFIG_BITS_WIDTH-1 downto 0);

            ---------------- OUTPUT SIGNALS-----------------------
            --CLK to the RAM

            --Data and ADRS signal to RAM
            RAM_A_IN : out std_logic_vector (DATA_PATH_WIDTH-1 downto
0) ;
            RAM_A_ADRS_OUT: out std_logic_vector (RAM_ADRS_WIDTH-1
downto 0);
            RAM_B_ADRS_OUT: out std_logic_vector (RAM_ADRS_WIDTH-1
downto 0);
```

```
            NEW_DATA_ARVD_2RAM: out std_logic

            );
end component;


component RPU_CTRL_NEW
      port (
            CONFIGURATION_BITS: in STD_LOGIC_VECTOR (63 downto 0);
            CTRL_CLK: in STD_LOGIC;
            FROM_RAM_FIFO_EMPTY: in STD_LOGIC;
            FROM_RAM_FIFO_FULL: in STD_LOGIC;
            GO_CONFIG: in STD_LOGIC;
            GO_HOLD: in STD_LOGIC;
            GO_RUN: in STD_LOGIC;
            GO_SLEEP: in STD_LOGIC;
            NEW_DATA_ARVD_2RAM: in STD_LOGIC;
            RESET_CTROL: in STD_LOGIC;
            RPU_START_HOLD_FROM_X: in STD_LOGIC_VECTOR (3 downto 0);
            DISABLE_CLK: out STD_LOGIC;
            DRPU_DONE: out STD_LOGIC;
            DRPU_FULL_CONFIG_BITS: out STD_LOGIC_VECTOR (63 downto 0);
            DRPU_START_HOLD: out STD_LOGIC;
            ENABLE_DRPU: out STD_LOGIC;
            RAM_RD: out STD_LOGIC;
            RAM_WR: out STD_LOGIC;
            RESET_DRPU: out STD_LOGIC;
            START_NEXT_DRPU: out STD_LOGIC
            );
end component ;


component  RAP16
      generic ( RAP_CONFIG_BITS_WIDTH: integer:=22;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------
            RAP_CLK: in std_logic; --Active high
            RAP_ENABLE: in std_logic; --Active high
            RAP_RESET: in std_logic;  --Active high
            RAP_CONFIG_BITS: in std_logic_vector(
RAP_CONFIG_BITS_WIDTH -1 downto 0);
```

```
            RAP_X_IN: in std_logic_vector (DATA_PATH_WIDTH -1 downto
0);
            RAP_Y_IN: in std_logic_vector (DATA_PATH_WIDTH -1 downto
0);
            RAP_CRY_IN: in std_logic;

            RAP_CARY_OUT : out std_logic;
            RAP_OVR_FLW : out std_logic;

            RAP_DONE_RUNING: out std_logic;--'1' done, '0' Running
            RAP_OUT: out std_logic_vector (DATA_PATH_WIDTH -1 downto 0)
                        );


end component;

signal DRPU_CLK: std_logic;
signal DRPU_ENABLE:  std_logic;
signal DRPU_RESET:  std_logic;
signal RPU_START_HOLD_FROM_X:  std_logic_vector(3 downto 0);
signal RPU_CARRY_IN_FROM_N: std_logic;
signal RPU_CARRY_IN_FROM_S: std_logic;
signal RPU_CARRY_IN_FROM_E: std_logic;
signal RPU_CARRY_IN_FROM_W: std_logic;
signal RPU_IN_FACE_1:  std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
signal RPU_IN_FACE_2:  std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
signal RPU_IN_FACE_3:  std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
signal NEW_DATA_ARVD_1:  std_logic:='0';
signal NEW_DATA_ARVD_2:  std_logic:='0';
signal NEW_DATA_ARVD_3:  std_logic:='0';
signal CLFSR_IN_16:  std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
signal CLFSR_OUT_16:   std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
signal NEW_DATA_ARVD_FROM_CLFSR:  std_logic;
signal FROM_RAP:  std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
signal FROM_RAM_A:  std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
signal SPRD_OUT :  STD_LOGIC_VECTOR (DATA_PATH_WIDTH-1 downto 0);
signal RAP_X_IN:  std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
signal RAP_Y_IN:  std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
signal RAP_CARY_OUT :  std_logic;
signal RAP_OVR_FLW :  std_logic;
signal RAP_DONE_RUNING:  std_logic;--'1' done, '0' Running
signal RAP_OUT:  std_logic_vector (DATA_PATH_WIDTH -1 downto 0);
signal RPU_CARRY_IN_1: std_logic;
signal RPU_CARRY_IN_2: std_logic;
signal RAP_CRY_IN: std_logic;
signal SPRD_DATA:  std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
signal SPRD_PN:  std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
signal RAM_A_IN :  std_logic_vector (DATA_PATH_WIDTH-1 downto 0) ;
signal RAM_A_ADRS_OUT:  std_logic_vector (RAM_ADRS_WIDTH-1 downto 0);
signal RAM_B_ADRS_OUT:  std_logic_vector (RAM_ADRS_WIDTH-1 downto 0);
signal NEW_DATA_ARVD_2RAM:  std_logic;
signal RAM_WR:  std_logic;-- 0 no op, 1 write. --Port A
signal RAM_RD:  std_logic;-- 0 no op, 1 READ.   --Port A
signal RAM_ENABLE:  std_logic;  -- synchronous, '1' enabled.
signal RAM_CLEAR:  std_logic; -- synchronous Clear
signal RAM_CONFIG_BITS:std_logic:='0';
signal FROM_RAM_FIFO_FULL:  std_logic; --'1' for full
signal FROM_RAM_FIFO_EMPTY:  std_logic; --'1' for empty
signal FROM_RAM_A_OUT : std_logic_vector (DATA_PATH_WIDTH-1 downto 0);
signal DISABLE_CLK:  STD_LOGIC;
signal DRPU_FULL_CONFIG_BITS:  STD_LOGIC_VECTOR (63 downto 0);
signal DRPU_START_HOLD:  STD_LOGIC;
signal START_NEXT_DRPU:  STD_LOGIC;


begin
            U0_AND_GATE: AND_GATE
      port map (
            I_IN0  =>DISABLE_CLK ,
            I_IN1  => CLK,
            O_OUT => DRPU_CLK
      );

      U1_INPUT_INTERFACE:   IN_FACE

      port map (

      ---------------- INPUT SIGNALS-----------------------
      --Control signals
      IN_FACE_CONFIG_BITS => CONFIGURATION_BITS(4 downto 0)  ,
      INFACE_RESET => DRPU_RESET,  --Active high
      CLK                      => DRPU_CLK,
      CARRY_IN_FROM_N=>CARRY_IN_FROM_N ,
      CARRY_IN_FROM_S=> CARRY_IN_FROM_S,
      CARRY_IN_FROM_E=> CARRY_IN_FROM_E,
      CARRY_IN_FROM_W=> CARRY_IN_FROM_W,
      SRART_HOLD_FROM_N=> SRART_HOLD_FROM_N,
      SRART_HOLD_FROM_S=>SRART_HOLD_FROM_S ,
      SRART_HOLD_FROM_W=>SRART_HOLD_FROM_W,
      SRART_HOLD_FROM_E=>SRART_HOLD_FROM_E,

      --Data lines from global communication channels
      IN_G_1_BUS=>IN_G_1_BUS ,
      IN_G_2_BUS=> IN_G_2_BUS,

      -- Data lines from neighboring RPU
      IN_N_RPU=>IN_FROM_N_RPU,
      IN_S_RPU=>IN_FROM_S_RPU,
      IN_W_RPU=>IN_FROM_W_RPU ,
      IN_E_RPU=>IN_FROM_E_RPU ,


      ---------------- OUTPUT SIGNALS-----------------------
      --Done signals from neighboring RPU to local control

      RPU_START_HOLD_FROM_X=>RPU_START_HOLD_FROM_X, -- to be connected
to the controller

      RPU_CARRY_IN_1 => RPU_CARRY_IN_1,
      RPU_CARRY_IN_2 => RPU_CARRY_IN_2,

      -- Data lines selected from neighboring RPU and global going to
the RAPS
      RPU_IN_FACE_1=>RPU_IN_FACE_1, --changed to inout to enable the
new_data process to read them.
      RPU_IN_FACE_2=>RPU_IN_FACE_2,
      RPU_IN_FACE_3=>RPU_IN_FACE_3,

      NEW_DATA_ARVD_1=>NEW_DATA_ARVD_1,
      NEW_DATA_ARVD_2=>NEW_DATA_ARVD_2,
      NEW_DATA_ARVD_3=>NEW_DATA_ARVD_3

);




 U2_CLFSR: CLFSR_INFACE

      port map (
            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1 => RPU_IN_FACE_1,
            RPU_IN_FACE_2 => RPU_IN_FACE_2,
            RPU_IN_FACE_3 => RPU_IN_FACE_3 ,

            CLFSR_INFACE_CONFIG_BITS => CONFIGURATION_BITS(29 downto
28)  ,
            ---------------- OUTPUT SIGNALS-----------------------
            -- data Out lines.
            CLFSR_INPUT =>CLFSR_IN_16
            );




U3_CLFSR: CLFSR
--    Generic map(CLFSR_CONFIG_BITS_WIDTH: integer :=8;
DATA_PATH_WIDTH: integer:=16);
port map(
CLFSR_CLK=>DRPU_CLK , --Clock
CLFSR_CLR => T_RESET, --DRPU_RESET, --Clear
CLFSR_ENABLE => T_ENABLE, -- DRPU_ENABLE , -- Enable

CLFSR_CONFIG_BITS => CONFIGURATION_BITS(63 downto 56)  ,
CLFSR_IN_16 => CLFSR_IN_16,    -- from clfsr inface
CLFSR_IN_1_LFT =>CLFSR_IN_1_LFT,
CLFSR_IN_1_RHT =>  CLFSR_IN_1_RHT ,

CLFSR_OUT_16 =>CLFSR_OUT_16 ,--to RAM
CLFSR_OUT_1_LFT => CLFSR_OUT_1_LFT,
CLFSR_OUT_1_RGT => CLFSR_OUT_1_RGT,
NEW_DATA_ARVD_FROM_CLFSR  => NEW_DATA_ARVD_FROM_CLFSR    -- to con-
troll unit

);


U4_RAP_INFACE: RAP_INTR_FC

      port map(
            ---------------- INPUT SIGNALS-----------------------
            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1 =>RPU_IN_FACE_1 ,
            RPU_IN_FACE_2 => RPU_IN_FACE_2,
            RPU_IN_FACE_3 =>RPU_IN_FACE_3,

            RPU_CARRY_IN_1=>  RPU_CARRY_IN_1 ,
            RPU_CARRY_IN_2=> RPU_CARRY_IN_2,

            -- data lines from the RAP
            FROM_RAP =>  FROM_RAP,
            -- data lines from RAMs
            FROM_RAM_A =>FROM_RAM_A ,
            --data from SPRD unit
            SPRD_OUT  =>SPRD_OUT,
            RAP_INTR_FC_CONFIG_BITS =>  CONFIGURATION_BITS(21 downto
17)  ,
            ---------------- OUTPUT SIGNALS-----------------------
            -- data lines, X, and Y
            RAP_X_IN => RAP_X_IN,
            RAP_Y_IN => RAP_Y_IN ,

            RAP_CRY_IN => RAP_CRY_IN
            );


U5_RAP_UNIT: RAP16

      port map(

            ---------------- INPUT SIGNALS-----------------------
            RAP_CLK => DRPU_CLK,
            RAP_ENABLE => T_ENABLE,
            RAP_RESET => T_RESET,

            RAP_CONFIG_BITS =>  CONFIGURATION_BITS(54 downto 33)  ,
            RAP_X_IN => RAP_X_IN ,
            RAP_Y_IN =>RAP_Y_IN,
            RAP_CRY_IN =>RAP_CRY_IN ,


            RAP_CARY_OUT  =>RAP_CARY_OUT ,
            RAP_OVR_FLW  => RAP_OVR_FLW,

            RAP_DONE_RUNING =>RAP_DONE_RUNING,
            RAP_OUT => RAP_OUT
            );

 U6_CSDP_INTERFACE: SPRD_INFACE

      port map (
            ---------------- INPUT SIGNALS-----------------------
            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1 => RPU_IN_FACE_1 ,
            RPU_IN_FACE_2 => RPU_IN_FACE_2 ,
            RPU_IN_FACE_3 => RPU_IN_FACE_3 ,

            SPRD_INTR_FC_CONFIG_BITS => CONFIGURATION_BITS(32 downto
30) ,
            ---------------- OUTPUT SIGNALS-----------------------
            -- data Out lines, Data, and PN
            SPRD_DATA => SPRD_DATA ,
            SPRD_PN => SPRD_PN

            );



  U7_CSDP: SPRD_UNIT

      port map(
            SPRD_CLK  =>  DRPU_CLK,   --Clock
            SPRD_ENABLE   => DRPU_ENABLE,   --Enable
            SPRD_RESET  =>  DRPU_RESET,
            DATA_IN  =>  SPRD_DATA     ,  -- from sprd inface
            PN1  =>   SPRD_PN    ,  -- from sprd inface
            SPRD_OUT  =>  SPRD_OUT      -- to rap inface

            );

U8_RAM_INFACE:  RAM_FIFO_FACE

      port map(
            ---------------- INPUT SIGNALS-----------------------
            RAM_FIFO_INFACE_CLK  => DRPU_CLK ,

            --Data lines from RAP
            RAP_OUT  =>RAP_OUT  ,
            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1  => RPU_IN_FACE_1 ,
            RPU_IN_FACE_2  =>RPU_IN_FACE_2  ,
            RPU_IN_FACE_3  => RPU_IN_FACE_3 ,
            -- Data from the CLFSR
            CLFSR_OUT_16  =>  CLFSR_OUT_16,

            NEW_DATA_ARVD_1  => NEW_DATA_ARVD_1 ,
            NEW_DATA_ARVD_2  => NEW_DATA_ARVD_2 ,
            NEW_DATA_ARVD_3  => NEW_DATA_ARVD_3 ,
            NEW_DATA_ARVD_FROM_RAP  => RAP_DONE_RUNING ,
NEW_DATA_ARVD_FROM_CLFSR  =>NEW_DATA_ARVD_FROM_CLFSR  ,
            --configuration bits
            RAM_FACE_CONFIG_BITS  =>  CONFIGURATION_BITS(27 downto 22)
,

            ---------------- OUTPUT SIGNALS-----------------------
            --CLK to the RAM

            --Data and ADRS signal to RAM
            RAM_A_IN  => RAM_A_IN ,
            RAM_A_ADRS_OUT  =>RAM_A_ADRS_OUT  ,
            RAM_B_ADRS_OUT  => RAM_B_ADRS_OUT ,

            NEW_DATA_ARVD_2RAM  => NEW_DATA_ARVD_2RAM

            );

U9_RAM_FIFO: RAM_FIFO

      port map (
            RAM_A_IN =>RAM_A_IN  ,
            --RAM_B_IN : in std_logic_vector (DATA_PATH_WIDTH-1 downto
0);

            RAM_A_ADRS =>  RAM_A_ADRS_OUT ,
            RAM_B_ADRS  => RAM_B_ADRS_OUT  , -- Port B is only used for
reading.
            -- No input for port B since it is only read port.

            RAM_WR =>  RAM_WR ,-- 0 no op, 1 write. --Port A
            RAM_RD => RAM_RD  ,-- 0 no op, 1 READ.   --Port A
            RAM_ENABLE => DRPU_ENABLE  ,  -- synchronous, '1' enabled.
            RAM_CLEAR => DRPU_RESET  , -- synchronous Clear
            RAM_CONFIG_BITS =>   CONFIGURATION_BITS(55) ,
            --'0' for RAM behavior and
            -- '1' for FIFO behavior
            RAM_CLK => DRPU_CLK  ,

            FROM_RAM_FIFO_FULL =>  FROM_RAM_FIFO_FULL , --'1' for full
            FROM_RAM_FIFO_EMPTY => FROM_RAM_FIFO_EMPTY  ,--'1' for
empty
            FROM_RAM_A_OUT => FROM_RAM_A_OUT

            );


U10_RPU_CONTROLLER: RPU_CTRL_NEW
      port map (
            CONFIGURATION_BITS => CONFIGURATION_BITS ,
            CTRL_CLK =>DRPU_CLK ,
            FROM_RAM_FIFO_EMPTY => FROM_RAM_FIFO_EMPTY ,
            FROM_RAM_FIFO_FULL =>  FROM_RAM_FIFO_FULL,
            GO_CONFIG =>GO_CONFIG  ,
            GO_HOLD => GO_HOLD ,
            GO_RUN =>GO_RUN  ,
            GO_SLEEP => GO_SLEEP ,
            NEW_DATA_ARVD_2RAM =>NEW_DATA_ARVD_2RAM  ,
            RESET_CTROL => DRPU_RESET  ,
            RPU_START_HOLD_FROM_X => RPU_START_HOLD_FROM_X ,
            DISABLE_CLK => DISABLE_CLK ,
            DRPU_DONE => DRPU_DONE ,
            DRPU_FULL_CONFIG_BITS => DRPU_FULL_CONFIG_BITS ,
            DRPU_START_HOLD =>DRPU_START_HOLD  , -- to go to the outface
            ENABLE_DRPU => DRPU_ENABLE ,
            RAM_RD => RAM_RD ,
            RAM_WR =>  RAM_WR,
            RESET_DRPU => DRPU_RESET ,
            START_NEXT_DRPU =>  START_NEXT_DRPU

              );


U11_OUTPUT_INTERFACE: OUT_FACE

      port map(

            ---------------- INPUT SIGNALS-----------------------
            --Control signals
            TO_OTHER_RPU_START =>  START_NEXT_DRPU,
            OUT_FACE_CLK => DRPU_CLK ,

            RAP_1_OUT => RAP_OUT ,    -- the output of the RAP
            RAM_A_OUT  => FROM_RAM_A_OUT , -- ,,  ,, ,, RAM

            CLFSR_OUT_16 =>  CLFSR_OUT_16,
            SPRD_OUT=> SPRD_OUT  ,

            RPU_IN_FACE_1 => RPU_IN_FACE_1 ,
            RPU_IN_FACE_2 => RPU_IN_FACE_2 ,

            -- Configuration bits;
            OUT_FACE_CONFIG_BITS => CONFIGURATION_BITS( 16 downto 5)  ,

            ------------------------- OUTPUT SIGNALS ---------------
            RPU_OUT_1_GBUS  =>  RPU_OUT_1_GBUS,
            RPU_OUT_2_GBUS  =>RPU_OUT_2_GBUS  ,

            IN_N_RPU => IN_N_RPU ,
            IN_S_RPU => IN_S_RPU ,
            IN_W_RPU => IN_W_RPU ,
            IN_E_RPU => IN_E_RPU ,

            RPU_START_HOLD_E => RPU_START_HOLD_E , --'1' start, '0'
hold
            RPU_START_HOLD_W => RPU_START_HOLD_W , --'1' start, '0'
hold
            RPU_START_HOLD_N => RPU_START_HOLD_N , --'1' start, '0'
hold
            RPU_START_HOLD_S => RPU_START_HOLD_S --'1' start, '0' hold

            );

end Behav;
```

2 Communication and Switching Unit

```
---------------------------------------------------------------------
---------------------------------------------------------------------
--
--  Project            : DRAW
--  File name          : CSU.vhd
--  Title              : Configuration ans Switching control Unit
--  Description        : controls the configuration of the CMU
--  Design Libray      : DRAW
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
---------------------------------------------------------------------
--   Revisions   :
-- Date     Author  Revision         Comments
-- 4/15/02  A. Alsolaim   Rev 5
--
---------------------------------------------------------------------
---------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity CSU is
      generic(CONFIG_BITS_WIDTH: integer:=64;
            DATA_PATH_WIDTH: integer:=16; NUM_OF_SWB_PER_CSU: integer:=5);
-- NUM_OF_SWB_PER_CSU is either 4 or 5.
--it depends on the size of the array. the array is square array, where
--N in the number of elements in one
-- side.  If N is 4 then NUM_OF_SWB_PER_CSU must be 6 or 7.
--N takes the values of 8, 16, 32,...
      port(
            CSU_CLK: in std_logic;

            ---- INPUT from DRPUs
            DRPU_DONE: in std_logic_vector (3 downto 0); -- Done signales from RPU1(bit 0), RPU2 (bit 1),..., RPU4 (bit 3)



            ----INPOUT FROM GCM
            -- LODING NEW CONFIGurationdata for the RPUs or the SWBs to be saved in CMU or inside the CSU (near the SWB)
            CONFIG_BITS_TYPE: in std_logic; -- 0 means a RPU configuration type, 1 means SWB configuration type
            CONFIG_BITS: in std_logic_vector (CONFIG_BITS_WIDTH-1
downto 0); -- COnfiguration bits comes from the GCU as 8 bits word.
            --if the configuration is for the RPU it will take 8 cycles
to complete one configuration word of 64
            -- if it is for the SWB it will take 3 cycles. 24 bits
            STORE_CONFIG: in std_logic; -- from the GCM. when 1 save the
above data to the lower level of the CMU

            --COMMANDING RPUs
            DRPU_COMND: in std_logic_vector (15 downto 0); -- Command
from the GCU to the four RPUS. the structure of this signal is as follows
--bit 0     1      2     3          4      5     6     7
-- SLEEEP1 HOLD1 RUN1 CONFIG1  SLEEP2 HOLD2  RUN2  CONFIG2

            --- when configuration is high, then the number of the con-
figuration need to be selected. and so we need this signal
            CONFIG_NUM: in std_logic_vector(2 downto 0);
-- three bits. 2 bits were only needed if we use it for the RPUs only,
            --but since we use this input to point to one of the SWB we
need three bits. SWB may be 4 or 5.
            ----        OUTPUT SIGNALS-----------------
            TO_RPUS: out std_logic_vector(15 downto 0); -- this signal
is to control the four RPUS. the structure of this signal is as follows:
-- bit 0         1          2         3         4     5          6
7
-- GO_SLEEEP1 GO_HOLD1 GO_RUN1 GO_CONFIG1 GO_SLEEP2 GO_HOLD2 GO_RUN2
GO_CONFIG2...


            -- configuration bits of the SWB
            SWB_CONFIG_BITS: out std_logic_vector(NUM_OF_SWB_PER_CSU
*24 downto 0); -- 24 bits X 4 SWB . 0-23 for SWB1,...


            -- signal to control the writing-to and reading-from the CMU
            --Writing new value to the CMU
            ENABLE_CMU: out std_logic;
            WR_CONFIG: out std_logic;
            NEW_CONFIG: out std_logic_vector (CONFIG_BITS_WIDTH-1
downto 0);
            --Reading from CMU to load it to RPU
            CONFIG_NUM_TO_CMU: out std_logic_vector (1 downto 0);
            RPU_NUM:  out std_logic_vector (1 downto 0);
            RD_CONFIG: out std_logic ;
            -- out signals to GCU
            DRPU_DONE_TO_GCU: out std_logic_vector(3 downto 0);

            DRPU_STATUS: out std_logic_vector(7 downto 0)
            -- Signals to report the status of every RPU. structured as
follows:
            -- bit 0-1 for RPU1, bits 2-3 for RPU2, ... and so on.
            -- bits reports the following:
            --    00 RPU is sleep
            --    01 RPU is Hold
            --    10 RPU is Running
            --    11 RPU is Configuring.
            );
end entity;



Architecture BEHAV of CSU is

      signal START_CONFIG_RPU1:  std_logic;
      signal START_CONFIG_RPU2:  std_logic;
      signal START_CONFIG_RPU3:  std_logic;
      signal START_CONFIG_RPU4:  std_logic;


begin


      -- rout the RPU_DONEs to GCU  (registered)
      process( CSU_CLK)
      begin
      if rising_edge(CSU_CLK) then
            DRPU_DONE_TO_GCU <=DRPU_DONE;
            end if;
      end process;



      -- process to take the commands from the GCU and send the appro-
priate signal to the RPU
      process(CSU_CLK)
      begin
```

```
            if rising_edge(CSU_CLK) then
            -- for RPU1, RPU2, RPU3, and RPU4
            TO_RPUS<= DRPU_COMND;
            end if;
      end process;


      --process to report the status of the RPU
      process(CSU_CLK)
      begin
                  if rising_edge(CSU_CLK) then
            -- DRPU_COMND  --16 bits bits 3, 7, 11, and 15  are config-
uration commands, so they must be considered with CONFID_NUM

            --For RPU1
            case conv_integer(DRPU_COMND(3 downto 0)) is --  only one
command is active so value
                  -- of DRPU_COMND can only be 1, 2, 4, or 8
                  when 1 => DRPU_STATUS(1 downto 0)<="00";  -- RPU is
SLEEP
                  when 2 => DRPU_STATUS(1 downto 0)<="01";  -- RPU is
HOLD
                  when 4 => DRPU_STATUS(1 downto 0)<="10";  -- RPU is
RUN
                  when 8 => DRPU_STATUS(1 downto 0)<="11";
START_CONFIG_RPU1<='1';  -- RPU is CONFIG
                  when others => null;
            end case;

            --For RPU2
            case conv_integer(DRPU_COMND(7 downto 4)) is -- 1, 2, 4, or
8
                  when 1 => DRPU_STATUS(3 downto 2)<="00";  -- RPU is
SLEEP
                  when 2 => DRPU_STATUS(3 downto 2)<="01";  -- RPU is
HOLD
                  when 4 => DRPU_STATUS(3 downto 2)<="10";  -- RPU is
RUN
                  when 8 => DRPU_STATUS(3 downto 2)<="11";
START_CONFIG_RPU2<='1';  -- RPU is CONFIG
                  when others => null;
            end case;


            --For RPU3
            case conv_integer(DRPU_COMND(11 downto 8)) is -- 1, 2, 4,
or 8
                  when 1 => DRPU_STATUS(5 downto 4)<="00";  -- RPU is
SLEEP
                  when 2 => DRPU_STATUS(5 downto 4)<="01";  -- RPU is
HOLD
                  when 4 => DRPU_STATUS(5 downto 4)<="10";  -- RPU is
RUN
                  when 8 => DRPU_STATUS(5 downto 4)<="11";
START_CONFIG_RPU3<='1';  -- RPU is CONFIG
                  when others => null;
            end case;



            --For RPU4
            case conv_integer(DRPU_COMND(15 downto 12)) is -- 1, 2, 4,
or 8
                  when 1 => DRPU_STATUS(7 downto 6)<="00";  -- RPU is
SLEEP
                  when 2 => DRPU_STATUS(7 downto 6)<="01";  -- RPU is
HOLD
                  when 4 => DRPU_STATUS(7 downto 6)<="10";  -- RPU is
RUN
                  when 8 => DRPU_STATUS(7 downto 6)<="11";
START_CONFIG_RPU4<='1';  -- RPU is CONFIG
                  when others => null;
            end case;
                  end if;

      end process;
      --process to configure one of the RPU
      process (
START_CONFIG_RPU1,START_CONFIG_RPU2,START_CONFIG_RPU3,START_CONFIG_RP
U4, CSU_CLK)
      begin
                  if rising_edge(CSU_CLK) then
            if START_CONFIG_RPU1='1' then
                  CONFIG_NUM_TO_CMU <= CONFIG_NUM(1 downto 0);
                  RPU_NUM <= "00"; -- first RPU
                  RD_CONFIG <= '1';-- enable RD config

            elsif START_CONFIG_RPU2='1' then
                  CONFIG_NUM_TO_CMU <= CONFIG_NUM(1 downto 0);
                  RPU_NUM <= "01"; -- second RPU
                  RD_CONFIG <= '1';-- enable RD config
            elsif START_CONFIG_RPU3='1' then
                  CONFIG_NUM_TO_CMU <= CONFIG_NUM(1 downto 0);
                  RPU_NUM <= "10"; -- third RPU
                  RD_CONFIG <= '1';-- enable RD config

            elsif START_CONFIG_RPU4='1' then
                  CONFIG_NUM_TO_CMU <= CONFIG_NUM(1 downto 0);
                  RPU_NUM <= "11"; -- fourth RPU
                  RD_CONFIG <= '1';-- enable RD config

            else
                  START_CONFIG_RPU1 <='0';
                  START_CONFIG_RPU2 <='0';
                  START_CONFIG_RPU3 <='0';
                  START_CONFIG_RPU4 <='0';
            end if;
            if  ( START_CONFIG_RPU1 ='0'and START_CONFIG_RPU2 ='0' and
START_CONFIG_RPU3 ='0' and START_CONFIG_RPU4 ='0') then
                  RD_CONFIG<='0';

            end if;
            end if;
      end process;

      -- process to load a new configuration at the bottom of the RPU
or to the selected location in the SWB_CONFIG_BITS
      process (CSU_CLK)
      begin
                  if rising_edge(CSU_CLK) then
            if STORE_CONFIG='1' then
                  if CONFIG_BITS_TYPE='0' then  -- 0 means a RPU con-
figuration type, 1 means SWB configuration type
                        ENABLE_CMU<='1' ; -- enable the CMU to load the
bits into it.
                        WR_CONFIG<='1'; -- set the WR  signal to 1 for
writing to the CMU
                        NEW_CONFIG <= CONFIG_BITS;
                  end if;
                  if  CONFIG_BITS_TYPE='1' then  -- 0 means a RPU con-
figuration type, 1 means SWB configuration type

                        case conv_integer(CONFIG_NUM) is -- value can
be from 1  to 5
                              when 1 => SWB_CONFIG_BITS(23 downto 0) <=
CONFIG_BITS(23 downto 0);
```

```
                             when 2 => SWB_CONFIG_BITS(47 downto 24) <=
CONFIG_BITS(23 downto 0);
                             when 3 => SWB_CONFIG_BITS(71 downto 48) <=
CONFIG_BITS(23 downto 0);
                             when 4 => SWB_CONFIG_BITS(95 downto 72) <=
CONFIG_BITS(23 downto 0);
                             when 5 => SWB_CONFIG_BITS(119 downto 96)
<=    CONFIG_BITS(23 downto 0);
                             when others => null;

                       end case;

                 end if;
           end if;
           end if;
     end process;

end BEHAV;
```

3 Configuration Memory Unit

```
---------------------------------------------------------------------
---------------------------------------------------------------------
--
--  Project            : DREAM
--  File name          : COM_MEM_UNIT.vhd
--  Title              : Configuration Memory Unit
--  Description       : Configure the RPU with the help of the control
Unit
--                     :
--  Design Libray      :
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 4.2
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
---------------------------------------------------------------------
--   Revisions   :
--          Date             Author  Revision         Comments
--        12/25/01  A. Alsolaim   Rev 5
--
---------------------------------------------------------------------
```

```
----------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity COM_MEM_UNIT is
      generic(CONFIG_BITS_WIDTH: integer:=64;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ------------INPUT PORTS------------
            CLK: in std_logic;

            --------- CONTROL SIGNALS
            ENABLE_CMU: in std_logic;
            WR_CONFIG: in std_logic; --write one (selected by
CONFIG_NUM) of the stored configuration to the output
            CONFIG_NUM: in std_logic_vector (1 downto 0);---- CMU
stores 4 configurations.
            RPU_NUM: in    std_logic_vector (1 downto 0);---- CMU con-
figuration one of the RPUs.
            RD_CONFIG: in std_logic; --READ new configuration into the
unit.


            NEW_CONFIG: in std_logic_vector(CONFIG_BITS_WIDTH-1 downto
0);-- the new configuration bits

            ------------- OUTPUT PORTS----------
            CONFIG_RPU_1: out std_logic_vector(CONFIG_BITS_WIDTH-1
downto 0);
            CONFIG_RPU_2: out std_logic_vector(CONFIG_BITS_WIDTH-1
downto 0);
            CONFIG_RPU_3: out std_logic_vector(CONFIG_BITS_WIDTH-1
downto 0);
            CONFIG_RPU_4: out std_logic_vector(CONFIG_BITS_WIDTH-1
downto 0)
            );
end COM_MEM_UNIT;

Architecture BEHAV of COM_MEM_UNIT is
      type CMU_type is array (3 downto 0) of
      std_logic_vector(CONFIG_BITS_WIDTH-1 downto 0) ;
      signal CONFIG_REG: CMU_type;
```

```
begin

      process (clk)

      begin
            if    ENABLE_CMU='1' then
                  if WR_CONFIG='1' then --writing one of the configura-
tion (CONFIG_NUM)
                        --to one of the rpu (RPU_NUM)
                        case conv_integer(RPU_NUM) is
                              when 0 => CONFIG_RPU_1 <=
CONFIG_REG(conv_integer(CONFIG_NUM));
                              when 1 => CONFIG_RPU_2 <=
CONFIG_REG(conv_integer(CONFIG_NUM));
                              when 2 => CONFIG_RPU_3 <=
CONFIG_REG(conv_integer(CONFIG_NUM));
                              when 3 => CONFIG_RPU_4 <=
CONFIG_REG(conv_integer(CONFIG_NUM));
                              when others => null;
                        end case;
                  end if;
                  if RD_CONFIG='1' then --READ a new configuration into
the CMU
                        -- all configuration register are shifted up and
then
                        --the new configuration is stored into the
lowest reg.
                        -- the reg are numbered as 0 is the top and 3
is in the bottom
                        for i in 0 to 2 loop
                              CONFIG_REG(i)<=CONFIG_REG(i+1);
                        end loop;

                        CONFIG_REG(3)<=NEW_CONFIG;
                  end if;
            end if;
      end process;

end behav;
```

4 Dedicated I/O Unit

```
---------------------------------------------------------------------
---------------------------------------------------------------------
--
--  Project            : DREAM
--  File name          : DIO_Unit.vhd
--  Title              : Dedicated I/O Unit
--  Description       : the I/O unit. high speed and bandwidth I/O unit
--                     :
--  Design Libray      :
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 4.2
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
---------------------------------------------------------------------
--   Revisions   :
--          Date             Author  Revision         Comments
--        4/15/02 A. Alsolaim   Rev 5
--
---------------------------------------------------------------------
---------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.alL;

entity DIO_UNIT is
      generic (DATA_PATH_WIDTH: integer:=8);
      port (
            BUS_1: inout std_logic_vector (DATA_PATH_WIDTH-1 downto
0);
            BUS_2: inout std_logic_vector (DATA_PATH_WIDTH-1 downto
0);
            BUS_3: inout std_logic_vector (DATA_PATH_WIDTH-1 downto
0);
            BUS_4: inout std_logic_vector (DATA_PATH_WIDTH-1 downto
0);

            IO_CLK: in std_logic;
            X4_CLK: in std_logic;
            IO_CONFIG: in std_logic_vector(5 downto 0);
            -- Configuration bits are as follows:
            --bit 0 IN_OUT
            --bit 1 reg_not_reg
            --bit 2FIXD_SELECT_BUS
            --bit 3-4 the number of the fixed selected bus


            IO_PAD: inout std_logic_vector (DATA_PATH_WIDTH-1 downto 0)
            );
end DIO_UNIT;

architecture BEHAV of DIO_UNIT is

      signal ONE_BUS: std_logic_vector (DATA_PATH_WIDTH-1 downto 0);
      signal ONE_BUS_OUT: std_logic_vector (DATA_PATH_WIDTH-1 downto
0);
      signal ONE_BUS_IN:std_logic_vector (DATA_PATH_WIDTH-1 downto 0);
      signal REG_OUT: std_logic_vector (DATA_PATH_WIDTH-1 downto 0);
      signal REG_IN: std_logic_vector (DATA_PATH_WIDTH-1 downto 0);

      signal CLK: std_logic;
      signal CONTR_OUT: std_logic_vector(1 downto 0);
      signal BUS_SELECT: std_logic_vector(1 downto 0);


begin

      process (IO_CONFIG(2))
      begin
            if (IO_CONFIG(2))='0' then
                  CLK<=X4_CLK;
            else CLK<= IO_CLK;
            end if;
      end process;

      --2 bit counter
      process(X4_CLK)
            variable COUNT: std_logic_vector(1 downto 0) :="00";
      begin
            if rising_edge(X4_CLK) then
                  CONTR_OUT<=COUNT;
                  COUNT:=COUNT+1;
                  if COUNT="100" then
                        COUNT:="00";
                  end if ;
            end if;
      end process;

      process (IO_CONFIG(2))
      begin
            if IO_CONFIG(2)='0' then
                  BUS_SELECT<= CONTR_OUT;
            else BUS_SELECT<= IO_CONFIG(4 downto 3);
            end if;
      end process;

      process(IO_CONFIG(0))
      begin
            if IO_CONFIG(0)='1' then -- IO is an output pin
                  ONE_BUS_OUT<=ONE_BUS;
                  ONE_BUS_IN<=(others=>'Z');
            else
                  ONE_BUS_IN<=ONE_BUS;
                  ONE_BUS_OUT<=(others=>'Z');
            end if;
      end process;

      REG1:process(CLK)
      begin
            if rising_edge(CLK) then
                  REG_OUT<=ONE_BUS_OUT;
            end if;
      end process;

      REG2:process(CLK)
      begin
            if rising_edge(CLK) then
                  REG_IN<=IO_PAD;
            end if;
      end process;

      twomuxs: process(IO_CONFIG(1))
      begin
            if IO_CONFIG(1)='1' then --select unregistered in or out
                  IO_PAD<=ONE_BUS_OUT;
                  ONE_BUS_IN<=IO_PAD;
            else
                  IO_PAD<=REG_OUT;
```

```
                  ONE_BUS_IN<=REG_IN;
            end if;
      end process;




end BEHAV;
```

5 Arithmetic and Logical Unit

```
-----------------------------------------------------------------------
---------
-----------------------------------------------------------------------
----------
--
--  Project             : DRAW
--  File name           : ALU16.vhd
--  Title               : Arithmatic and Logical Unit
--  Description         : 16-bits Arithmatic and Logical Unit
--                      :
--  Design Libray       : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)        : AHDL 5.1
--                      :
--  Initialization      : none
--  Notes               :
--                      : Compile in VHDL'93
-----------------------------------------------------------------------
----------
--   Revisions    :
--          Date        Author  Revision         Comments
--        01/18/02  A. Alsolaim   Rev 18
--
-----------------------------------------------------------------------
---------
-----------------------------------------------------------------------
----------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;
```

```
entity ALU16 is
      generic (DATA_BIT_WIDTH : integer:=16);
      port(
            ALU_X_IN : in std_logic_vector (DATA_BIT_WIDTH-1 downto 0);
            ALU_Y_IN : in std_logic_vector (DATA_BIT_WIDTH-1 downto 0);
            AluCfg : in std_logic_vector (2 downto 0);
            CARY_IN : in std_logic;
            ALU_CLK : in std_logic;
            ALU_CLEAR : in std_logic;
            ALU_ENABLE : in std_logic;
            LOG_ARITH : in std_logic; -- configuration bit # 17. note
that we only
                  --pass the alu op which ic configuration bits 2-0 and
LOG_ARITH bit.
            CARY_OUT : out std_logic;
            OVR_FLW : out std_logic;
            ALU_OUT : out std_logic_vector (DATA_BIT_WIDTH-1 downto 0)
            );

end ALU16;

architecture rtl of ALU16 is

      ---------------------------------------------------------------
----
      -- CONFGI_BITSLOG_ARITH='0'LOG_ARITH='1'LOG_ARITH='1'
      --                     (LOGICAL)    CARY_IN='0 ' CARY_IN='1'
      ---------------------------------------------------------------
---
      --         0           X AND Y     X + Y                  X + Y
+ CARY_IN
      --         1           X NAND Y    X - Y                  X - Y
- CARY_IN
      --         2           X OR Y            X
X + CARY_IN
      --         3           X NOR Y     X
X - CARY_IN
      --         4           X XOR Y     X + 1                  X + 1
      --         5           X XNOR Y    X - 1                  X - 1
      --         6           NOT Y             MAX(x,y)
      --         7           NOT X             Min(x,y)

      component Two_Cmpl16
            generic (DATA_BIT_WIDTH:integer:=16);
            port(
                  x : in std_logic_vector(DATA_BIT_WIDTH-1 downto 0);
                  x_Cmpl : out std_logic_vector(DATA_BIT_WIDTH-1 downto
0)
                  );
      end component;

      component fa16
            generic (DATA_BIT_WIDTH :integer:=16);
            port (
                  cin:in std_logic;
                  a,b   :in std_logic_vector(DATA_BIT_WIDTH-1 downto
0);
                  s: out std_logic_vector(DATA_BIT_WIDTH-1 downto 0);
                  cout: out std_logic);
      end component;






      signal YC_Cmpl, YCarry, CarryIn2Vec, CarryInVec  :
std_logic_vector(DATA_BIT_WIDTH-1 downto 0);
      signal tmp_out, tmp_out1, tmp_x,tmp_y : std_logic_vector
(DATA_BIT_WIDTH-1 downto 0);
      signal tmp, tmp_ovflow, tmp_ov,zero: std_logic;
      signal CnstZero,CnstOne, CnstMinusOne :std_logic_vector
(DATA_BIT_WIDTH-1 downto 0);

begin
      zero<='0';
      CnstOne(DATA_BIT_WIDTH-1 downto 1)<=(oth-
ers=>'0');CnstOne(0)<='1';
      CnstMinusOne<=(others=>'1');
      CnstZero<=(others=>'0');
      CarryInVec(DATA_BIT_WIDTH-1 downto 1) <=(others=>'0');--
"0000000" & CARY_IN;
      CarryInVec(0)<=Cary_IN;
      CarryIN2Vec<=(others=>CARY_IN);-- & CARY_IN & CARY_IN & CARY_IN
& CARY_IN & CARY_IN & CARY_IN & CARY_IN;
      U_Two_Cmpl16:Two_Cmpl16 port map(x=>YCarry, x_Cmpl=>YC_Cmpl);
      U1_fa16:fa16 port map (
            cin=>zero,
            a=>CarryInVec,
            b=>ALU_Y_IN,
            s=>YCarry,
            cout=>tmp);

      process(ALU_ENABLE, AluCfg,ALU_X_IN,ALU_Y_IN,YCarry,YC_Cmpl,CarryInVec,CarryIn2Vec)
      begin
            if(ALU_ENABLE='1')then
                  case AluCfg is
                        when "000"=>
                        tmp_x<=ALU_X_IN;
                        tmp_y<=YCarry;
                        when "001"=>
                        tmp_x<=ALU_X_IN;
                        tmp_y<=YC_Cmpl;
                        when "010"=>
                        tmp_x<=ALU_X_IN;
                        tmp_y<=CarryInVec;
                        when "011"=>
                        tmp_x<=ALU_X_IN;
                        tmp_y<=CarryIn2Vec;
                        when "100"=>
                        tmp_x<=ALU_X_IN;
                        tmp_y<=CnstOne;
                        when "101"=>
                        tmp_x<=ALU_X_IN;
                        tmp_y<=CnstMinusOne;
                        when "110"=>
                        tmp_x<=ALU_X_IN;
                        tmp_y<=ALU_Y_IN;
                        when "111"=>
                        tmp_x<=ALU_X_IN;
                        tmp_y<=ALU_Y_IN;
                        when others=>null;
                  end case;
            end if;
      end process;
      U2_fa16:fa16 port map(
            cin=>zero,
            a=>tmp_x,
            b=>tmp_y,
            s=>tmp_out,
            cout=>tmp_ov);
      tmp_ovflow<=(not ( tmp_x(DATA_BIT_WIDTH-1) xor
tmp_y(DATA_BIT_WIDTH-1)))and (tmp_out(DATA_BIT_WIDTH-1) xor
tmp_y(DATA_BIT_WIDTH-1));
      process(ALU_ENABLE, AluCfg,ALU_X_IN,ALU_Y_IN)
      begin
            if(ALU_ENABLE='1')then
                  case AluCfg is
                        when "000"=>
                        tmp_out1<=ALU_X_IN and ALU_Y_IN;

                        when "001"=>
                        tmp_out1<=not(ALU_X_IN and ALU_Y_IN);
                        when "010"=>
                        tmp_out1<=ALU_X_IN or ALU_Y_IN;
                        when "011"=>
                        tmp_out1<=not(ALU_X_IN or ALU_Y_IN);
                        when "100"=>
                        tmp_out1<=ALU_X_IN xor ALU_Y_IN;
                        when "101"=>
                        tmp_out1<=not(ALU_X_IN xor ALU_Y_IN);
                        when "110"=>
                        tmp_out1<=not ALU_X_IN;
                        when "111"=>
                        tmp_out1<=not ALU_Y_IN;
                        when others=>null;
                  end case;
            end if;
      end process;
      process(ALU_ENABLE, ALU_CLK,
ALU_CLEAR,tmp_out,tmp_out1,LOG_ARITH)
      begin
            if(ALU_ENABLE='1')then
                  if(ALU_CLEAR='1')then
                        ALU_OUT <=(others=>'0');
                        OVR_FLW <='0';
                  elsif(ALU_CLK'event and ALU_CLK='1')then

                        if(LOG_ARITH='1')then
                              ALU_OUT<=tmp_out; --***********
                              OVR_FLW<=tmp_ovflow;
                        else
                              ALU_OUT<=tmp_out1;
                              OVR_FLW<='0';
                        end if;
                  end if;
            end if;
      end process;

end rtl;
```

6 Booth Decoder Unit

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project            : DRAW
--  File name          : boothdec.vhd
--  Title              : Booth decoder Unit
--  Description        : Booth decoder unit for booth multiplier.
--                     :
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date        Author  Revision         Comments
--        01/18/02  M. Ding   Rev 2 By A. Alsolaim
--
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity BoothDec is
port (
Din : in std_logic_vector(2 downto 0);
Dout :out std_logic_vector(1 downto 0);
P_N : out std_logic
);
end BoothDec;

architecture BoothDec of BoothDec is
begin
P_N<= Din(2);
process(Din)
      begin
      case Din is
            when "000"|"111" =>
                  dout<="00";

            when "001"|"010"|"101"|"110"=>
                  dout<="01";

            when "011"|"100"=>
                  dout<="10";

            when  others=>
                  null;
      end case;
end process;
end BoothDec;
```

7 Barrel shifter

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project            : DRAW
--  File name          : BRL_SFT_16.vhd
--  Title              : Barrel Shift
--  Description        : 16-bit arithmetic shift unit
--                     :
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
---------------------------------------------------------------------
----------
--   Revisions   :
--          Date             Author   Revision         Comments
--        02/2/02  A. Alsolaim   Rev 6
--
---------------------------------------------------------------------
---------
---------------------------------------------------------------------
----------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity BRL_SFT_16 is
      generic(DATA_WIDTH : integer:=16);
      port (
            DIR : in STD_LOGIC;  ----1 right 0 left
            X_IN : in STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
            Y_OUT : out STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
            NUM_SHFTS : in STD_LOGIC_VECTOR (3 downto 0) ;
            RTT_SHF: in STD_LOGIC:='1'; --0 Rotate 1 Shift
            ARTH_LOGC: in std_logic:='0' -- 1 Arithmatic 0 logical
            );
end entity ;

architecture BRL_SFT_16 of BRL_SFT_16 is

      function MUX2 (A, B, C: std_logic) return std_logic is
      begin
            if C='1' then
                  return A;
            else
                  return B;
            end if;
      end function;
      signal LEVEL_A : STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
      signal LEVEL_B : STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
      signal LEVEL_C : STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);

      signal TT: std_logic:='0';
      -- TT will take the LSB or MSB ofX_IN depending on Logical or
Arithmatic
begin
      TT<=X_IN(15) when (ARTH_LOGC='1'and DIR='1') else X_IN(0) when
(ARTH_LOGC='1' and DIR='0')
      else '0' when (ARTH_LOGC='0') ;

      LEVEL_A(0) <= MUX2(X_IN(15), X_IN(0), NUM_SHFTS(0)) when (DIR =
'0'and RTT_SHF='0') else MUX2(X_IN(1), X_IN(0), NUM_SHFTS(0))when (DIR
= '1'and RTT_SHF='0')
      else MUX2(TT, X_IN(0), NUM_SHFTS(0)) when (DIR = '0'and
RTT_SHF='1') else MUX2(X_IN(1), X_IN(0), NUM_SHFTS(0)) when (DIR =
'1'and RTT_SHF='1');

      LEVEL_A(1) <= MUX2(X_IN(0), X_IN(1), NUM_SHFTS(0)) when (DIR =
'0') else MUX2(X_IN(2), X_IN(1), NUM_SHFTS(0));
      LEVEL_A(2) <= MUX2(X_IN(1), X_IN(2), NUM_SHFTS(0)) when (DIR =
'0') else MUX2(X_IN(3), X_IN(2), NUM_SHFTS(0));
      LEVEL_A(3) <= MUX2(X_IN(2), X_IN(3), NUM_SHFTS(0)) when (DIR =
'0') else MUX2(X_IN(4), X_IN(3), NUM_SHFTS(0));
      LEVEL_A(4) <= MUX2(X_IN(3), X_IN(4), NUM_SHFTS(0)) when (DIR =
'0') else MUX2(X_IN(5), X_IN(4), NUM_SHFTS(0));
      LEVEL_A(5) <= MUX2(X_IN(4), X_IN(5), NUM_SHFTS(0)) when (DIR =
'0') else MUX2(X_IN(6), X_IN(5), NUM_SHFTS(0));
      LEVEL_A(6) <= MUX2(X_IN(5), X_IN(6), NUM_SHFTS(0)) when (DIR =
'0') else MUX2(X_IN(7), X_IN(6), NUM_SHFTS(0));
      LEVEL_A(7) <= MUX2(X_IN(6), X_IN(7), NUM_SHFTS(0)) when (DIR =
'0') else MUX2(X_IN(8), X_IN(7), NUM_SHFTS(0));
      LEVEL_A(8) <= MUX2(X_IN(7), X_IN(8), NUM_SHFTS(0)) when (DIR =
'0') else MUX2(X_IN(9), X_IN(8), NUM_SHFTS(0));
      LEVEL_A(9) <= MUX2(X_IN(8), X_IN(9), NUM_SHFTS(0)) when (DIR =
'0') else MUX2(X_IN(10), X_IN(9), NUM_SHFTS(0));
      LEVEL_A(10) <= MUX2(X_IN(9), X_IN(10), NUM_SHFTS(0)) when (DIR =
'0') else MUX2(X_IN(11), X_IN(10), NUM_SHFTS(0));
      LEVEL_A(11) <= MUX2(X_IN(10), X_IN(11), NUM_SHFTS(0)) when (DIR
= '0') else MUX2(X_IN(12), X_IN(11), NUM_SHFTS(0));
      LEVEL_A(12) <= MUX2(X_IN(11), X_IN(12), NUM_SHFTS(0)) when (DIR
= '0') else MUX2(X_IN(13), X_IN(12), NUM_SHFTS(0));
      LEVEL_A(13) <= MUX2(X_IN(12), X_IN(13), NUM_SHFTS(0)) when (DIR
= '0') else MUX2(X_IN(14), X_IN(13), NUM_SHFTS(0));
      LEVEL_A(14) <= MUX2(X_IN(13), X_IN(14), NUM_SHFTS(0)) when (DIR
= '0') else MUX2(X_IN(15), X_IN(14), NUM_SHFTS(0));
```

```
      LEVEL_A(15) <= MUX2(X_IN(14), X_IN(15), NUM_SHFTS(0)) when (DIR
= '0'and RTT_SHF='0') else MUX2(X_IN(0), X_IN(15), NUM_SHFTS(0))when
(DIR = '1'and RTT_SHF='0')
      else MUX2(X_IN(14), X_IN(15), NUM_SHFTS(0)) when (DIR = '0'and
RTT_SHF='1') else MUX2(TT, X_IN(15), NUM_SHFTS(0))when (DIR = '1'and
RTT_SHF='1');

      --*********************************




      LEVEL_B(0) <= MUX2(LEVEL_A(14), LEVEL_A(0), NUM_SHFTS(1)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_A(2), LEVEL_A(0), NUM_SHFTS(1))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_A(0), NUM_SHFTS(1)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_A(2), LEVEL_A(0), NUM_SHFTS(1))when (DIR = '1'and
RTT_SHF='1');


      LEVEL_B(1) <= MUX2(LEVEL_A(15), LEVEL_A(1), NUM_SHFTS(1)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_A(3), LEVEL_A(1), NUM_SHFTS(1))when (DIR = '1'and
RTT_SHF='0')
      else MUX2( TT, LEVEL_A(1), NUM_SHFTS(1)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_A(3), LEVEL_A(1), NUM_SHFTS(1))when (DIR = '1'and
RTT_SHF='1');


      LEVEL_B(2) <= MUX2(LEVEL_A(0), LEVEL_A(2), NUM_SHFTS(1)) when
(DIR = '0')
      else MUX2(LEVEL_A(4), LEVEL_A(2), NUM_SHFTS(1));
      LEVEL_B(3) <= MUX2(LEVEL_A(1), LEVEL_A(3), NUM_SHFTS(1)) when
(DIR = '0')
      else MUX2(LEVEL_A(5), LEVEL_A(3), NUM_SHFTS(1));
      LEVEL_B(4) <= MUX2(LEVEL_A(2), LEVEL_A(4), NUM_SHFTS(1)) when
(DIR = '0')
      else MUX2(LEVEL_A(6), LEVEL_A(4), NUM_SHFTS(1));
      LEVEL_B(5) <= MUX2(LEVEL_A(3), LEVEL_A(5), NUM_SHFTS(1)) when
(DIR = '0')
```

```
      else MUX2(LEVEL_A(7), LEVEL_A(5), NUM_SHFTS(1));
      LEVEL_B(6) <= MUX2(LEVEL_A(4), LEVEL_A(6), NUM_SHFTS(1)) when
(DIR = '0')
      else MUX2(LEVEL_A(8), LEVEL_A(6), NUM_SHFTS(1));
      LEVEL_B(7) <= MUX2(LEVEL_A(5), LEVEL_A(7), NUM_SHFTS(1)) when
(DIR = '0')
      else MUX2(LEVEL_A(9), LEVEL_A(7), NUM_SHFTS(1));
      LEVEL_B(8) <= MUX2(LEVEL_A(6), LEVEL_A(8), NUM_SHFTS(1)) when
(DIR = '0')
      else MUX2(LEVEL_A(10), LEVEL_A(8), NUM_SHFTS(1));
      LEVEL_B(9) <= MUX2(LEVEL_A(7), LEVEL_A(9), NUM_SHFTS(1)) when
(DIR = '0')
      else MUX2(LEVEL_A(11), LEVEL_A(9), NUM_SHFTS(1));
      LEVEL_B(10) <= MUX2(LEVEL_A(8), LEVEL_A(10), NUM_SHFTS(1)) when
(DIR = '0')
      else MUX2(LEVEL_A(12), LEVEL_A(10), NUM_SHFTS(1));
      LEVEL_B(11) <= MUX2(LEVEL_A(9), LEVEL_A(11), NUM_SHFTS(1)) when
(DIR = '0')
      else MUX2(LEVEL_A(13), LEVEL_A(11), NUM_SHFTS(1));
      LEVEL_B(12) <= MUX2(LEVEL_A(10), LEVEL_A(12), NUM_SHFTS(1)) when
(DIR = '0')
      else MUX2(LEVEL_A(14), LEVEL_A(12), NUM_SHFTS(1));
      LEVEL_B(13) <= MUX2(LEVEL_A(11), LEVEL_A(13), NUM_SHFTS(1)) when
(DIR = '0')
      else MUX2(LEVEL_A(15), LEVEL_A(13), NUM_SHFTS(1));

      LEVEL_B(14) <= MUX2(LEVEL_A(12), LEVEL_A(14), NUM_SHFTS(1)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_A(0), LEVEL_A(14), NUM_SHFTS(1))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(LEVEL_A(12), LEVEL_A(14), NUM_SHFTS(1)) when (DIR =
'0'and RTT_SHF='1')
      else MUX2(TT, LEVEL_A(14), NUM_SHFTS(1))when (DIR = '1'and
RTT_SHF='1');


      LEVEL_B(15) <= MUX2(LEVEL_A(13), LEVEL_A(15), NUM_SHFTS(1))when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_A(1), LEVEL_A(15), NUM_SHFTS(1))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(LEVEL_A(13), LEVEL_A(15), NUM_SHFTS(1))when (DIR =
'0'and RTT_SHF='1')
      else MUX2(TT,LEVEL_A(15), NUM_SHFTS(1))when (DIR = '1'and
RTT_SHF='1') ;

      --**************************************
      LEVEL_C(0) <= MUX2(LEVEL_B(12), LEVEL_B(0), NUM_SHFTS(2)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_B(4), LEVEL_B(0), NUM_SHFTS(2))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_B(0), NUM_SHFTS(2)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_B(4), LEVEL_B(0), NUM_SHFTS(2))when (DIR = '1'and
RTT_SHF='1');


      LEVEL_C(1) <= MUX2(LEVEL_B(13), LEVEL_B(1), NUM_SHFTS(2))  when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_B(5), LEVEL_B(1), NUM_SHFTS(2)) when (DIR =
'1'and RTT_SHF='0')
      else MUX2(TT, LEVEL_B(1), NUM_SHFTS(2))  when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_B(5), LEVEL_B(1), NUM_SHFTS(2)) when (DIR =
'1'and RTT_SHF='1');


      LEVEL_C(2) <= MUX2(LEVEL_B(14), LEVEL_B(2), NUM_SHFTS(2)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_B(6), LEVEL_B(2), NUM_SHFTS(2))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_B(2), NUM_SHFTS(2)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_B(6), LEVEL_B(2), NUM_SHFTS(2))when (DIR = '1'and
RTT_SHF='1');


      LEVEL_C(3) <= MUX2(LEVEL_B(15), LEVEL_B(3), NUM_SHFTS(2)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_B(7), LEVEL_B(3), NUM_SHFTS(2)) when (DIR =
'1'and RTT_SHF='0')
      else MUX2(TT, LEVEL_B(3), NUM_SHFTS(2)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_B(7), LEVEL_B(3), NUM_SHFTS(2)) when (DIR =
'1'and RTT_SHF='1') ;



      LEVEL_C(4) <= MUX2(LEVEL_B(0), LEVEL_B(4), NUM_SHFTS(2)) when
(DIR = '0')
      else MUX2(LEVEL_B(8), LEVEL_B(4), NUM_SHFTS(2));
      LEVEL_C(5) <= MUX2(LEVEL_B(1), LEVEL_B(5), NUM_SHFTS(2)) when
(DIR = '0')
      else MUX2(LEVEL_B(9), LEVEL_B(5), NUM_SHFTS(2));
      LEVEL_C(6) <= MUX2(LEVEL_B(2), LEVEL_B(6), NUM_SHFTS(2)) when
(DIR = '0')
      else MUX2(LEVEL_B(10), LEVEL_B(6), NUM_SHFTS(2));
      LEVEL_C(7) <= MUX2(LEVEL_B(3), LEVEL_B(7), NUM_SHFTS(2)) when
(DIR = '0')
      else MUX2(LEVEL_B(11), LEVEL_B(7), NUM_SHFTS(2));
      LEVEL_C(8) <= MUX2(LEVEL_B(4), LEVEL_B(8), NUM_SHFTS(2)) when
(DIR = '0')
      else MUX2(LEVEL_B(12), LEVEL_B(8), NUM_SHFTS(2));
      LEVEL_C(9) <= MUX2(LEVEL_B(5), LEVEL_B(9), NUM_SHFTS(2)) when
(DIR = '0')
      else MUX2(LEVEL_B(13), LEVEL_B(9), NUM_SHFTS(2));
      LEVEL_C(10) <= MUX2(LEVEL_B(6), LEVEL_B(10), NUM_SHFTS(2)) when
(DIR = '0')
      else MUX2(LEVEL_B(14), LEVEL_B(10), NUM_SHFTS(2));
      LEVEL_C(11) <= MUX2(LEVEL_B(7), LEVEL_B(11), NUM_SHFTS(2)) when
(DIR = '0')
      else MUX2(LEVEL_B(15), LEVEL_B(11), NUM_SHFTS(2));




      LEVEL_C(12) <= MUX2(LEVEL_B(8), LEVEL_B(12), NUM_SHFTS(2)) when
(DIR = '0' and RTT_SHF='0')
      else MUX2(LEVEL_B(0), LEVEL_B(12), NUM_SHFTS(2)) when (DIR = '1'
and RTT_SHF='0')
      else MUX2(LEVEL_B(8), LEVEL_B(12), NUM_SHFTS(2)) when (DIR = '0'
and RTT_SHF='1')
      else MUX2(TT, LEVEL_B(12), NUM_SHFTS(2)) when (DIR = '1' and
RTT_SHF='1');


      LEVEL_C(13) <= MUX2(LEVEL_B(9), LEVEL_B(13), NUM_SHFTS(2)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_B(1), LEVEL_B(13), NUM_SHFTS(2))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(LEVEL_B(9), LEVEL_B(13), NUM_SHFTS(2)) when (DIR =
'0'and RTT_SHF='1')
      else MUX2(TT, LEVEL_B(13), NUM_SHFTS(2))when (DIR = '1'and
RTT_SHF='1');
```

```
      LEVEL_C(14) <= MUX2(LEVEL_B(10), LEVEL_B(14), NUM_SHFTS(2)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_B(2), LEVEL_B(14), NUM_SHFTS(2))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(LEVEL_B(10), LEVEL_B(14), NUM_SHFTS(2)) when (DIR =
'0'and RTT_SHF='1')
      else MUX2(TT, LEVEL_B(14), NUM_SHFTS(2))when (DIR = '1'and
RTT_SHF='1');



      LEVEL_C(15) <= MUX2(LEVEL_B(11), LEVEL_B(15), NUM_SHFTS(2)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_B(3), LEVEL_B(15), NUM_SHFTS(2))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(LEVEL_B(11), LEVEL_B(15), NUM_SHFTS(2)) when (DIR =
'0'and RTT_SHF='1')
      else MUX2(TT, LEVEL_B(15), NUM_SHFTS(2))when (DIR = '1'and
RTT_SHF='1');



      --********************************************************
      Y_OUT(0)<= MUX2(LEVEL_C(8), LEVEL_C(0), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(8), LEVEL_C(0), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_C(0), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(8), LEVEL_C(0), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='1');

      Y_OUT(1) <= MUX2(LEVEL_C(9), LEVEL_C(1), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(7), LEVEL_C(1), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_C(1), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(9), LEVEL_C(1), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='1');

      Y_OUT(2) <= MUX2(LEVEL_C(10), LEVEL_C(2), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(6), LEVEL_C(2), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_C(2), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
```

```
      else MUX2(LEVEL_C(10), LEVEL_C(2), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='1');

      Y_OUT(3) <= MUX2(LEVEL_C(11), LEVEL_C(3), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(5), LEVEL_C(3), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_C(3), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(11), LEVEL_C(3), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='1');

      Y_OUT(4) <= MUX2(LEVEL_C(12), LEVEL_C(4), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(4), LEVEL_C(4), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_C(4), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(12), LEVEL_C(4), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='1');

      Y_OUT(5) <= MUX2(LEVEL_C(13), LEVEL_C(5), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(3), LEVEL_C(5), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_C(5), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(13), LEVEL_C(5), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='1');

      Y_OUT(6) <= MUX2(LEVEL_C(14), LEVEL_C(6), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(2), LEVEL_C(6), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_C(6), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(14), LEVEL_C(6), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='1');

      Y_OUT(7) <= MUX2(LEVEL_C(15), LEVEL_C(7), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(1), LEVEL_C(7), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_C(7), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(15), LEVEL_C(7), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='1');
```

```
      --**************

      Y_OUT(8) <= MUX2(LEVEL_C(0), LEVEL_C(8), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(0), LEVEL_C(8), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='0')
      else MUX2(TT, LEVEL_C(8), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(0), LEVEL_C(8), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='1');

      Y_OUT(9) <= MUX2(LEVEL_C(1), LEVEL_C(9), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(15), LEVEL_C(9), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(TT, LEVEL_C(9), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(1), LEVEL_C(9), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='1');

      Y_OUT(10) <= MUX2(LEVEL_C(2), LEVEL_C(10), NUM_SHFTS(3)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(14), LEVEL_C(10), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(TT, LEVEL_C(10), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(2), LEVEL_C(10), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='1');

      Y_OUT(11) <= MUX2(LEVEL_C(3), LEVEL_C(11), NUM_SHFTS(3)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(13), LEVEL_C(11), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(TT, LEVEL_C(11), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(3), LEVEL_C(11), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='1');

      Y_OUT(12) <= MUX2(LEVEL_C(4), LEVEL_C(12), NUM_SHFTS(3)) when
(DIR = '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(12), LEVEL_C(12), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(TT, LEVEL_C(12), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(4), LEVEL_C(12), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='1');
```

```
      Y_OUT(13) <= MUX2(LEVEL_C(5), LEVEL_C(3), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(11), LEVEL_C(13), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(TT, LEVEL_C(13), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(5), LEVEL_C(3), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='1');


      Y_OUT(14) <=MUX2(LEVEL_C(6), LEVEL_C(3), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(10), LEVEL_C(14), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(TT, LEVEL_C(14), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(6), LEVEL_C(3), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='1');

      Y_OUT(15) <= MUX2(LEVEL_C(7), LEVEL_C(3), NUM_SHFTS(3)) when (DIR
= '0'and RTT_SHF='0')
      else MUX2(LEVEL_C(9), LEVEL_C(15), NUM_SHFTS(3))when (DIR =
'1'and RTT_SHF='0')
      else MUX2(TT, LEVEL_C(15), NUM_SHFTS(3)) when (DIR = '0'and
RTT_SHF='1')
      else MUX2(LEVEL_C(7), LEVEL_C(3), NUM_SHFTS(3))when (DIR = '1'and
RTT_SHF='1');

--------********************************************************
------Y_OUT(0) <= MUX2(LEVEL_C(8), LEVEL_C(0), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(TT, LEVEL_C(0), NUM_SHFTS(3))when (RTT_SHF='1');
------
------Y_OUT(1) <= MUX2(LEVEL_C(9), LEVEL_C(1), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(TT, LEVEL_C(1), NUM_SHFTS(3))when (RTT_SHF='1');
------
------Y_OUT(2) <= MUX2(LEVEL_C(10), LEVEL_C(2), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(TT, LEVEL_C(2), NUM_SHFTS(3))when (RTT_SHF='1');
------
------Y_OUT(3) <= MUX2(LEVEL_C(11), LEVEL_C(3), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(TT, LEVEL_C(3), NUM_SHFTS(3))when (RTT_SHF='1');
------
```

```
------Y_OUT(4) <= MUX2(LEVEL_C(12), LEVEL_C(4), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(TT, LEVEL_C(4), NUM_SHFTS(3))when (RTT_SHF='1');
------
------Y_OUT(5) <= MUX2(LEVEL_C(13), LEVEL_C(5), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(TT, LEVEL_C(5), NUM_SHFTS(3))when (RTT_SHF='1');
------
------Y_OUT(6) <= MUX2(LEVEL_C(14), LEVEL_C(6), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(TT, LEVEL_C(6), NUM_SHFTS(3))when (RTT_SHF='1');
------
------Y_OUT(7) <= MUX2(LEVEL_C(15), LEVEL_C(7), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(TT, LEVEL_C(7), NUM_SHFTS(3))when (RTT_SHF='1');
------
--------**************
------
------Y_OUT(8) <= MUX2(LEVEL_C(0), LEVEL_C(8), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(LEVEL_C(0), TT, NUM_SHFTS(3))when (RTT_SHF='1');
------
------Y_OUT(9) <= MUX2(LEVEL_C(1), LEVEL_C(9), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(LEVEL_C(1),TT, NUM_SHFTS(3))when (RTT_SHF='1');
------
------Y_OUT(10) <= MUX2(LEVEL_C(2), LEVEL_C(10), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(LEVEL_C(2), TT, NUM_SHFTS(3))when (RTT_SHF='1') ;
------
------Y_OUT(11) <= MUX2(LEVEL_C(3), LEVEL_C(11), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(LEVEL_C(3), TT, NUM_SHFTS(3))when (RTT_SHF='1');
------
------Y_OUT(12) <= MUX2(LEVEL_C(4), LEVEL_C(12), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(LEVEL_C(4), TT, NUM_SHFTS(3))when (RTT_SHF='1');
------
------Y_OUT(13) <= MUX2(LEVEL_C(5), LEVEL_C(13), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(LEVEL_C(5), TT, NUM_SHFTS(3))when (RTT_SHF='1');
------
------
------Y_OUT(14) <= MUX2(LEVEL_C(6), LEVEL_C(14), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(LEVEL_C(6), TT, NUM_SHFTS(3))when (RTT_SHF='1');
```

```
------
------Y_OUT(15) <= MUX2(LEVEL_C(7), LEVEL_C(15), NUM_SHFTS(3))when
(RTT_SHF='0')
------else MUX2(LEVEL_C(7), TT, NUM_SHFTS(3))when (RTT_SHF='1');

end architecture BRL_SFT_16;
```

8 RAP configuration control unit

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project             : DRAW
--  File name           : CfgCtl.vhd
--  Title               : RAP configuration control unit
--  Description      : To control the implementation of all arithmatic
and logical operation. Additionally,
--                              : it controls the implemntation of the
booth multiplier on the ALU.
--                      :
--  Design Libray       : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)        : Ahdl 5.1
--                      :
--  Initialization      : none
--  Notes               :
--                      : Compile in VHDL'93
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date        Author   Revision         Comments
--        01/10/02  M. Ding   Rev 7 By A. ALsolaim
--
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
-------------------------------------------------------------------
-- RAP CONFGI_BITSEIGHT_ONE='0'EIGHT_ONE='1'EIGHT_ONE='1'
--                      (LOGICAL)    CARY_IN='0 ' CARY_IN='1'
------------------------------------------------------------------
--          0           X AND Y     X + Y + CARY_IN
--          1           X NAND Y    X - Y - CARY_IN
--          2           X OR Y            X*y
--          3           X NOR Y     x/y  --not used now
--          4           X XOR Y     X + 1                  X + 1
--          5           X XNOR Y    X - 1                  X - 1
--          6           NOT X             MAX(x,y)
--          7           NOT Y             Min(x,y)
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity CfgCtl is
      generic (RAP_CONFIG_BITS_WIDTH: integer:=22;
            CFGCTRL_DATA_PATH_WIDTH: integer:=8);
      port (
            RapCfg: in std_logic_vector( RAP_CONFIG_BITS_WIDTH -1
downto 0);
            BoothX : in std_logic_vector( CFGCTRL_DATA_PATH_WIDTH-1
downto 0);
            Py : in std_logic;
            Pa : in std_logic;
            clk : in std_logic;
            --start: in std_logic;
            en : in std_logic;
            DIR_X : out STD_LOGIC;
            DIR_Y : out STD_LOGIC;
            NUM_SHFTS_X : out STD_LOGIC_VECTOR (2 downto 0);
            NUM_SHFTS_Y : out STD_LOGIC_VECTOR (2 downto 0);
            YSel : out STD_LOGIC_VECTOR (1 downto 0);
            XSel : out STD_LOGIC_VECTOR (1 downto 0);
            RTT_SHF_X: out STD_LOGIC:='1'; --0 Rotate 1 Shift
            RTT_SHF_Y: out STD_LOGIC:='1'; --0 Rotate 1 Shift
            ARTH_LOGC_X: out std_logic:='0'; -- 1 Arithmatic 0 logical
            ARTH_LOGC_Y: out std_logic:='0'; -- 1 Arithmatic 0 logical
            ASel : out STD_LOGIC_VECTOR (1 downto 0);
            ALU_Op : out  STD_LOGIC_VECTOR (2 downto 0);
            StateCnt: out std_logic_vector(1 downto 0)
            );
end CfgCtl;
--RAPCfg Words
```

```
--0 1 2  |3 4 |5 6 |7 8 |9    |10 11 12|13   |14 15 16|17 |
--ALU_OP |XSel|YSel|ASel|X_dir|X_Shft  |y_dir|Y_Shft  |ext|
architecture CfgCtl of CfgCtl is
      component BoothDec
            port (
                  Din : in std_logic_vector(2 downto 0);
                  Dout :out std_logic_vector(1 downto 0);
                  P_N : out std_logic
                  );
      end component;

      signal cnt :integer range 0 to 3;
      signal BoothCodeX, BoothCodeY,Op :std_logic_vector(2 downto 0);
      signal BoothDecodeX,BoothDecodeY :std_logic_vector(1 downto 0);
      signal Pxy:std_logic_vector(1 downto 0);
      signal Px, P_NX, P_NY :std_logic;
      signal LOG_ARITH :std_logic;

begin

      U1_BoothDec: BoothDec
      port map(
            din=>BoothCodeX,
            dout=>BoothDecodeX,
            P_N=>P_NX);

      U2_BoothDec: BoothDec
      port map(
            din=>BoothCodeY,
            dout=>BoothDecodeY,
            P_N=>P_NY);



      counter:process(clk,en)   -- Changed the Start signal to en --
removed the Start signal from the ports.
      begin
            if(en='0')then
                  cnt<=3;
            elsif(clk'event and clk='1')then
                  case cnt is
                        when 3=>
                        cnt<=0;
                        when 2=>
                        cnt<=3;
                        when 1=>
                        cnt<=2;
                        when 0=>
                        cnt<=1;
                        when others=>
                        cnt<=0;
                  end case;
            end if;
      end process;
      StateCnt<=conv_std_logic_vector(cnt,2);
      ShftGen:process(cnt,BoothX,en)
      begin
            if(en='1')then
                  case cnt is
                        when 0 =>

                        BoothCodeY<=BoothX(1 downto 0) & '0';
                        BoothCodeX<=BoothX(3 downto 1);
                        when 1=>
                        BoothCodeY<=BoothX(5 downto 3);
                        BoothCodeX<="000";
                        when 2=>
                        BoothCodeY<=BoothX(7 downto 5);
                        BoothCodeX<="000";
                        when others =>
                        null;
                  end case;
            end if;
      end process;
      LOG_ARITH<=RapCfg(17);
      --RAPCfg Words
      --0 1 2  |3 4 |5 6 |7 8 |9    |10 11 12|13   |14 15 16|17 |
      --ALU_OP |XSel|YSel|ASel|X_dir|X_Shft  |y_dir|Y_Shft  |LOG_ARITH|
      Op<=RapCfg(2 downto 0);
      process(clk)--, LOG_ARITH,Op,cnt,BoothDecodeY)**************
      begin
            if(LOG_ARITH='0')then
                  Ysel<=RapCfg(6 downto 5);
                  NUM_SHFTS_Y<=RapCfg(16 downto 14);
                  Dir_Y<=RapCfg(13);
                  RTT_SHF_X<=RapCfg(18) ; --0 Rotate 1 Shift
                  ARTH_LOGC_X<=RapCfg(19);   -- 1 Arithmatic 0 logical
            elsif(clk'event and clk='1')then
```

```
dir_y<='0';
RTT_SHF_X<='1' ; --0 Rotate 1 Shift
ARTH_LOGC_X<='0';   -- 1 Arithmatic 0 logical
case Op is
      when "010"=>

      case cnt is
            when 0=>
            case BoothDecodeY is
                  when "00"=>
                  YSel<="11";
                  NUM_SHFTS_Y<="000";
                  when "01"=>
                  YSel<="01";
                  NUM_SHFTS_Y<="000";
                  when "10"=>
                  Ysel<="01";
                  NUM_SHFTS_Y<="001";
                  when others=>
                  null;
            end case;
            when 1=>
            case BoothDecodeY is
                  when "00"=>
                  YSel<="11";
                  NUM_SHFTS_Y<="100";
                  when "01"=>
                  YSel<="01";
                  NUM_SHFTS_Y<="100";
                  when "10"=>
                  Ysel<="01";
                  NUM_SHFTS_Y<="101";
                  when others=>
                  null;
            end case;
            when 2=>
            case BoothDecodeY is
                  when "00"=>
                  YSel<="11";
                  NUM_SHFTS_Y<="110";
                  when "01"=>
                  YSel<="01";
                  NUM_SHFTS_Y<="110";
                  when "10"=>
```

```
                              Ysel<="01";
                              NUM_SHFTS_Y<="111";
                              when others=>
                              null;
                        end case;
                        when others=>
                        null;
                  end case;
                  when "110"=>
                  Ysel<="00";
                  NUM_SHFTS_Y<="000";
                  when "111"=>
                  Ysel<="00";
                  Num_SHFTS_Y<="000";

                  when others=>
                  Ysel<="00";
                  Num_SHFTS_Y<="000";
            end case;
      end if;
end process;
--RAPCfg Words
--0 1 2  |3 4 |5 6 |7 8 |9    |10 11 12|13   |14 15 16|17 |
--ALU_OP |XSel|YSel|ASel|X_dir|X_Shft  |y_dir|Y_Shft  |LOG_ARITH|

process(clk)--, LOG_ARITH,Op,cnt,BoothDecodeX)
begin
      if(LOG_ARITH='0')then
            Xsel<=RapCfg(4 downto 3);
            NUM_SHFTS_X<=RapCfg(12 downto 10);
            Dir_X<=Rapcfg(9);
            RTT_SHF_Y<=RapCfg(20) ; --0 Rotate 1 Shift
            ARTH_LOGC_Y<=RapCfg(21);   -- 1 Arithmatic 0 logical

      elsif(clk'event and clk='1')then
            dir_x<='0';
            RTT_SHF_Y<='1' ; --0 Rotate 1 Shift
            ARTH_LOGC_Y<='0';   -- 1 Arithmatic 0 logical
            case Op is
                  when "010" =>
                  case cnt is
                        when 0=>
                        case BoothDecodeX is
                              when "00"=>
                              XSel<="11";
                              NUM_SHFTS_X<="000";
                              when "01"=>
                              XSel<="01";
                              NUM_SHFTS_X<="010";
                              when "10"=>
                              Xsel<="01";
                              NUM_SHFTS_X<="011";
                              when others=>
                              null;
                        end case;
                        when 1|2=>
                        XSel<="10";
                        NUM_SHFTS_X<="000";

                        when others=>
                        null;
                  end case;
                  when "110"=>
                  Xsel<="00";
                  NUM_SHFTS_X<="000";
                  when "111"=>
                  Xsel<="00";
                  NUM_SHFTS_X<="000";
                  when others =>
                  Xsel<="00";
                  NUM_SHFTS_X<="000";

            end case;
      end if;
end process;

process(clk)--LOG_ARITH,Op, clk, cnt,P_NX,P_NY)
      variable XY : std_logic;
begin
      if(LOG_ARITH='1')then
            case Op is
                  when "010"=>
                  if(clk'event and clk='1')then
                        XY:=P_NX xor P_NY;
                        case cnt is
                              when 0=>
                              if(XY='1')then
                                    ALU_op<="001";
                              else
                                    ALU_Op<="000";
                              end if;
                              when 1|2=>
                              if(P_NY='1')then
                                    ALU_Op<="001";
                              else
                                    ALU_Op<="000";
                              end if;
                              when others=>
                              null;
                        end case;
                  end if;
                  when "110"|"111"=>
                  Alu_op<="001";
                  when others=>
                  Alu_op<=Op;
            end case;
      else
            Alu_op<=Op;
      end if;
end process;

Px<=BoothX(7);
Pxy<=Px & Py;
process(clk)--Op, Pxy,Pa, cnt,clk, P_NX)
begin
      if(LOG_ARITH='1')then
      case Op is
            when "010"=>
            if(clk'event and clk='1')then
                  case cnt is
                        when 0=>
                        if(P_NX='0')then
                              ASel<="00";
                        else
                              ASel<="11";
                        end if;
                        when 1|2=>
                        ASel<="00";
                        when others=>
                        null;
                  end case;
            end if;
```

```
                  when "110"=>
                  case Pxy is
                        when "01"=>
                        Asel<="01";
                        when "10"=>
                        ASel<="10";
                        when "00"|"11"=>
                        Asel<=Pa &(not Pa);
                        when others=>
                        null;
                  end case;
                  when "111"=>
                  case Pxy is
                        when "01"=>
                        Asel<="10";
                        when "10"=>
                        ASel<="01";
                        when "00"|"11"=>
                        Asel<=(not Pa)& Pa;
                        when others=>
                        null;
                  end case;
                  when others=>
                  ASel<="00";
            end case;
            else ASel<="00";
            end if;
      end process;

end CfgCtl;
```

9 Configurable Linear Feedback Shift Regester Unit

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project            : DRAW
--  File name          : CLFSR.vhd
--  Title            : Configurable Linear Feedback Shift Regester Unit
--  Description       : 16-bit linear shift regester for generation of
PN and other codes
--                     :
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
---------------------------------------------------------------------
----------
--   Revisions   :
--          Date             Author   Revision         Comments
--        04/28/02  Ahmad Aloslaim   Rev 11
--
---------------------------------------------------------------------
---------
---------------------------------------------------------------------
----------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity CLFSR is
      Generic(CLFSR_CONFIG_BITS_WIDTH: integer :=8;DATA_PATH_WIDTH:
integer:=16);
      port(
            CLFSR_CLK: in std_logic; --Clock
            CLFSR_CLR: in std_logic; --Clear
            CLFSR_ENABLE: in std_logic; -- Enable

            CLFSR_CONFIG_BITS: in
std_logic_vector(CLFSR_CONFIG_BITS_WIDTH-1 downto 0);
            -- the configuration bits are as follows:
            --The lower 3 bit for the POLY of the lsr. and the upper
            --three are for the 2or3 stages option, left or midel loca-
tion, and right or midel location.
            -- the upper bits are A, B, and C are used to configure the
MUXs.
            -- A bit 5 (MSB) is for the length of the Shift Regester.
'0' means two reg
            -- and '1' means three reg.
            -- B is used to indacate if the last stage is fed from out-
side or
            -- is febback from the regs to the right.
            --C is used to control wither the shift reg is to be fed
from the right
            -- or not.
            --    the last two bits are used to select the input and
output of the SLR.
            --    For bit 6, 0 means the SLR is connected to the nighbor
SLR.
            --    1 means that the input is from the input_interface.
            --    For bit 7, 0 means the SLR is connected to the nighbour
SLR
            --    1 means the SLR will write it output in 16-bit words
(Serial2Parallel)
            --    that can be routed to RAM or to OUT_inface.

            --    7                       6                       5
4                 3           2           1           0
            --    1=>out_RGT_161=>IN_LFT_16A MUXB MUXC MUX POLY3 POLY2
POLY1
            --    0=>OUT_RGT_10=>IN_LFT_1



            -- INPUTS AND OUTPUTS
            CLFSR_IN_16: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0);
            CLFSR_IN_1_LFT: in std_logic;
            CLFSR_IN_1_RHT: in std_logic;

            CLFSR_OUT_16: out  std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            CLFSR_OUT_1_LFT:out std_logic;
            CLFSR_OUT_1_RGT: out std_logic;
            NEW_DATA_ARVD_FROM_CLFSR: out std_logic





            );
end entity;

architecture BEHAV of CLFSR is
      signal CLFSR_IN_LEFT: std_logic;
      signal CLFSR_IN_RIGHT: std_logic;
      signal CLFSR_OUT_LEFT: std_logic;
      signal CLFSR_OUT_RIGHT: std_logic:='0';

      component SERL2PARALL
            port (
                  CLK: in std_logic;
                  RESET: in std_logic;
                  ENABLE: in STD_LOGIC;
                  SERIAL: in std_logic;
                  PARALL: out std_logic_vector (DATA_PATH_WIDTH-1
downto 0)

                  );
      end component;
      component PRLL2SRL
            generic (DATA_PATH_WIDTH: integer:=16);
            port (
                  CLK: in std_logic;
                  PARALL: in std_logic_vector (DATA_PATH_WIDTH-1 downto
0);
                  SERIAL: out std_logic
                  );
      end component;

      component ffd
            port (
                  CLR : in std_logic;
                  CE : in std_logic;
                  CLK : in std_logic;
                  DATA : in std_logic;
                  Q : out std_logic
                  );
      end component ;
      component mux is
            port (
                  EN : in std_logic;
                  I1 : in std_logic;
                  I2 : in std_logic;
                  S : in std_logic;
                  O : out std_logic
                  );
      end component ;

      component xnor_gate is
            port (
                  I0 : in STD_LOGIC;
                  I1 : in STD_LOGIC;
                  O : out STD_LOGIC
                  );
      end component ;

      component buff is
            port (
                  EN : in std_logic;
                  I : in std_logic;
                  O : out std_logic
                  );
      end component;


      signal  D1: std_logic;
      signal  Q1: std_logic ;
      signal  D2: std_logic ;
      signal  Q2: std_logic ;
      signal  Y1: std_logic ;
      signal  X1: std_logic ;
      signal  Y2: std_logic ;
      signal  X2: std_logic ;
      signal  Y3: std_logic ;
      signal  M1: std_logic ;
      signal  M2: std_logic ;
      signal  M3: std_logic ;
      signal  T1: std_logic:='0' ;
      signal TEMP_P2S: std_logic;
      signal TEMP_S2P: std_logic_vector (DATA_PATH_WIDTH-1 downto 0);
      signal TEMP_CLFSR_OUT_16:  std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
begin




      p2s: PRLL2SRL
      port map(
            CLK=>CLFSR_CLK ,
            PARALL=>CLFSR_IN_16 ,
            SERIAL=>TEMP_P2S
            );
      s2p: SERL2PARALL
```

```
port map(
      CLK=>CLFSR_CLK,
      RESET =>  CLFSR_CLR,
      ENABLE => CLFSR_ENABLE,
      SERIAL=>CLFSR_OUT_RIGHT,
      PARALL=>TEMP_CLFSR_OUT_16

      );


--inputConnection: process (CLFSR_CONFIG_BITS(6),CLFSR_CLK) is
--    begin
--          if rising_edge(CLFSR_CLK) then
--          if CLFSR_CONFIG_BITS(6)='0' then
--                CLFSR_IN_LEFT<=CLFSR_IN_1_LFT;
--                CLFSR_IN_RIGHT <=CLFSR_IN_1_RHT ;
--          else if CLFSR_CONFIG_BITS(6)='1' then
--                      CLFSR_IN_LEFT<=TEMP_P2S;
--                      CLFSR_IN_RIGHT <=CLFSR_IN_1_RHT;
--                end if;
--          end if;
--          end if;
--    end process;

MUX_IN_LFT: mux
port map (
      EN => CLFSR_ENABLE,
      I1 => TEMP_P2S,
      I2 =>CLFSR_IN_1_LFT,
      S =>CLFSR_CONFIG_BITS(6),
      O => CLFSR_IN_LEFT
      );
CLFSR_IN_RIGHT <=CLFSR_IN_1_RHT;
--    MUX_IN_RIGHT: mux   --- not needed
--    port map (
--          EN => CLFSR_ENABLE,
--          I1 =>CLFSR_IN_1_RHT,
--          I2 =>CLFSR_IN_1_RHT,
--          S =>CLFSR_CONFIG_BITS(6),
--          O => CLFSR_IN_RIGHT
--          );
```

```
      --    OutputConnection: process (CLFSR_CONFIG_BITS(7),CLFSR_CLK)
is
      --    begin
      --           if rising_edge(CLFSR_CLK) then
      --          if CLFSR_CONFIG_BITS(7)='0' then
      --                CLFSR_OUT_1_LFT<= CLFSR_OUT_LEFT;
      --                CLFSR_OUT_1_RGT<= CLFSR_OUT_RIGHT ;
      --
      --          else if CLFSR_CONFIG_BITS(7)='1' then
      --                      CLFSR_OUT_1_LFT<= CLFSR_OUT_LEFT;
      --                      TEMP_CLFSR_OUT_16 <= TEMP_S2P ;
      --                end if;
      --          end if;
      --          end if;
      --    end process;

      --    MUX_OUT_LFT: mux   --not needed
      --    port map (
      --          EN => CLFSR_ENABLE,
      --          I1 =>CLFSR_OUT_LEFT,
      --          I2 =>CLFSR_OUT_LEFT,
      --          S =>CLFSR_CONFIG_BITS(7),
      --          O => CLFSR_OUT_1_LFT
      --          );

--    TEMP_CLFSR_OUT_16 <= TEMP_S2P ;
      CLFSR_OUT_1_LFT<= CLFSR_OUT_LEFT;
      CLFSR_OUT_1_RGT<= CLFSR_OUT_RIGHT ;


      --MUX_OUT_RIGHT: mux
      --    port map (
      --          EN => CLFSR_ENABLE,
      --          I1 =>CLFSR_OUT_RIGHT,
      --          I2 =>TEMP_S2P,
      --          S =>CLFSR_CONFIG_BITS(7),
      --          O => CLFSR_OUT_1_RGT
      --          );
```

```
--







CLFSR_OUT_RIGHT <= T1;
FF1:  ffd
port map (
      CLR => CLFSR_CLR,
      CE  => CLFSR_ENABLE,
      CLK => CLFSR_CLK,
      DATA => D1,
      Q => Q1
      );

FF2:  ffd
port map (
      CLR => CLFSR_CLR,
      CE  => CLFSR_ENABLE,
      CLK => CLFSR_CLK,
      DATA => D2,
      Q => Q2
      );

FF3:  ffd
port map (
      CLR => CLFSR_CLR,
      CE  => CLFSR_ENABLE,
      CLK => CLFSR_CLK,
      DATA => Q2,
      Q => T1
      );

------------------------------
MUX1: mux
port map (
      EN => CLFSR_ENABLE,
      I1 =>Q1,
      I2 =>M2,
      S =>CLFSR_CONFIG_BITS(5),
      O => D2
      );

MUX2: mux
port map (
      EN => CLFSR_ENABLE,
      I1 =>M1,
      I2 =>X2,
      S =>CLFSR_CONFIG_BITS(5),
      O => CLFSR_OUT_LEFT
      );

MUX3: mux
port map (
      EN => CLFSR_ENABLE,
      I1 =>CLFSR_IN_LEFT,
      I2 =>M1,
      S =>CLFSR_CONFIG_BITS(4),
      O => D1
      );

MUX4: mux
port map (
      EN => CLFSR_ENABLE,
      I1 =>CLFSR_IN_LEFT,
      I2 =>X2,
      S =>CLFSR_CONFIG_BITS(4),
      O => M2
      );



MUX5: mux
port map (
      EN => CLFSR_ENABLE,
      I1 =>X1,
      I2 =>T1,
      S =>CLFSR_CONFIG_BITS(3),
      O => M3
      );



-------------------------------------------------
```

```
XNOR1: xnor_gate
port map (
      I0 => Y3,
      I1 =>X2,
      O =>M1
      );

XNOR2: xnor_gate
port map (
      I0 => Y2,
      I1 => M3,
      O =>X2
      );

XNOR3: xnor_gate
port map (
      I0 => Y1,
      I1 => CLFSR_IN_RIGHT,
      O =>X1
      );

---------------------------
P3: buff
port map(
      EN  =>CLFSR_CONFIG_BITS(2),
      I =>Q1 ,
      O=> Y3
      );

P2: buff
port map (
      EN  => CLFSR_CONFIG_BITS(1),
      I =>Q2 ,
      O => Y2
      );

P1: buff
port map(
      EN  => CLFSR_CONFIG_BITS(0),
      I =>T1,
      O =>Y1
      );
      process (CLFSR_CLK)
      begin
            if rising_edge(CLFSR_CLK) then
                  CLFSR_OUT_16 <=TEMP_CLFSR_OUT_16;
            end if;
      end process;


      Process  (TEMP_CLFSR_OUT_16 ,CLFSR_CLK)
            variable TEMP_CLFSR_OUT: std_logic_vector(
DATA_PATH_WIDTH-1 downto 0);
            variable CLK_CYCLE: integer:=0;
      begin
      if falling_edge(CLFSR_CLK)then
                  NEW_DATA_ARVD_FROM_CLFSR<='0';

            end if;
            if ( TEMP_CLFSR_OUT/=TEMP_CLFSR_OUT_16) then
                  NEW_DATA_ARVD_FROM_CLFSR<='1';
            end if;

            TEMP_CLFSR_OUT:=TEMP_CLFSR_OUT_16;
      end process;

end architecture ;
```

10 Configurable Linear Feedback Shift Regester interface Unit

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project             : DRAW
--  File name           : CLFSR_INFACE.vhd
--  Title               : Configurable Linear Feedback Shift Regester
interface Unit
--  Description         :
```

```
--                     :
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date             Author   Revision         Comments
--        04/29/02  Ahmad Aloslaim   Rev 10
--
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity CLFSR_INFACE is
      generic (CLFSR_INFACE_CONFIG_BITS_WIDTH: integer:=2;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------

            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_2: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_3: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);



            CLFSR_INFACE_CONFIG_BITS: in std_logic_vector
(CLFSR_INFACE_CONFIG_BITS_WIDTH-1 downto 0);
            ---------------- OUTPUT SIGNALS-----------------------
            -- data Out lines. PN code in 16 bis word every 16 cycles
            CLFSR_INPUT: out std_logic_vector(DATA_PATH_WIDTH-1 downto
0)



            );


end entity;

Architecture BEHAV of CLFSR_INFACE is
begin
      process (RPU_IN_FACE_1,RPU_IN_FACE_2,RPU_IN_FACE_3)
      begin
            case
conv_integer(CLFSR_INFACE_CONFIG_BITS(CLFSR_INFACE_CONFIG_BITS_WIDTH-
1 downto 0))is
                  when 0 =>CLFSR_INPUT<= RPU_IN_FACE_1;
                  when 1 =>CLFSR_INPUT<= RPU_IN_FACE_2;
                  when 2 =>CLFSR_INPUT<= RPU_IN_FACE_3;


                  when others =>null;


            end case;
      end process;
end BEHAV;
```

11 16-bit Full Adder unit

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project             : DRAW
--  File name           : FA16.vhd
--  Title               : Full Adder.
--  Description         : 16-bit Full Adder unit.
--                     :
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date        Author   Revision         Comments
--        01/10/02  A. Alsolaim   Rev 1
--
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
Library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;


entity fa16 is
      generic (DATA_BIT_WIDTH :integer:=16);
      port (
            CIN:in std_logic;
            a,b   :in std_logic_vector(DATA_BIT_WIDTH-1 downto 0);
            S: out std_logic_vector(DATA_BIT_WIDTH-1 downto 0);
            cout: out std_logic);
end fa16;

architecture fa16 of fa16 is
      component fa1
            port (CIN, A, B : in std_logic;
                  S, COUT: out std_logic);
      end component;
      signal one,zero:std_logic;
      signal ci :std_logic_vector (DATA_BIT_WIDTH-1 downto 0);
begin
      one<='1';
      zero<='0';
      adder8:for i in 0 to DATA_BIT_WIDTH-1 generate
            bit0 : if (i = 0) generate
                  b0 : fa1
                  port map (CIN =>cin, A =>a(0) , B =>b(0) , S => s(0), 
COUT => ci(0));
            end generate;
            bitm : if (i > 0)  generate
                  bm : fa1
                  port map (CIN =>ci(i-1), A => a(i), B =>b(i) , S => 
s(i), COUT => ci(i));
            end generate;

      end generate;
      cout<=ci(DATA_BIT_WIDTH-1);
end fa16;
```

12 DRPU input interface unit

```
-----------------------------------------------------------------------
---------
-----------------------------------------------------------------------
----------
--
--  Project             : DRAW
--  File name           : IN_FACE.vhd
--  Title               : DRPU input interface unit.
--  Description       : The DRPU 16-bit input interface unit. It select 
three
--                      : out of the five possible incoming signals.
--  Design Libray       : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)        : AHDL 5.1
--                      :
--  Initialization      : none
--  Notes               :
--                      : Compile in VHDL'93
-----------------------------------------------------------------------
----------
--   Revisions   :
--          Date        Author   Revision         Comments
--        04/5/02  A. Alsolaim   Rev 15
--
-----------------------------------------------------------------------
---------
```

```
----------------------------------------------------------------------
----------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity  IN_FACE is
      generic (IN_FACE_CONFIG_BITS_WIDTH: integer:=5;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------
            --Control signals
            IN_FACE_CONFIG_BITS: in
std_logic_vector(IN_FACE_CONFIG_BITS_WIDTH-1 downto 0);
            INFACE_RESET: in std_logic;  --Active high
            CLK: in std_logic;
            CARRY_IN_FROM_N:in std_logic;
            CARRY_IN_FROM_S:in std_logic;
            CARRY_IN_FROM_E:in std_logic;
            CARRY_IN_FROM_W:in std_logic;


            SRART_HOLD_FROM_N: in std_logic;
            SRART_HOLD_FROM_S: in std_logic;
            SRART_HOLD_FROM_W: in std_logic;
            SRART_HOLD_FROM_E: in std_logic;

            --Data lines from glocal communication channels
            IN_G_1_BUS: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');
            IN_G_2_BUS: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');

            -- Data lines from neighboring RPU
            IN_N_RPU: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');
            IN_S_RPU: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');
            IN_W_RPU: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');
            IN_E_RPU: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0):=(others=>'0');
```

```
            ---------------- OUTPUT SIGNALS-----------------------
            --Done signals from neighboring RPU to local control

            RPU_START_HOLD_FROM_X: out std_logic_vector(3 downto
0):=(others=>'0'); --START from NSWE respectively

            RPU_CARRY_IN_1:out std_logic;
            RPU_CARRY_IN_2:out std_logic;
            --RPU_CARRY_IN_FROM_E:out std_logic;
--          RPU_CARRY_IN_FROM_W:out std_logic;



            -- Data lines selected from neighboring RPU and global going
to the RAPS
            RPU_IN_FACE_1: inout std_logic_vector( DATA_PATH_WIDTH-1
downto 0):=(others=>'0'); --changed to inout to enable the new_data
process to read them.
            RPU_IN_FACE_2: inout std_logic_vector( DATA_PATH_WIDTH-1
downto 0):=(others=>'0');
            RPU_IN_FACE_3: inout std_logic_vector( DATA_PATH_WIDTH-1
downto 0):=(others=>'0');

            NEW_DATA_ARVD_1: out std_logic:='0';
            NEW_DATA_ARVD_2: out std_logic:='0';
            NEW_DATA_ARVD_3: out std_logic:='0'



            );
end IN_FACE;


architecture  BEHAV of IN_FACE is


begin



      --for the clock and control signals it is only a direct connection


      -- the ATART_HOLD signals are passed to the local control
      RPU_START_HOLD_FROM_X <= SRART_HOLD_FROM_N & SRART_HOLD_FROM_S &
SRART_HOLD_FROM_W & SRART_HOLD_FROM_E;


--    RPU_CARRY_IN_FROM_N<= CARRY_IN_FROM_N;
--    RPU_CARRY_IN_FROM_S <= CARRY_IN_FROM_S;
--    RPU_CARRY_IN_FROM_E <= CARRY_IN_FROM_E;
--    RPU_CARRY_IN_FROM_W<= CARRY_IN_FROM_W;


      -- the Data lines entering the RPU are selected based on the confi
bits.
      -- we will use case inside a process.
      -----------------------------------
      -- config number  RPU_IN_1RPU_IN_2RPU_IN_3   CARY1 CARY2
      --    0                      N                 S
W             N      S
      --    1                      N                 S
E             N      S
      --    2                      N                 S
G1      N      S
      --    3                      N                 S
G2      N      S
      -------------
      --    4                      N                 W
E             N      W
      --    5                      N                 W
G1      N      W
      --    6                      N                 W
G2      N      W
      -------------
      --    7                      N                 E
G1      N      E
      --    8                      N                 E
G2      N      E
      -------------
      --    9                      N                 G1
G2      N      E
      ---------------------------------------------------
      --    10                     S                 W
E             S      W
      --    11                     S                 W
G1      S      W
```

```
      --    12                      S                 W
G2            S      W
      -------------
      --    13                      S                 E
G1      S      E
      --    14                      S                 E
G2            S      E
      -------------
      --    15                      S                 G1
G2            S      S
      -------------------------------------------------------
      --    16                      W                 E
G1            W      E
      --    17                      W                 E
G2            W      E
      -----------
      --    18                      W                 G1
G2            W      W
      --------------------------------------------------
      --    19                      E                 G1
G2            E      E




      process
(IN_N_RPU,IN_S_RPU,IN_W_RPU,IN_E_RPU,IN_G_1_BUS,IN_G_2_BUS)
      begin
            case conv_integer(IN_FACE_CONFIG_BITS)is
                  when 0 =>RPU_IN_FACE_1<= IN_N_RPU;  RPU_IN_FACE_2<=
IN_S_RPU; RPU_IN_FACE_3<= IN_W_RPU;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_N ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_S;
                  when 1 => RPU_IN_FACE_1<= IN_N_RPU;  RPU_IN_FACE_2<=
IN_S_RPU; RPU_IN_FACE_3<= IN_E_RPU;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_N ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_S;
                  when 2 => RPU_IN_FACE_1<= IN_N_RPU;  RPU_IN_FACE_2<=
IN_S_RPU; RPU_IN_FACE_3<= IN_G_1_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_N ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_S;
                  when 3 =>RPU_IN_FACE_1<= IN_N_RPU;  RPU_IN_FACE_2<=
IN_S_RPU; RPU_IN_FACE_3<= IN_G_2_BUS;
```

```
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_N ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_S;
                  ----------------------
                  when 4 => RPU_IN_FACE_1<= IN_N_RPU;  RPU_IN_FACE_2<=
IN_W_RPU; RPU_IN_FACE_3<= IN_E_RPU;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_N ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_W;
                  when 5 =>RPU_IN_FACE_1<= IN_N_RPU;  RPU_IN_FACE_2<=
IN_W_RPU; RPU_IN_FACE_3<= IN_G_1_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_N ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_W;
                  when 6 => RPU_IN_FACE_1<= IN_N_RPU;  RPU_IN_FACE_2<=
IN_W_RPU; RPU_IN_FACE_3<= IN_G_2_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_N ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_W;
                  ----------------------------
                  when 7 => RPU_IN_FACE_1<= IN_N_RPU;  RPU_IN_FACE_2<=
IN_E_RPU; RPU_IN_FACE_3<= IN_G_1_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_N ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_E;
                  when 8 => RPU_IN_FACE_1<= IN_N_RPU;  RPU_IN_FACE_2<=
IN_E_RPU; RPU_IN_FACE_3<= IN_G_2_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_N ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_E;
                  --------------
                  when 9 => RPU_IN_FACE_1<= IN_N_RPU;  RPU_IN_FACE_2<=
IN_G_1_BUS; RPU_IN_FACE_3<= IN_G_2_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_N ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_E;
                  -----------------------------------------------------
-----------------------------------------------
                  when 10 =>RPU_IN_FACE_1<= IN_S_RPU;  RPU_IN_FACE_2<=
IN_W_RPU; RPU_IN_FACE_3<= IN_E_RPU;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_S ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_W;
                  when 11 => RPU_IN_FACE_1<= IN_S_RPU;  RPU_IN_FACE_2
<= IN_W_RPU; RPU_IN_FACE_3<= IN_G_1_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_S ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_W;
                  when 12 => RPU_IN_FACE_1<= IN_S_RPU;  RPU_IN_FACE_2
<= IN_W_RPU; RPU_IN_FACE_3<= IN_G_2_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_S ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_W;
                  ----------------------------
                  when 13 => RPU_IN_FACE_1<= IN_S_RPU;  RPU_IN_FACE_2
<= IN_E_RPU; RPU_IN_FACE_3<= IN_G_1_BUS;
```

```
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_S ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_E;
                  when 14 => RPU_IN_FACE_1<= IN_S_RPU;  RPU_IN_FACE_2
<= IN_E_RPU; RPU_IN_FACE_3<= IN_G_2_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_S ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_E;
                  --------------------
                  when 15 => RPU_IN_FACE_1<= IN_S_RPU;  RPU_IN_FACE_2
<= IN_G_1_BUS; RPU_IN_FACE_3<= IN_G_2_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_S ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_S;
                  ----------------------------------------------------
-------------------------------------------------
                  when 16 => RPU_IN_FACE_1<= IN_W_RPU;  RPU_IN_FACE_2
<= IN_E_RPU; RPU_IN_FACE_3<= IN_G_1_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_W ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_E;
                  when 17 => RPU_IN_FACE_1<= IN_W_RPU;  RPU_IN_FACE_2
<= IN_E_RPU; RPU_IN_FACE_3<= IN_G_2_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_W ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_E;
                  ------------------------
                  when 18 => RPU_IN_FACE_1<= IN_W_RPU;  RPU_IN_FACE_2
<= IN_G_1_BUS; RPU_IN_FACE_3<= IN_G_2_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_W ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_W;
                  ----------------------------------------------------
-------------------------------------------------
                  when 19 => RPU_IN_FACE_1<= IN_E_RPU;  RPU_IN_FACE_2
<= IN_G_1_BUS; RPU_IN_FACE_3<= IN_G_2_BUS;
                  RPU_CARRY_IN_1 <= CARRY_IN_FROM_E ; RPU_CARRY_IN_2 <=
CARRY_IN_FROM_E;

                  when others =>RPU_IN_FACE_1<= (others=>'0');
RPU_IN_FACE_2<= (others=>'0'); RPU_IN_FACE_3<= (others=>'0');

            end case;
      end process;




      Process  (RPU_IN_FACE_1,CLK)
            variable TEMP_RPU1: std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            variable TEMP_DATA_ARVD: std_logic:='0';
            variable CLK_CYCLE: integer:=0;
      begin
            if rising_edge(CLK) then
                  if (TEMP_RPU1/=RPU_IN_FACE_1 ) then
                        NEW_DATA_ARVD_1<='1';
                        TEMP_DATA_ARVD:='1';


                  elsif TEMP_DATA_ARVD='1' then

                        NEW_DATA_ARVD_1<='0';
                  end if;
                  TEMP_RPU1:=RPU_IN_FACE_1;
            end if;

      end process;

      Process  (RPU_IN_FACE_2,CLK)
            variable TEMP_RPU2: std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            variable TEMP_DATA_ARVD: std_logic:='0';
            variable CLK_CYCLE: integer:=0;
      begin
            if rising_edge(CLK) then
                  if (TEMP_RPU2/=RPU_IN_FACE_2 ) then
                        NEW_DATA_ARVD_2<='1';
                        TEMP_DATA_ARVD:='1';


                  elsif TEMP_DATA_ARVD='1' then

                        NEW_DATA_ARVD_2<='0';
                  end if;
                  TEMP_RPU2:=RPU_IN_FACE_2;
            end if;
      end process;
      Process  (RPU_IN_FACE_3,CLK)
            variable TEMP_RPU3: std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            variable TEMP_DATA_ARVD: std_logic:='0';
            variable CLK_CYCLE: integer:=0;
      begin
            if rising_edge(CLK) then
                  if (TEMP_RPU3/=RPU_IN_FACE_3 ) then
                        NEW_DATA_ARVD_3<='1';
                        TEMP_DATA_ARVD:='1';


                  elsif TEMP_DATA_ARVD='1' then

                        NEW_DATA_ARVD_3<='0';
                  end if;
                  TEMP_RPU3:=RPU_IN_FACE_3;
            end if;

      end process;





      --    Process  (RPU_IN_FACE_1,CLK)
      --          variable TEMP_RPU1: std_logic_vector(
DATA_PATH_WIDTH-1 downto 0);
      --          variable CLK_CYCLE: integer:=0;
      --          begin
      --          if (TEMP_RPU1/=RPU_IN_FACE_1 ) then
      --                NEW_DATA_ARVD_1<='1';
      --                end if;
      --          if falling_edge(CLK)then
      --          NEW_DATA_ARVD_1<='0';
      --
      --          end if;
      --          TEMP_RPU1:=RPU_IN_FACE_1;
      --    end process;

      Process  (RPU_IN_FACE_2 ,CLK)
            variable TEMP_RPU2: std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            variable CLK_CYCLE: integer:=0;
      begin
            if ( TEMP_RPU2/=RPU_IN_FACE_2) then
                  NEW_DATA_ARVD_2<='1';
            end if;
            if falling_edge(CLK)then
                  NEW_DATA_ARVD_2<='0';

            end if;
            TEMP_RPU2:=RPU_IN_FACE_2;
      end process;

      Process  (RPU_IN_FACE_3 ,CLK)
            variable TEMP_RPU3: std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            variable CLK_CYCLE: integer:=0;
      begin
            if ( TEMP_RPU3/=RPU_IN_FACE_3) then
                  NEW_DATA_ARVD_3<='1';
            end if;
            if falling_edge(CLK)then
                  NEW_DATA_ARVD_3<='0';

            end if;
            TEMP_RPU3:=RPU_IN_FACE_3;
      end process;


end BEHAV;
```

13 DRPU output interface unit

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project            : DRAW
--  File name          : OUT_FACE.vhd
--  Title              : DRPU output interface unit.
--  Description       : The DRPU output interface unit. It is a 16-bit
regester
--                     : with scaling and delay cabablities.
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
```

```
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date        Author  Revision         Comments
--        04/15/02  A. Alsolaim   Rev 12
--
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity  OUT_FACE is
      generic ( OUT_FACE_CONFIG_BITS_WIDTH: integer:=12;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------
            --Control signals
            TO_OTHER_RPU_START :in std_logic;
            OUT_FACE_CLK:in std_logic;



            RAP_1_OUT   : in std_logic_vector(DATA_PATH_WIDTH-1 downto
0):=(others=>'0');
            RAM_A_OUT   : in std_logic_vector(DATA_PATH_WIDTH-1 downto
0):=(others=>'0');
            --          RAM_1_B_OUT : in
std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
            --FROM CLFSR and CSDP
            CLFSR_OUT_16: in  std_logic_vector( DATA_PATH_WIDTH-1
downto 0):=(others=>'0');
            SPRD_OUT : in STD_LOGIC_VECTOR (DATA_PATH_WIDTH-1 downto
0):=(others=>'0') ;

            --line directly from input
            RPU_IN_FACE_1: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0):=(others=>'0');
            RPU_IN_FACE_2: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0):=(others=>'0');

            -- Configuration bits;
            OUT_FACE_CONFIG_BITS: in 
std_logic_vector(OUT_FACE_CONFIG_BITS_WIDTH-1 downto 0):=(oth-
ers=>'0');



            ------------------------- OUTPUT SIGNALS ---------------
            RPU_OUT_1_GBUS : out std_logic_vector ( DATA_PATH_WIDTH-1 
downto 0):=(others=>'0');
            RPU_OUT_2_GBUS : out std_logic_vector ( DATA_PATH_WIDTH-1 
downto 0):=(others=>'0');

            IN_N_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto 
0):=(others=>'0');
            IN_S_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto 
0):=(others=>'0');
            IN_W_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto 
0):=(others=>'0');
            IN_E_RPU: out std_logic_vector( DATA_PATH_WIDTH-1 downto 
0):=(others=>'0');

            RPU_START_HOLD_E: out std_logic; --'1' start, '0' hold
            RPU_START_HOLD_W: out std_logic; --'1' start, '0' hold
            RPU_START_HOLD_N: out std_logic; --'1' start, '0' hold
            RPU_START_HOLD_S: out std_logic --'1' start, '0' hold



            );
end OUT_FACE;


architecture  BEHAV of OUT_FACE is
      Component SCALE_REG is
            port (
                  D : in STD_LOGIC_VECTOR (15 downto 0);--DATA in
                  O : out STD_LOGIC_VECTOR (15 downto 0);  --DATA OUT
                  S : in STD_LOGIC_VECTOR (3 downto 0)  --Number of 
shifts
                  );
      end component ;


      signal SCLD_RAP: std_logic_vector(DATA_PATH_WIDTH-1 downto 
0):=(others=>'0');
      signal SCLD_RAM: std_logic_vector(DATA_PATH_WIDTH-1 downto 
0):=(others=>'0');
```

```
      signal DLYD_RAP: std_logic_vector(DATA_PATH_WIDTH-1 downto 
0):=(others=>'0');
      signal DLYD_RAM: std_logic_vector(DATA_PATH_WIDTH-1 downto 
0):=(others=>'0');
      signal SCALE: integer:=0;
      signal DELAY: integer:=0;
      signal SCALE_BIN: std_logic_vector(3 downto 0):=(others=>'0');
begin

      --ADD SCALING AND DELAY MODULE HERE.
      --This process calculates the delay and scale values.
      ScaleAndDelayValues: Pro-
cess(OUT_FACE_CONFIG_BITS,RPU_IN_FACE_1(3 downto 0))
            variable V_SCALE: integer:=0;
            --
            variable V_DELAY: integer:=0;
      begin
            --Either scale by a fixed amount specified in the Configu-
ration
            -- or scale based on the value of RPU_IN_FACE_1 MAX scale 
is bu 15 bits.
            V_SCALE:= conv_integer(OUT_FACE_CONFIG_BITS(11 downto 
8));--scale RAPout and RAMout
            --by scale value gevin in the configuration. if the value 
in the confiduration
            -- value is 15, then use the RPU-1 input as a scale value.
            --same is true for the delay but it takes its value from 
RPU2.
            --MAX delay is 3 colck cycles through the configuration
            V_DELAY:= conv_integer(OUT_FACE_CONFIG_BITS(7 downto 6));-
-))
            if V_SCALE=15 then
                  V_SCALE:= conv_integer(RPU_IN_FACE_1(3 downto 0));
            end if;
            -- no delay through RAPu2 will be avilable--if V_DELAY=3 
then
            --                V_DELAY:= conv_integer(RPU_IN_FACE_2(3 
downto 0));
            --          end if;
            DELAY<= V_DELAY;
            SCALE<= V_SCALE;
      end process;
      process( SCALE)
      begin
            SCALE_BIN<=conv_std_logic_vector(SCALE,4);
      end process;
      --    Scaling is acomplished by SCALE_REG component
      ScaleRegRAP: SCALE_REG
      port map(
            D => RAP_1_OUT,--DATA in
            O => SCLD_RAP,  --DATA OUT
            S => SCALE_BIN --Number of shifts
            );

      ScaleRegRAM: SCALE_REG
      port map(
            D => RAM_A_OUT,--DATA in
            O => SCLD_RAM,  --DATA OUT
            S => SCALE_BIN --Number of shifts
            );



      RapDelayProc: process(OUT_FACE_CLK)
            variable Num_Cycles:integer:=0;
            variable TEMP_REG1: std_logic_vector(DATA_PATH_WIDTH-1
downto 0);
            variable TEMP_REG2: std_logic_vector(DATA_PATH_WIDTH-1
downto 0);
            variable TEMP_REG3: std_logic_vector(DATA_PATH_WIDTH-1
downto 0);

      begin
            if rising_edge(OUT_FACE_CLK)then
                  if DELAY= 0 then
                        DLYD_RAP <= SCLD_RAP;

                  elsif DELAY=1 then
                        DLYD_RAP<=TEMP_REG1;
                        TEMP_REG1:=SCLD_RAP;


                  elsif DELAY=2 then
                        DLYD_RAP<=TEMP_REG2;
                        TEMP_REG2:=TEMP_REG1;
                        TEMP_REG1:=SCLD_RAP;


                  elsif DELAY=3 then
                        DLYD_RAP<=TEMP_REG3;
                        TEMP_REG3:=TEMP_REG2;
                        TEMP_REG2:=TEMP_REG1;
                        TEMP_REG1:=SCLD_RAP;


                  end if;
            end if;
      end process;

      RamDelayProc: process(OUT_FACE_CLK)
            variable Num_Cycles:integer:=0;
            variable TEMP_1: std_logic_vector(DATA_PATH_WIDTH-1 downto
0);
            variable TEMP_2: std_logic_vector(DATA_PATH_WIDTH-1 downto
0);
            variable TEMP_3: std_logic_vector(DATA_PATH_WIDTH-1 downto
0);

      begin
            if rising_edge(OUT_FACE_CLK)then
                  if DELAY= 0 then
                        DLYD_RAM <= SCLD_RAM;

                  elsif DELAY=1 then
                        DLYD_RAM<=TEMP_1;
                        TEMP_1:=SCLD_RAM;


                  elsif DELAY=2 then
                        DLYD_RAM<=TEMP_2;
                        TEMP_2:=TEMP_1;
                        TEMP_1:=SCLD_RAM;


                  elsif DELAY=3 then
                        DLYD_RAM<=TEMP_3;
                        TEMP_3:=TEMP_2;
                        TEMP_2:=TEMP_1;
                        TEMP_1:=SCLD_RAM;


                  end if;
            end if;
      end process;
      ----  -- no need to select on the START signals,
      ----  -- since one of the signal will be selected by the input
interface.
      ----  process ( OUT_FACE_CONFIG_BITS)
      ----  begin
      ----        case conv_integer(OUT_FACE_CONFIG_BITS(1 downto 0))
is
      ----              -- sending the DONE signal to the other rpus
      ----              when 0 =>RPU_DONE_N <=CTRL_RAP_DONE;
      ----              when 1 => RPU_DONE_S <=CTRL_RAP_DONE;
      ----              when 2 =>RPU_DONE_W <=CTRL_RAP_DONE;
      ----              when 3 =>RPU_DONE_E <=CTRL_RAP_DONE;
      ----              when others =>null;
      ----        end case;



      process ( OUT_FACE_CONFIG_BITS)
      begin
            case conv_integer(OUT_FACE_CONFIG_BITS(1 downto 0)) is --
))
                  -- sending a START signal to other rpu
                  when 0 =>RPU_START_HOLD_E <=TO_OTHER_RPU_START;
                  when 1 => RPU_START_HOLD_W <=TO_OTHER_RPU_START;
                  when 2 =>RPU_START_HOLD_N <=TO_OTHER_RPU_START;
                  when 3 => RPU_START_HOLD_S <=TO_OTHER_RPU_START;

                  when others =>null;
            end case;
      end process;
      --this process will rout the output of RAP and RAM_port-A
      --also will route the inputs from the IN_FACEe (when the RPU is
only routing)
      DataOut: process ( OUT_FACE_CONFIG_BITS,OUT_FACE_CLK)
      begin
            if rising_edge(OUT_FACE_CLK) then
                  case conv_integer(OUT_FACE_CONFIG_BITS(5 downto 2))
is     --))
                        -- 4 bit configuration.  Z is high impedance.
                        ---------------------------
                        --config Number N S           E           W
G1          G2
                        ------------------------------------------------
----------------
                        --          0           Z           Z
Z           Z           RPU1  RPU2
                        --          1           Z           Z
Z           Z           RAP         RAM
                        --          2           Z           Z
Z           Z           RPU2  RPU1
                        --          3           Z           Z
Z           Z           RAM         RAP
                        ------------------------------------------------
---------------
                        --          4           RAP   RAM         Z
Z           Z           Z
                        --          5           RAP   Z           RAM
Z           Z           Z
                        --          6           RAP   Z           Z
RAM   Z           Z
                        ------------------------------------------------
--------------
                        --          7           RAM         RAP   Z
Z           Z           Z
                        --          8            Z          RAP   RAM
Z           Z           Z
                        --          9            Z          RAP   Z
RAM   Z           Z
                        ------------------------------------------------
----------
                        --          10           RAM  Z           RAP
Z           Z           Z
                        --          11           Z          RAM
RAP   Z           Z           Z
                        --          12           Z          Z
RAP   RAM   Z           Z
                        ------------------------------------------------
----------------
                        --          13           RAM  Z           Z
RAP   Z           Z
                        --          14           Z          RAM
Z           RAP   Z           Z
                        --          15           Z          Z
RAM         RAP   Z           Z
                        ------------------------------------------------
------------
```

```
                        -- ADDING SPRD AND LFSR OUTS
                        --4 bit configuration.  Z is high impedance.
                        ---------------------------
                        --config Number N S           E           W
G1          G2
                        -----------------------------------------------
---------------
                        --          0           LFSR  LFSR  LFSR  LFSR
RPU1  RPU2
                        --          1           Z           Z
Z           Z           RAP         RAM
                        --          2           SPRD  SPRD  SPRD  SPRD
RPU2  RPU1
                        --          3           Z           Z
Z           Z           RAM         RAP
                        -----------------------------------------------
---------------
                        --          4           RAP         RAM
Z           Z           Z           Z
                        --          5           RAP         Z
RAM         Z           Z           Z
                        --          6           RAP         Z
Z           RAM   Z           Z
                        -----------------------------------------------
--------------
                        --          7           RAM         RAP   Z
Z           Z           Z
                        --          8            Z          RAP   RAM
Z           Z           Z
                        --          9            Z          RAP   Z
RAM   Z           Z
                        -----------------------------------------------
----------
                        --          10           RAM  Z           RAP
Z           Z           Z
                        --          11           Z          RAM
RAP   Z           Z           Z
                        --          12           Z          Z
RAP   RAM   Z           Z
                        -----------------------------------------------
----------------
                        --          13           RAM  Z           Z
RAP   Z           Z
                        --          14           Z          RAM
Z           RAP   Z           Z
                        --          15           Z          Z
RAM         RAP   Z           Z
```

```
----------------------------------------------
------------

when 0 =>
IN_N_RPU    <= CLFSR_OUT_16;
IN_S_RPU    <=CLFSR_OUT_16;
IN_W_RPU    <=CLFSR_OUT_16;
IN_E_RPU    <= CLFSR_OUT_16;
RPU_OUT_1_GBUS<= RPU_IN_FACE_1;
RPU_OUT_2_GBUS <= RPU_IN_FACE_2;



when 1 =>
IN_N_RPU    <= SPRD_OUT;
IN_S_RPU    <= SPRD_OUT;
IN_W_RPU    <= SPRD_OUT;
IN_E_RPU    <= SPRD_OUT;
RPU_OUT_1_GBUS<= DLYD_RAP;
RPU_OUT_2_GBUS <= DLYD_RAM;

when 2 =>
IN_N_RPU    <= (others =>'Z');
IN_S_RPU    <= (others =>'Z');
IN_W_RPU    <= (others =>'Z');
IN_E_RPU    <= (others =>'Z');
RPU_OUT_1_GBUS<= RPU_IN_FACE_2;
RPU_OUT_2_GBUS <= RPU_IN_FACE_1;
when 3 =>
IN_N_RPU    <= (others =>'Z');
IN_S_RPU    <= (others =>'Z');
IN_W_RPU    <= (others =>'Z');
IN_E_RPU    <= (others =>'Z');
RPU_OUT_1_GBUS<= DLYD_RAM;
RPU_OUT_2_GBUS <=DLYD_RAP ;

when 4 =>
IN_N_RPU    <= DLYD_RAP;
IN_S_RPU    <= RPU_IN_FACE_2;--DLYD_RAM;
IN_W_RPU    <= (others =>'Z');
IN_E_RPU    <= (others =>'Z');
RPU_OUT_1_GBUS<= (others =>'Z');
RPU_OUT_2_GBUS <= (others =>'Z');
when 5 =>
```

```
IN_N_RPU    <= DLYD_RAP;
IN_S_RPU    <=(others =>'Z');
IN_W_RPU    <= DLYD_RAM;
IN_E_RPU    <= (others =>'Z');
RPU_OUT_1_GBUS<= (others =>'Z');
RPU_OUT_2_GBUS <= (others =>'Z');
when 6 =>
IN_N_RPU    <= DLYD_RAP;
IN_S_RPU    <= (others =>'Z');
IN_W_RPU    <= (others =>'Z');
IN_E_RPU    <= DLYD_RAM;
RPU_OUT_1_GBUS<= (others =>'Z');
RPU_OUT_2_GBUS <= (others =>'Z');
when 7 =>
IN_N_RPU    <= DLYD_RAM;
IN_S_RPU    <= DLYD_RAP;
IN_W_RPU    <= (others =>'Z');
IN_E_RPU    <= (others =>'Z');
RPU_OUT_1_GBUS<= (others =>'Z');
RPU_OUT_2_GBUS <= (others =>'Z');
when 8 =>
IN_N_RPU    <= (others =>'Z');
IN_S_RPU    <= DLYD_RAP;
IN_W_RPU    <= DLYD_RAM;
IN_E_RPU    <= (others =>'Z');
RPU_OUT_1_GBUS<= (others =>'Z');
RPU_OUT_2_GBUS <= (others =>'Z');
when 9 =>
IN_N_RPU    <= (others =>'Z');
IN_S_RPU    <= DLYD_RAP;
IN_W_RPU    <= (others =>'Z');
IN_E_RPU    <= DLYD_RAM;
RPU_OUT_1_GBUS<= (others =>'Z');
RPU_OUT_2_GBUS <= (others =>'Z');
when 10 =>
IN_N_RPU    <= DLYD_RAM;
IN_S_RPU    <= (others =>'Z');
IN_W_RPU    <= DLYD_RAP;
IN_E_RPU    <= (others =>'Z');
RPU_OUT_1_GBUS<= (others =>'Z');
RPU_OUT_2_GBUS <= (others =>'Z');
when 11 =>
IN_N_RPU    <= (others =>'Z');
IN_S_RPU    <= DLYD_RAM;
```

```
                        IN_W_RPU    <= DLYD_RAP;
                        IN_E_RPU    <= (others =>'Z');
                        RPU_OUT_1_GBUS<= (others =>'Z');
                        RPU_OUT_2_GBUS <= (others =>'Z');
                        when 12 =>
                        IN_N_RPU    <= (others =>'Z');
                        IN_S_RPU    <= (others =>'Z');
                        IN_W_RPU    <= DLYD_RAP;
                        IN_E_RPU    <= DLYD_RAM;
                        RPU_OUT_1_GBUS<= (others =>'Z');
                        RPU_OUT_2_GBUS <= (others =>'Z');
                        when 13 =>
                        IN_N_RPU    <= DLYD_RAM;
                        IN_S_RPU    <= (others =>'Z');
                        IN_W_RPU    <= (others =>'Z');
                        IN_E_RPU    <=DLYD_RAP;
                        RPU_OUT_1_GBUS<= (others =>'Z');
                        RPU_OUT_2_GBUS <= (others =>'Z');
                        when 14 =>
                        IN_N_RPU    <= (others =>'Z');
                        IN_S_RPU    <= DLYD_RAM;
                        IN_W_RPU    <= (others =>'Z');
                        IN_E_RPU    <= DLYD_RAP;
                        RPU_OUT_1_GBUS<= (others =>'Z');
                        RPU_OUT_2_GBUS <= (others =>'Z');
                        when 15 =>
                        IN_N_RPU    <= (others =>'Z');
                        IN_S_RPU    <= (others =>'Z');
                        IN_W_RPU    <= DLYD_RAM;
                        IN_E_RPU    <= DLYD_RAP;
                        RPU_OUT_1_GBUS<= (others =>'Z');
                        RPU_OUT_2_GBUS <= (others =>'Z');

                        when others => null;
                  end case;
            end if;
      end process;



end BEHAV;
```

14 Paralle to serial unit

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project            : DRAW
--  File name          : PRLL2SRL.vhd
--  Title              : Parall to serial unit.
--  Description        : A 16-bits parallel to serial formating unit
--                     :
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date        Author  Revision         Comments
--        03/29/02  A. Alsolaim   Rev 2
--
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity PRLL2SRL is
      generic (DATA_PATH_WIDTH: integer:=16);
      port (
      CLK: in std_logic;
      PARALL: in std_logic_vector (DATA_PATH_WIDTH-1 downto 0);
      SERIAL: out std_logic
      );
end entity;

Architecture BEHAV of PRLL2SRL is
```

```
begin


      P2S: process(CLK)

        variable i: integer:=0;
      begin
             if rising_edge(CLK) then
             SERIAL<=PARALL(i);
             i:=i+1;
             end if;
             if i>15 then
                   i:=0;
             end if;
      end process;

end BEHAV;
```

15 RAM/FIFO unit

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project            : DRAW
--  File name          : RAM_FIFO.vhd
--  Title              : RAM/FIFO unit.
--  Description       : A RAM unit that can be configured to be a FIFO
Unit.
--                     :
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
```

```
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date        Author  Revision          Comments
--        02/19/02  A. Alsolaim   Rev 10
--
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;
use IEEE.numeric_std.all;

entity RAM_FIFO is
      --Dual port RAM constructed of regesters.
      generic (DATA_PATH_WIDTH     :natural := 16
;RAM_CONFIG_BITS_WIDTH: integer :=1;--RAM width
            RAM_ADRS_WIDTH  :natural := 3--RAM depth =
2^RAM_ADRS_WIDTH.
            );

      port (
            RAM_A_IN : in std_logic_vector (DATA_PATH_WIDTH-1 downto
0);
            --RAM_B_IN : in std_logic_vector (DATA_PATH_WIDTH-1 downto
0);

            RAM_A_ADRS: in std_logic_vector (RAM_ADRS_WIDTH-1 downto
0);
            RAM_B_ADRS : in std_logic_vector (RAM_ADRS_WIDTH-1 downto
0); -- Port B is only used for reading.
            -- No input for port B since it is only read port. and no
outpout for port B also
            -- since it can use port A output while port A is writing.

            RAM_WR: in std_logic;-- 0 no op, 1 write. --Port A
            RAM_RD: in std_logic;-- 0 no op, 1 READ.   --Port A
            RAM_ENABLE: in std_logic;  -- synchronous, '1' enabled.
            RAM_CLEAR: in std_logic; -- synchronous Clear
            RAM_CONFIG_BITS:instd_logic:='0';
            --'0' for RAM behavior and
            -- '1' for FIFO behavior
            RAM_CLK: in std_logic;

            FROM_RAM_FIFO_FULL: out std_logic; --'1' for full
            FROM_RAM_FIFO_EMPTY: out std_logic; --'1' for empty
            FROM_RAM_A_OUT :out std_logic_vector (DATA_PATH_WIDTH-1
downto 0)
            --    FROM_RAM_B_OUT :out std_logic_vector
(DATA_PATH_WIDTH-1 downto 0)
            );

end RAM_FIFO;


architecture RAM_FIFO of RAM_FIFO is

      type mem_type is array (2**RAM_ADRS_WIDTH-1 downto 0) of
      std_logic_vector(DATA_PATH_WIDTH - 1 downto 0) ;
      Signal RD_ptr, WR_ptr: integer range 0 to 2**RAM_ADRS_WIDTH;--
Read and Write pointers
      --For the FIFO
begin
      Main_Process:
            process (RAM_CLK,RAM_CONFIG_BITS)

            variable mem : mem_type;
            -- This stores the number of elements needs to be saved.
            --    variable fifo : mem_type ;



      begin

            case  RAM_CONFIG_BITS is
                  when '0' =>
                  if rising_edge(RAM_CLK) then
                        if RAM_ENABLE = '1' then
                              if RAM_CLEAR = '1' then
                                    for INDEX in 2**RAM_ADRS_WIDTH-1
downto 0 loop
                                          mem(INDEX) := (others => '0');
-- clear the ram.
                                    end loop;
                              else
                                    if  (RAM_WR ='1')then  --write oper-
ation ## -- CLEAR ='0' RAM is enabled for reading or writing
```

```
mem(conv_integer(RAM_A_ADRS)):=RAM_A_IN;
                                   end if;
                                   if(RAM_RD ='1')then  --Read opera-
tion ##     --reading at the rising edge
                                         FROM_RAM_A_OUT <=
mem(conv_integer(RAM_B_ADRS));

                                   end if;
                             end if;
                       end if;
                 end if;





                 --end RAM_Process.

                 when '1' => --FIFO behvior
                 if rising_edge(RAM_CLK) then
                       if RAM_ENABLE = '1' then
                             if RAM_CLEAR = '1' then
                                   for INDEX in 2**RAM_ADRS_WIDTH-1
downto 0 loop
                                         mem(INDEX) := (others => '0');
-- clear the ram.
                                   end loop;
                             else -- CLEAR ='0' FIFO is enabled for
reading or writing
                                   --FIFO writing operation
                                   if (RAM_WR ='1')then   ---   ##
                                         mem(WR_ptr):=RAM_A_IN; --stor
the input at the writing pointer

                                   end if;
                                   if (RAM_RD ='1')then    -- ###
                                         FROM_RAM_A_OUT<=mem(WR_ptr);
--read at the reading pointer
                                   end if;
                             end if;
                       end if;
                 end if;
                 when others => null;
           end case;
```

```
      end process;


      FIFO_ADRS_PTR: process(RAM_CLK)
            variable PTR : INTEGER range 0 to 16;
      begin
            if rising_edge(RAM_CLK) then
                  if RAM_ENABLE = '1' then
                        if RAM_CLEAR = '1'then --Clear all pointers
                              WR_ptr <= 0;
                              RD_ptr <= 0;
                              FROM_RAM_FIFO_EMPTY <= '1';
                              FROM_RAM_FIFO_FULL <= '0';
                              PTR := 0;
                        elsif RAM_WR ='1' and PTR <  2**RAM_ADRS_WIDTH
then    --##
                              if WR_PTR < 15 then
                                    WR_PTR <= WR_PTR + 1;
                              elsif WR_PTR = 15 then
                                    WR_PTR <= 0;
                              end if;
                              PTR := PTR + 1;
                        elsif RAM_RD ='1' and PTR > 0 then  ---##
                              if RD_PTR<15 then
                                    RD_PTR <= RD_PTR + 1;
                              elsif RD_PTR = 15 then
                                    RD_PTR <= 0;
                              end if;
                              PTR := PTR - 1;
                        end if;

                        if PTR = 0 then
                              FROM_RAM_FIFO_EMPTY <= '1';
                        else
                              FROM_RAM_FIFO_EMPTY <= '0';
                        end if;

                        if PTR = 16 then
                              FROM_RAM_FIFO_FULL<= '1';
                        else
                              FROM_RAM_FIFO_FULL <= '0';
                        end if;

                  end if;
            end if;

      end process;
end RAM_FIFO;
```

16 RAM/FIFO Interface

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project            : DRAW
--  File name          : RAM_FIFO_FACE.vhd
--  Title              : RAM/FIFO Interface
--  Description        : The interface of the RAM/FIFO unit.
--                     :
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.4
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date        Author  Revision         Comments
--        03/10/02  A. Alsolaim   Rev 8
--
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity  RAM_FIFO_FACE is
      generic (RAM_FIFO_FACE_CONFIG_BITS_WIDTH: integer:=6;
            DATA_PATH_WIDTH: integer:=16; RAM_ADRS_WIDTH:integer:=3);
      port (

            ---------------- INPUT SIGNALS-----------------------
            RAM_FIFO_INFACE_CLK:in std_logic;


            --Data lines from RAP
            RAP_OUT: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_2: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_3: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            -- Data from the CLFSR
            CLFSR_OUT_16: in STD_LOGIC_VECTOR (DATA_PATH_WIDTH-1 downto
0);


            NEW_DATA_ARVD_1: in std_logic;
            NEW_DATA_ARVD_2: in std_logic;
            NEW_DATA_ARVD_3: in std_logic;
            NEW_DATA_ARVD_FROM_RAP: in std_logic; -- the Done_runing
signal out of the RAP
            NEW_DATA_ARVD_FROM_CLFSR: in std_logic;
            --configuration bits
            RAM_FACE_CONFIG_BITS: in
std_logic_vector(RAM_FIFO_FACE_CONFIG_BITS_WIDTH-1 downto 0);


            ---------------- OUTPUT SIGNALS-----------------------
            --CLK to the RAM

            --Data and ADRS signal to RAM
            RAM_A_IN : out std_logic_vector (DATA_PATH_WIDTH-1 downto
0) ;
            RAM_A_ADRS_OUT: out std_logic_vector (RAM_ADRS_WIDTH-1
downto 0);
            RAM_B_ADRS_OUT: out std_logic_vector (RAM_ADRS_WIDTH-1
downto 0);

            NEW_DATA_ARVD_2RAM: out std_logic

            );
```

```
end RAM_FIFO_FACE;


architecture  BEHAV of RAM_FIFO_FACE is

begin


      SelectInputForRAM: Process
(RAM_FACE_CONFIG_BITS,RPU_IN_FACE_1,RPU_IN_FACE_2,RPU_IN_FACE_3,RAP_O
UT,CLFSR_OUT_16)

      begin

            case
conv_integer(RAM_FACE_CONFIG_BITS(RAM_FIFO_FACE_CONFIG_BITS_WIDTH-1
downto 0)) is
                  ---------------
                  --          inputs
                  --    1     RPU1
                  --    2     RPU2
                  --    3     RPU3
                  --  4   RAP
                  --  5   CLFSR
                  ------------------note: input 5 CLFSR in only used as
DATA_A
                  -------------
                  -- Out puts
                  -- DATA_A ADRS_A ADRS_B
                  --0      1       2    3
                  --1        1       2    4
                  --2        1       3    2
                  --3        1       3    4
                  --4        1       4    2
                  --5        1       4    3
                  -------------------
                  --6        2       1    3
                  --7        2       1    4
                  --8        2       3    1
                  --9        2       3    4
                  --10   2       4    1
                  --11   2       4    3
                  ------------------------
                  --12   3       1    2
                  --13   3       1    4
                  --14   3       2    1
                  --15   3       2    4
                  --16   3       4    1
                  --17   3       4    2
                  ------------------------
                  --18   4       1    2
                  --19   4       1    3
                  --20   4       2    1
                  --21   4       2    3
                  --22   4       3    1
                  --23   4       3    2
                  -------------------------
                  --24   5       1    2
                  --25   5       1    3
                  --26   5       1    4
                  --27   5       2    1
                  --28   5       2    3
                  --29   5       2    4
                  --30   5       3    1
                  --31   5       3    2
                  --32   5       3    4
                  --33   5       4    1
                  --34   5       4    2
                  --35   5       4    3
```

```
                  when 0 => RAM_A_IN <=  RPU_IN_FACE_1;
RAM_A_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0)  ;
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_1;

                  when 1 => RAM_A_IN <=  RPU_IN_FACE_1;
RAM_A_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM<=NEW_DATA_ARVD_1 ;

                  when 2 => RAM_A_IN <=  RPU_IN_FACE_1;
RAM_A_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_1 ;
                  when 3 => RAM_A_IN <=  RPU_IN_FACE_1;
RAM_A_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_1 ;

                  when 4 => RAM_A_IN <=  RPU_IN_FACE_1;
RAM_A_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_1 ;

                  when 5 => RAM_A_IN <=  RPU_IN_FACE_1;
RAM_A_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_1 ;
                  ----------------------------------------------------
--------------------------------

                  when 6 => RAM_A_IN <=  RPU_IN_FACE_2;
RAM_A_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_2 ;

                  when 7 => RAM_A_IN <=  RPU_IN_FACE_2;
RAM_A_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_2 ;

                  when 8 => RAM_A_IN <=  RPU_IN_FACE_2;
RAM_A_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_2 ;

                  when 9 => RAM_A_IN <=  RPU_IN_FACE_2;
RAM_A_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_2 ;

                  when 10 => RAM_A_IN <=  RPU_IN_FACE_2;
RAM_A_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_2 ;

                  when 11 => RAM_A_IN <=  RPU_IN_FACE_2;
RAM_A_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_2 ;
                  -----------------------------------------------------
---------------------------------

                  when 12 => RAM_A_IN <=  RPU_IN_FACE_3;
RAM_A_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_3 ;

                  when 13 => RAM_A_IN <=  RPU_IN_FACE_3;
RAM_A_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_3 ;

                  when 14 => RAM_A_IN <=  RPU_IN_FACE_3;
RAM_A_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_3 ;

                  when 15 => RAM_A_IN <=  RPU_IN_FACE_3;
RAM_A_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_3 ;

                  when 16 => RAM_A_IN <=  RPU_IN_FACE_3;
RAM_A_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_3 ;

                  when 17 => RAM_A_IN <=  RPU_IN_FACE_3;
RAM_A_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_3 ;
                  -----------------------------------------------------
-----------------------------------

                  when 18 => RAM_A_IN <=  RAP_OUT;
RAM_A_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_RAP ;

                  when 19 => RAM_A_IN <=  RAP_OUT;
RAM_A_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_RAP ;
                  when 20 => RAM_A_IN <=  RAP_OUT;
RAM_A_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_RAP ;

                  when 21 => RAM_A_IN <=  RAP_OUT;
RAM_A_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_RAP ;

                  when 22 => RAM_A_IN <=  RAP_OUT;
RAM_A_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_RAP ;

                  when 23 => RAM_A_IN <=  RAP_OUT;
RAM_A_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_RAP ;
                  -----------------------------------------------------
-----------------------

                  when 24 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;

                  when 25 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;

                  when 26 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;

                  when 27 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;

                  when 28 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;
                  when 29 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;

                  when 30 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;

                  when 31 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;

                  when 32 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;

                  when 33 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RPU_IN_FACE_1(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;

                  when 34 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto 0)
;RAM_B_ADRS_OUT<=RPU_IN_FACE_2(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;

                  when 35 => RAM_A_IN <=  CLFSR_OUT_16;
RAM_A_ADRS_OUT<=RAP_OUT(RAM_ADRS_WIDTH-1 downto
0);RAM_B_ADRS_OUT<=RPU_IN_FACE_3(RAM_ADRS_WIDTH-1 downto 0);
                  NEW_DATA_ARVD_2RAM <=NEW_DATA_ARVD_FROM_CLFSR ;



                  when others => null;
            end case;

      end process;



end BEHAV;
```

17 Dynamically Reconfigurable Processing Unit (DRAP)

```
---------------------------------------------------------------------
---------
---------------------------------------------------------------------
----------
--
--  Project            : DRAW
--  File name          : RAP.vhd
--  Title              : Dynamically Reconfigurable Processing Unit
--  Description        : 16-bit processing unit
--                     :
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : AHDL 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
---------------------------------------------------------------------
----------
--   Revisions   :
--         Date             Author  Revision         Comments
--        04/23/02  A. Alsolaim   Rev 13
--
---------------------------------------------------------------------
---------
---------------------------------------------------------------------
----------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity  RAP16 is
      generic ( RAP_CONFIG_BITS_WIDTH: integer:=22; -- Added two more
            --bits (for the barral shifter) see barral.vhd
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------
            RAP_CLK: in std_logic; --Active high
            RAP_ENABLE: in std_logic; --Active high
            RAP_RESET: in std_logic;  --Active high
            --    RAP_START_HOLD: in std_logic; --'1' start, '0' hold
__ REMOVED. was only needed in CfgCtrl enetity. removed from there.
            RAP_CONFIG_BITS: in std_logic_vector(
RAP_CONFIG_BITS_WIDTH -1 downto 0);
            RAP_X_IN: in std_logic_vector (DATA_PATH_WIDTH -1 downto
0);
            RAP_Y_IN: in std_logic_vector (DATA_PATH_WIDTH -1 downto
0);
            RAP_CRY_IN: in std_logic;
            -- RPU_IN_FACE_1 is also an input to be used by the
num_shft_gen as the variable X
            -- in the multiplication.
            --RPU_IN_FACE_1: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);

            -- rap8_1 is now configuration bit number 18
            -- RAP_LOG_ARITH: in std_logic; --'1' Arithmatic, '0' bit-
wise logical

            RAP_CARY_OUT : out std_logic;
            RAP_OVR_FLW : out std_logic;

            RAP_DONE_RUNING: out std_logic;--'1' done, '0' Runing
            RAP_OUT: out std_logic_vector (DATA_PATH_WIDTH -1 downto 0)
            ----------------------------------------AHMAD 1/20/02--
------------------
            -- Add the two inputs to the BRL shifter to control the type
of the
            -- shift, either rotate or shift. and also the type of shift
either
            -- Logical (fill with zeros) or Arithmatic (fill with the
MSB or
            -- LSB depending on the direction of shift
            --RAP_CONFIG_BITS(20) 0 Rotate 1 Shift
            --RAP_CONFIG_BITS(21) 1 Arithmatic 0 logical
            );


end RAP16;



architecture  BEHAV of RAP16 is


      component ALU16  --Configuration bit =3
```

```
            generic (DATA_BIT_WIDTH : integer:=16);
            port(
                  ALU_X_IN : in std_logic_vector (DATA_BIT_WIDTH-1
downto 0);
                  ALU_Y_IN : in std_logic_vector (DATA_BIT_WIDTH-1
downto 0);
                  AluCfg : in std_logic_vector (2 downto 0);
                  CARY_IN : in std_logic;
                  ALU_CLK : in std_logic;
                  ALU_CLEAR : in std_logic;
                  ALU_ENABLE : in std_logic;
                  LOG_ARITH : in std_logic;
                  CARY_OUT : out std_logic;
                  OVR_FLW : out std_logic;
                  ALU_OUT : out std_logic_vector (DATA_BIT_WIDTH-1
downto 0)
                  );
      end component;

      component CfgCtl
            generic (RAP_CONFIG_BITS_WIDTH: integer:=22;
                  CFGCTRL_DATA_PATH_WIDTH: integer:=8);
            port (
                  RapCfg: in std_logic_vector( RAP_CONFIG_BITS_WIDTH -
1 downto 0);
                  BoothX: in std_logic_vector( CFGCTRL_DATA_PATH_WIDTH-
1 downto 0);
                  Py : in std_logic;
                  Pa : in std_logic;clk : in std_logic;
                  --    start: in std_logic;
                  en : in std_logic;
                  DIR_X : out STD_LOGIC;
                  DIR_Y : out STD_LOGIC;
                  NUM_SHFTS_X : out STD_LOGIC_VECTOR (2 downto 0);
                  NUM_SHFTS_Y : out STD_LOGIC_VECTOR (2 downto 0);
                  YSel : out STD_LOGIC_VECTOR (1 downto 0);
                  XSel : out STD_LOGIC_VECTOR (1 downto 0);
                  RTT_SHF_X: out STD_LOGIC:='1'; --0 Rotate 1 Shift
                  RTT_SHF_Y: out STD_LOGIC:='1'; --0 Rotate 1 Shift
                  ARTH_LOGC_X: out std_logic:='0'; -- 1 Arithmatic 0
logical
                  ARTH_LOGC_Y: out std_logic:='0'; -- 1 Arithmatic 0
logical
                  ASel : out STD_LOGIC_VECTOR (1 downto 0);
                  ALU_Op : out  STD_LOGIC_VECTOR (2 downto 0);
                  StateCnt: out std_logic_vector(1 downto 0)
                  );
      end component;

      component BRL_SFT_16
            generic(DATA_WIDTH : integer:=16);
            port (
                  DIR : in STD_LOGIC;   --1 right 0 left
                  X_IN : in STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
                  Y_OUT : out STD_LOGIC_VECTOR (DATA_WIDTH-1 downto 0);
                  NUM_SHFTS : in STD_LOGIC_VECTOR (3 downto 0);
                  RTT_SHF: in STD_LOGIC:='1'; --0 Rotate 1 Shift
                  ARTH_LOGC: in std_logic:='0' -- 1 Arithmatic 0 logical
                  );
      end component ;

      component MUX41
            generic (DATA_PATH_WIDTH: integer:=16);
            port( X0: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
                  X1: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
                  X2: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
                  X3: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
                  Y: out std_logic_vector( DATA_PATH_WIDTH-1 downto 0);
                  Sel: in std_logic_vector(1 downto 0);
                  en: in std_logic
                  );
      end component;

      component REG16
            generic(DWidth : integer:=16);
            port (
                  REG_CLR : in std_logic;
                  REG_CLK : in std_logic;
                  X_IN : in std_logic_vector (DWidth-1 downto 0);
                  Y_OUT : out std_logic_vector (DWidth-1 downto 0)
                  );
      end component;

      component Two_Cmpl16
            generic (DATA_BIT_WIDTH:integer:=16);
            port(
                  x : in std_logic_vector(DATA_BIT_WIDTH-1 downto 0);
                  x_Cmpl : out std_logic_vector(DATA_BIT_WIDTH-1 downto
0)
```

```
                  );
      end component;
      component REG_NO_REG_16
            generic(DWidth : integer:=16);
            port (
                  REG_NO_REG: in std_logic;  -- 1 reg, 0 donot reg
                  REG_CLR : in std_logic;
                  REG_CLK : in std_logic;
                  X_IN : in std_logic_vector (DWidth-1 downto 0);
                  Y_OUT : out std_logic_vector (DWidth-1 downto 0)
                  );
      end component;

      signal NUM_SHFTS_X: std_logic_vector(2 downto 0);
      signal NUM_SHFTS_Y: std_logic_vector(2 downto 0);
      signal DIR_X,tmpRapClk, tmp2RapClk, RapRegClk: std_logic;
      signal DIR_Y: std_logic;
      signal SHFTED_X: std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
      signal SHFTED_Y: std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
      --signal OUT_TO_ACCUM:std_logic_vector(DATA_PATH_WIDTH-1 downto
0);
      signal BoothX : std_logic_vector(7 downto 0);
      signal RapXin, ALU_X:std_logic_vector(DATA_PATH_WIDTH-1 downto
0);
      signal RapYin, ALU_Y: std_logic_vector(DATA_PATH_WIDTH-1 downto
0);
      signal NegAluOut, ALU_OUT: std_logic_vector(DATA_PATH_WIDTH-1
downto 0);
      signal tmpRapOut, ALU_MUX_OUT: std_logic_vector(DATA_PATH_WIDTH-
1 downto 0);
      signal  ext,clk4, clk3,clk2, CRY_OUT: std_logic;
      signal Py, Pa, OVR_FLW: std_logic;
      --configuration signals
      signal AluCfg,AluOp : std_logic_vector (2 downto 0); --3 bits
      signal ASel1D, Asel,XSel,YSel,StateCnt: std_logic_vector(1
downto 0);
      --    signal YSel: std_logic_vector(1 downto 0);
      signal zero :std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
      signal extalucfg, NumShftsX4,NumShftsY4 :std_logic_vector (3
downto 0);
      signal TEMP_RAP_OUT: std_logic_vector (DATA_PATH_WIDTH -1 downto
0);
      signal RTT_SHF_X : STD_LOGIC; --0 Rotate 1 Shift
      signal RTT_SHF_Y:  STD_LOGIC; --0 Rotate 1 Shift
      signal ARTH_LOGC_X:  std_logic; -- 1 Arithmatic 0 logical
      signal ARTH_LOGC_Y:  std_logic; -- 1 Arithmatic 0 logical

begin
      --RAP_CONFIG_BITS Words
      --0 1 2  |3 4 |5 6 |7 8 |9    |10 11 12|13   |14 15 16|17 |
      --ALU_OP |XSel|YSel|ASel|X_dir|X_Shft  |y_dir|Y_Shft  |ext|
      AluCfg <=RAP_CONFIG_BITS(2 downto 0);  --3 bits

      ext <= RAP_CONFIG_BITS(17);
      zero<=(others=>'0');
      clk3<=not(StateCnt(1) or StateCnt(0));
      clk2<=not StateCnt(0);
      extAluCfg<=ext & AluCfg;  --bits 17-1-0



      process(extAluCfg,RAP_Y_IN, RAP_X_IN, clk2, clk3,RAP_CLK, 
RapRegClk)
      begin
            if(ext='1')then
                  case extAluCfg is
                        when "1010"=>  --
                        if(clk3'event and clk3='1')then
                              RapXin<=RAP_Y_IN;
                              RapYin<=RAP_Y_IN;
                              BoothX <=RAP_X_IN(7 downto 0);
                        end if;
                        when "1110" |"1111"=>
                        if(Clk2'event and Clk2='1')then
                              RapXin<=RAP_X_IN;
                              RapYin<=RAP_Y_IN;
                              BoothX<=(others=>'0');
                        end if;
                        when others=>
                        if(Rap_Clk'event and Rap_Clk='1')then
                              RapXin<=RAP_X_IN;
                              RapYin<=RAP_Y_IN;
                              BoothX<=(others=>'0');
                        end if;
                  end case;
            else
                  RapXin<=RAP_X_IN;
                  RapYin<=RAP_Y_IN;
            end if;
end process;

NumShftsX4<='0' & NUM_SHFTS_X;
NumShftsY4<='0' & NUM_SHFTS_Y;

U1X_BShift16: BRL_SFT_16
port map (
      DIR=>DIR_X,
      X_IN =>RapXin,
      Y_OUT=> SHFTED_X,
      NUM_SHFTS=> NumShftsX4,
      RTT_SHF=>  RTT_SHF_X,
      ARTH_LOGC=> ARTH_LOGC_X
      );
U2Y_BShift16: BRL_SFT_16
port map (
      DIR =>DIR_Y,
      X_IN =>RapYin,
      Y_OUT=> SHFTED_Y,
      NUM_SHFTS => numShftsY4,
      RTT_SHF=>  RTT_SHF_Y,
      ARTH_LOGC=> ARTH_LOGC_Y
      );


U3_MUX41: MUX41
port map(
      X0 =>RapXin,
      X1=>SHFTED_X,
      X2=>tmpRapOut,
      X3=>zero,
      Y=>ALU_X,
      Sel=>XSel,
      en =>RAP_ENABLE
      );
U4_Mux41: MUX41
port map(
      X0=> RapYin,
      X1=> SHFTED_Y,
      X2=>zero,
      X3=>zero,
      Y =>ALU_Y,
      Sel=>YSel,
      en =>RAP_ENABLE
```

```
            );
      U5_Mux41:Mux41
      port map(
            X0=> ALU_OUT,
            X1=> RapXin,
            X2=> RapYin,
            X3=> NegAluOut,
            Y => tmpRapOut,
            Sel=>ASel1D,
            en =>RAP_ENABLE
            );

      process(ext, AluCfg,RAP_CLK,ASel)
      begin
            if((ext & AluCfg)="1010")then
                  if(RAP_CLK'event and RAP_CLK='1')then
                        ASel1D<=ASel;
                  end if;
            else
                  Asel1D<=Asel;
            end if;
      end process;

      U6_ALU: ALU16
      port map(
            ALU_X_IN =>ALU_X,
            ALU_Y_IN=>ALU_Y ,
            AluCfg => AluOp,
            CARY_IN=>RAP_CRY_IN,
            ALU_CLK => RAP_CLK,
            ALU_CLEAR => RAP_RESET,
            ALU_ENABLE => RAP_ENABLE,
            LOG_ARITH =>RAP_CONFIG_BITS(17), --
<<<<<<<<<<***********************************
            CARY_OUT=> CRY_OUT,
            OVR_FLW =>OVR_FLW,
            ALU_OUT =>ALU_OUT
            );
      Py<=RapYin(DATA_PATH_WIDTH-1);
      Pa<=Alu_out(DATA_PATH_WIDTH-1);
      U7_CfgCtl:CfgCtl
      port map(
            RapCfg=>RAP_CONFIG_BITS,
            BoothX=>BoothX,
            Py=>Py,
            Pa=>Pa,
            clk=>RAP_CLK,
            --    start=>RAP_START_HOLD,  --removed from CfgCtrl entity
            en =>RAP_ENABLE,
            DIR_X =>DIR_X,
            DIR_Y =>DIR_Y,
            NUM_SHFTS_X =>NUM_SHFTS_X,
            NUM_SHFTS_Y =>NUM_SHFTS_y,
            YSel=> YSel,
            XSel=> XSel,
            RTT_SHF_X=>RTT_SHF_X, --0 Rotate 1 Shift
            RTT_SHF_Y=>RTT_SHF_Y, --0 Rotate 1 Shift
            ARTH_LOGC_X=>ARTH_LOGC_X, -- 1 Arithmatic 0 logical
            ARTH_LOGC_Y=>ARTH_LOGC_Y, -- 1 Arithmatic 0 logical
            ASel=> ASel,
            ALU_Op=>AluOp,
            StateCnt=>StateCnt
            );
      ----  U8_REG: REG16
      ----  port map (
      ----        REG_CLR=>RAP_RESET,
      ----        REG_CLK => RapRegClk,
      ----        X_IN=>tmpRapOut,
      ----        Y_OUT => TEMP_RAP_OUT
      ----        );

      U8_REG: REG_NO_REG_16
      port map(
            REG_NO_REG =>'0', -- 1 reg, 0 donot reg
            REG_CLR =>RAP_RESET,
            REG_CLK => RapRegClk,
            X_IN =>tmpRapOut,
            Y_OUT  => TEMP_RAP_OUT
            );






      --    tmpRapClk<=(not RAP_CLK) and (StateCnt(0) and (not State-
Cnt(1)));
      process(clk3,RAP_CLK)
```

```
begin
      if(RAP_CLK'event and RAP_CLK='1')then
            clk4<=clk3;
      end if;
end process;

process(RAP_CLK,AluCfg,ext,StateCnt)
begin
      if((ext & AluCfg)="0010" )then
            RapRegClk<=clk3 and (not RAP_CLK);
      elsif((ext & AluCfg(2 downto 1))="011")then
            --RapRegClk<=(not Rap_Clk) and StateCnt(0);
            RapRegClk<=StateCnt(0) and (not RAP_CLK);
      else
            RapRegClk<=RAP_CLK;
      end if;
end process;
U9_Two_Cmpl16:Two_cmpl16
port map(
      x=>ALU_OUT,
      x_Cmpl=>NegAluOut
      );



RAP_CARY_OUT<=  CRY_OUT;




counter:process(RAP_CLK,RAP_ENABLE,tmpRapOut)
      variable COUNT :integer range 0 to 3:=0;

begin
      if(RAP_ENABLE='0')then
            COUNT:=3;
      elsif(RAP_CLK'event and RAP_CLK='1')then
            case COUNT is
                  when 3=>
                  COUNT:=0;
                  when 2=>
                  COUNT:=3;
                  when 1=>
                  COUNT:=2;
                  when 0=>
```

```
                        COUNT:=1;
                        when others=>
                        COUNT:=0;
                  end case;
            end if;
            if  COUNT=3 then
                  RAP_OUT<=tmpRapOut;
                  end if;
      end process;

      --
      --RAP_OUT<=tmpRapOut;   -- jump
      --RAP_OUT<=TEMP_RAP_OUT;
      RAP_OVR_FLW<=  OVR_FLW ;

      Process  (TEMP_RAP_OUT ,RAP_CLK)
            variable TEMP_OUT: std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            variable CLK_CYCLE: integer:=0;
      begin
            if ( TEMP_OUT/=TEMP_RAP_OUT) then
                  RAP_DONE_RUNING<='1';
            end if;
            if falling_edge(RAP_CLK)then
                  RAP_DONE_RUNING<='0';

            end if;
            TEMP_OUT:=TEMP_RAP_OUT;
      end process;



end BEHAV;
```

18 DRAP interface unit

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
```

```
--  Project             : DRAW
--  File name           : RAP_INTR_FC.vhd
--  Title               : DRAP interface unit.
--  Description       : The DRAP interface. It routs the incoming signal
to the DRAP unit.
--                      :
--  Design Libray       : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)        : AHDL 5.1
--                      :
--  Initialization      : none
--  Notes               :
--                      : Compile in VHDL'93
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date              Author  Revision         Comments
--        4/22/02  A. Alsolaim   Rev 4
--
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;


-- Synthisizable and number of gates =797



entity  RAP_INTR_FC is
      generic (RAP_INTR_FC_CONFIG_BITS_WIDTH: integer:=5;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------

            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
```

```
            RPU_IN_FACE_2: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_3: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);

            RPU_CARRY_IN_1:in std_logic;
            RPU_CARRY_IN_2:in std_logic;



            -- data lines from the RAP
            FROM_RAP: in std_logic_vector( DATA_PATH_WIDTH-1 downto 0);

            -- data lines from RAMs
            FROM_RAM_A: in std_logic_vector( DATA_PATH_WIDTH-1 downto
0);

            --data from SPRD unit
            SPRD_OUT : in STD_LOGIC_VECTOR (DATA_PATH_WIDTH-1 downto
0);

            RAP_INTR_FC_CONFIG_BITS: in std_logic_vector
(RAP_INTR_FC_CONFIG_BITS_WIDTH-1 downto 0);
            ---------------- OUTPUT SIGNALS-----------------------

            -- data lines, X, and Y
            RAP_X_IN: out std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
            RAP_Y_IN: out std_logic_vector(DATA_PATH_WIDTH-1 downto 0);
            RAP_CRY_IN: out std_logic


            );
end RAP_INTR_FC;


architecture  BEHAV of RAP_INTR_FC is
      signal TEMP_RAP_X_IN: std_logic_vector(DATA_PATH_WIDTH-1 downto
0);
      signal TEMP_RAP_Y_IN: std_logic_vector(DATA_PATH_WIDTH-1 downto
0);

begin
```

```
--    --the 4 LSB bits of the configuration are used to select one of
the inputs
--    --the MSB is used to flip the inputs or not
      process (RPU_IN_FACE_1,RPU_IN_FACE_2,RPU_IN_FACE_3,FROM_RAP,
FROM_RAM_A,SPRD_OUT )
      begin
            case
conv_integer(RAP_INTR_FC_CONFIG_BITS(RAP_INTR_FC_CONFIG_BITS_WIDTH-2
downto 0))is
                  when 0 =>TEMP_RAP_X_IN<= RPU_IN_FACE_1;
TEMP_RAP_Y_IN<= RPU_IN_FACE_1; RAP_CRY_IN <= RPU_CARRY_IN_1;
                  when 1 =>TEMP_RAP_X_IN<= RPU_IN_FACE_1;
TEMP_RAP_Y_IN<= RPU_IN_FACE_2;  RAP_CRY_IN <= RPU_CARRY_IN_1;
                  when 2 =>TEMP_RAP_X_IN<= RPU_IN_FACE_1;
TEMP_RAP_Y_IN<= RPU_IN_FACE_3;  RAP_CRY_IN <= RPU_CARRY_IN_1;
                  when 3 =>TEMP_RAP_X_IN<= RPU_IN_FACE_1;
TEMP_RAP_Y_IN<= FROM_RAM_A; RAP_CRY_IN <= RPU_CARRY_IN_1;
                  when 4 =>TEMP_RAP_X_IN<= RPU_IN_FACE_1;
TEMP_RAP_Y_IN<= SPRD_OUT;    RAP_CRY_IN <= RPU_CARRY_IN_1;
                  ----------------------------------------------------
-----------------------
                  when 5 =>TEMP_RAP_X_IN <= RPU_IN_FACE_2;
TEMP_RAP_Y_IN <= RPU_IN_FACE_2; RAP_CRY_IN <= RPU_CARRY_IN_2;
                  when 6 =>TEMP_RAP_X_IN <= RPU_IN_FACE_2;
TEMP_RAP_Y_IN <= RPU_IN_FACE_3;  RAP_CRY_IN <= RPU_CARRY_IN_2;
                  when 7 =>TEMP_RAP_X_IN <= RPU_IN_FACE_2;
TEMP_RAP_Y_IN <= FROM_RAM_A; RAP_CRY_IN <= RPU_CARRY_IN_2;
                  when 8 =>TEMP_RAP_X_IN <= RPU_IN_FACE_2;
TEMP_RAP_Y_IN <= SPRD_OUT; RAP_CRY_IN <= RPU_CARRY_IN_2;
                  ----------------------------------------------------
------------------------
                  when 9 =>TEMP_RAP_X_IN <= RPU_IN_FACE_3;
TEMP_RAP_Y_IN <= RPU_IN_FACE_3;  RAP_CRY_IN <= RPU_CARRY_IN_2;
                  when 10 =>TEMP_RAP_X_IN <= RPU_IN_FACE_3;
TEMP_RAP_Y_IN <= FROM_RAM_A; RAP_CRY_IN <= RPU_CARRY_IN_2;
                  when 11 =>TEMP_RAP_X_IN <= RPU_IN_FACE_3;
TEMP_RAP_Y_IN <= SPRD_OUT;   RAP_CRY_IN <= RPU_CARRY_IN_1;
                  ----------------------------------------------------
-------------------------
                  when 12 =>TEMP_RAP_X_IN <= FROM_RAM_A; TEMP_RAP_Y_IN
<= FROM_RAM_A;  RAP_CRY_IN <= '0';
                  when 13 =>TEMP_RAP_X_IN <= FROM_RAM_A; TEMP_RAP_Y_IN
<= SPRD_OUT;  RAP_CRY_IN <= '0';
                  ----------------------------------------------------
-----------------------
                  when 14 =>TEMP_RAP_X_IN <= SPRD_OUT;  TEMP_RAP_Y_IN
<= SPRD_OUT;  RAP_CRY_IN <= '0';
```

```
                  when others =>null;


            end case;
      end process;

      -- data lines, X, and Y
      process (TEMP_RAP_Y_IN,TEMP_RAP_X_IN)
      begin
            case
conv_integer(RAP_INTR_FC_CONFIG_BITS(RAP_INTR_FC_CONFIG_BITS_WIDTH-
1))is
                  when 0 =>RAP_X_IN<= TEMP_RAP_X_IN;  RAP_Y_IN
<=TEMP_RAP_Y_IN ;
                  when others =>  RAP_X_IN<= TEMP_RAP_Y_IN;  RAP_Y_IN
<=TEMP_RAP_X_IN ;


            end case;
      end process;
end BEHAV;
```

19 Scaling Register

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project             : DRAW
--  File name           : SCAL_REG.vhd
--  Title               : Scaling Register
--  Description       : 16-bit Sscaling regester. Scales the input data
according
--                      : to scale value "S" in one clock cycle.
--  Design Libray       : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)        : Ahdl 5.1
--                      :
--  Initialization      : none
--  Notes              :
--                     : Compile in VHDL'93
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date             Author   Revision         Comments
--        02/21/02  Ahmad Aloslaim      Rev 1
--
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
library IEEE;
use IEEE.std_logic_1164.all;

entity SCALE_REG is
      port (
            D : in STD_LOGIC_VECTOR (15 downto 0);
            O : out STD_LOGIC_VECTOR (15 downto 0);
            S : in STD_LOGIC_VECTOR (3 downto 0)
            );
end entity ;

architecture BEHAV of SCALE_REG is

      function MUX2 (A, B, S: std_logic) return std_logic is
      begin
            if S='1' then
                  return A;
            else
                  return B;
            end if;
      end function;


      signal TEMP0 : STD_LOGIC_VECTOR (15 downto 0);
      signal TEMP1 : STD_LOGIC_VECTOR (15 downto 0);
      signal TEMP2 : STD_LOGIC_VECTOR (15 downto 0);

begin

      TEMP0(0) <= MUX2(D(1), D(0), S(0));
      TEMP0(1) <= MUX2(D(2), D(1), S(0));
      TEMP0(2) <= MUX2(D(3), D(2), S(0));
      TEMP0(3) <= MUX2(D(4), D(3), S(0));
```

```
TEMP0(4) <= MUX2(D(5), D(4), S(0));
TEMP0(5) <= MUX2(D(6), D(5), S(0));
TEMP0(6) <= MUX2(D(7), D(6), S(0));
TEMP0(7) <= MUX2(D(8), D(7), S(0));
TEMP0(8) <= MUX2(D(9), D(8), S(0));
TEMP0(9) <= MUX2(D(10), D(9), S(0));
TEMP0(10) <= MUX2(D(11), D(10), S(0));
TEMP0(11) <= MUX2(D(12), D(11), S(0));
TEMP0(12) <= MUX2(D(13), D(12), S(0));
TEMP0(13) <= MUX2(D(14), D(13), S(0));
TEMP0(14) <= MUX2(D(15), D(14), S(0));
TEMP0(15) <= MUX2('0', D(15), S(0));

TEMP1(0) <= MUX2(TEMP0(2), TEMP0(0), S(1));
TEMP1(1) <= MUX2(TEMP0(3), TEMP0(1), S(1));
TEMP1(2) <= MUX2(TEMP0(4), TEMP0(2), S(1));
TEMP1(3) <= MUX2(TEMP0(5), TEMP0(3), S(1));
TEMP1(4) <= MUX2(TEMP0(6), TEMP0(4), S(1));
TEMP1(5) <= MUX2(TEMP0(7), TEMP0(5), S(1));
TEMP1(6) <= MUX2(TEMP0(8), TEMP0(6), S(1));
TEMP1(7) <= MUX2(TEMP0(9), TEMP0(7), S(1));
TEMP1(8) <= MUX2(TEMP0(10), TEMP0(8), S(1));
TEMP1(9) <= MUX2(TEMP0(11), TEMP0(9), S(1));
TEMP1(10) <= MUX2(TEMP0(12), TEMP0(10), S(1));
TEMP1(11) <= MUX2(TEMP0(13), TEMP0(11), S(1));
TEMP1(12) <= MUX2(TEMP0(14), TEMP0(12), S(1));
TEMP1(13) <= MUX2(TEMP0(15), TEMP0(13), S(1));
TEMP1(14) <= MUX2('0', TEMP0(14), S(1));
TEMP1(15) <= MUX2('0', TEMP0(15), S(1));

TEMP2(0) <= MUX2(TEMP1(4), TEMP1(0), S(2));
TEMP2(1) <= MUX2(TEMP1(5), TEMP1(1), S(2));
TEMP2(2) <= MUX2(TEMP1(6), TEMP1(2), S(2));
TEMP2(3) <= MUX2(TEMP1(7), TEMP1(3), S(2));
TEMP2(4) <= MUX2(TEMP1(8), TEMP1(4), S(2));
TEMP2(5) <= MUX2(TEMP1(9), TEMP1(5), S(2));
TEMP2(6) <= MUX2(TEMP1(10), TEMP1(6), S(2));
TEMP2(7) <= MUX2(TEMP1(11), TEMP1(7), S(2));
TEMP2(8) <= MUX2(TEMP1(12), TEMP1(8), S(2));
TEMP2(9) <= MUX2(TEMP1(13), TEMP1(9), S(2));
TEMP2(10) <= MUX2(TEMP1(14), TEMP1(10), S(2));
TEMP2(11) <= MUX2(TEMP1(15), TEMP1(11), S(2));
TEMP2(12) <= MUX2('0', TEMP1(12), S(2));
TEMP2(13) <= MUX2('0', TEMP1(13), S(2));
```

```
      TEMP2(14) <= MUX2('0', TEMP1(14), S(2));
      TEMP2(15) <= MUX2('0', TEMP1(15), S(2));

      O(0) <= MUX2(TEMP2(8), TEMP2(0), S(3));
      O(1) <= MUX2(TEMP2(9), TEMP2(1), S(3));
      O(2) <= MUX2(TEMP2(10), TEMP2(2), S(3));
      O(3) <= MUX2(TEMP2(11), TEMP2(3), S(3));
      O(4) <= MUX2(TEMP2(12), TEMP2(4), S(3));
      O(5) <= MUX2(TEMP2(13), TEMP2(5), S(3));
      O(6) <= MUX2(TEMP2(14), TEMP2(6), S(3));
      O(7) <= MUX2(TEMP2(15), TEMP2(7), S(3));
      O(8) <= MUX2('0', TEMP2(8), S(3));
      O(9) <= MUX2('0', TEMP2(9), S(3));
      O(10) <= MUX2('0', TEMP2(10), S(3));
      O(11) <= MUX2('0', TEMP2(11), S(3));
      O(12) <= MUX2('0', TEMP2(12), S(3));
      O(13) <= MUX2('0', TEMP2(13), S(3));
      O(14) <= MUX2('0', TEMP2(14), S(3));
      O(15) <= MUX2( TEMP2(15), TEMP2(15), S(3));

end architecture BEHAV;
```

20 Serial to Paralle unit

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project            : DRAW
--  File name          : SERL2PARALL.vhd
--  Title              : Serail to Parallel unit
--  Description        : 16-bit Serial to parallel formating unit
--                     :
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : Ahdl 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
```

```
---------------------------------------------------------------------
----------
--   Revisions   :
--          Date             Author   Revision         Comments
--        04/1/02  Ahmad Aloslaim      Rev 3
--
---------------------------------------------------------------------
---------
---------------------------------------------------------------------
----------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity SERL2PARALL is
      generic (DATA_PATH_WIDTH: integer:=16);
      port (
            CLK: in std_logic;
            RESET: in std_logic;
            ENABLE: in STD_LOGIC;
            SERIAL: in std_logic:='0';
            PARALL: out std_logic_vector (DATA_PATH_WIDTH-1 downto 0)

            );
end entity;


Architecture BEHAV of  SERL2PARALL is



begin
      process(RESET,CLK)
      variable count: integer:=0;
      variable TEMP_PARALL: std_logic_vector(15 downto 0);
      begin
            if RESET='1' then
                  PARALL<=(others=>'0');
            else
                  if rising_edge(CLK)then

                        if count<16 then
                        TEMP_PARALL(count):=SERIAL;
                        count:=count+1 ;
                        end if;
                        if count=16 then
                              count:=0;
                              if count= 0 then
                              PARALL<=TEMP_PARALL;
                              end if;
                        end if;

                  end if;
            end if;
      end process;




end BEHAV;
```

21 Configurable Spreading Data Path Unit (CSDP)

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project            : DRAW
--  File name          : SPRD_UNIT.vhd
--  Title              : Spreading Unit
--  Description        : 16-bit Spreading/de-spreading Unit
--                     :
--  Design Libray      : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)       : Ahdl 5.1
--                     :
--  Initialization     : none
--  Notes              :
--                     : Compile in VHDL'93
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date             Author   Revision         Comments
--         04/6/02  Ahmad Aloslaim      Rev 4
--
---------------------------------------------------------------------
---------
---------------------------------------------------------------------
----------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity SPRD_UNIT is
      generic(DATA_PATH_WIDTH : integer:=16);
      port (
            SPRD_CLK : in STD_LOGIC;  --Clock
            SPRD_ENABLE : in STD_LOGIC;  --Enable
            SPRD_RESET: in std_logic;
            DATA_IN : in STD_LOGIC_VECTOR (DATA_PATH_WIDTH-1 downto 0);
            PN1: in std_logic_vector (DATA_PATH_WIDTH-1 downto 0); --
PN1 is converted inside the SDP
            --unit to serial stream PN1_SRL.
            SPRD_OUT : out STD_LOGIC_VECTOR (DATA_PATH_WIDTH-1 downto
0)

            );
end entity SPRD_UNIT;


architecture BEHAV of SPRD_UNIT is

      signal REG_16_BITS_1   : std_logic_vector(DATA_PATH_WIDTH-1
downto 0):=(others=>'0');
      signal PN1_SRL: std_logic;
      signal CLK16:std_logic:='0';
      --    signal READY_TO_SPRD: std_logic:='0';
begin

      GenCLK16: Process(SPRD_CLK)
            variable i: integer:=0;
      begin
            if SPRD_ENABLE='1' then
                  if rising_edge(SPRD_CLK) then
                        if i>15 then
                              i:=0;
                        end if;
                        if i=0 then
                              CLK16<=not CLK16;
                              i:=i+1;
                        else
                              i:=i+1;
                        end if;
                  end if;
            else null;
            end if;
      end process;

      --Load PN1 into Reg1
      process (SPRD_RESET,PN1, clk16)
      begin
            if SPRD_ENABLE='1' then
                  if (SPRD_RESET = '0') then
                        REG_16_BITS_1    <= (Others => '0');

                  else
                        if rising_edge(CLK16) then
                              REG_16_BITS_1<=PN1;

                        end if;
                  end if;
            else null;
            end if;
      end process;

      p2s: process (SPRD_CLK)
            variable i: integer :=0;
      begin
            if SPRD_ENABLE='1' then
                  if (rising_edge(SPRD_CLK)) then
                        PN1_SRL <= REG_16_BITS_1(i)  ;

                        i:=i+1;
                        if i=16 then i:=0; end if;
                  end if;
            else null;
            end if;
      end process;
```

```
      Spreading:process(SPRD_CLK,DATA_IN,PN1_SRL)
      begin
            if SPRD_ENABLE='1' then
                  if (rising_edge(SPRD_CLK))then

                        case PN1_SRL is
                              when '0' =>SPRD_OUT <= DATA_IN ;
                              when '1' =>  SPRD_OUT <= NOT DATA_IN ;
                              when others => null;
                        end case;
                  end if;
            else null;
            end if;
      end process;



End BEHAV;
```

22 Configurable Spreading Data Path Unit Interface

```
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
--
--  Project             : DRAW
--  File name           : SPRD_UNIT.vhd
--  Title               : Spreading Unit Interface
--  Description         : 16-bit Spreading/de-spreading intrface Unit
--                      :
--  Design Libray       : DRAW.lib
--  Analysis Dependency: none
--  Simulator(s)        : Ahdl 5.1
--                      :
--  Initialization      : none
--  Notes               :
--                      : Compile in VHDL'93
```

```
----------------------------------------------------------------------
----------
--   Revisions   :
--          Date             Author   Revision         Comments
--        04/1/02  Ahmad Aloslaim      Rev 3
--
----------------------------------------------------------------------
---------
----------------------------------------------------------------------
----------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.Std_Logic_Unsigned.all;

entity SPRD_INFACE is
      generic (SPRD_INTR_FC_CONFIG_BITS_WIDTH: integer:=3;
            DATA_PATH_WIDTH: integer:=16);
      port (

            ---------------- INPUT SIGNALS-----------------------

            -- data lines from the RPU IN_FACE
            RPU_IN_FACE_1: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_2: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);
            RPU_IN_FACE_3: in std_logic_vector( DATA_PATH_WIDTH-1
downto 0);



            SPRD_INTR_FC_CONFIG_BITS: in std_logic_vector
(SPRD_INTR_FC_CONFIG_BITS_WIDTH-1 downto 0);
            ---------------- OUTPUT SIGNALS-----------------------

            -- data Out lines, Data, and PN
            SPRD_DATA: out std_logic_vector(DATA_PATH_WIDTH-1 downto
0);
            SPRD_PN: out std_logic_vector(DATA_PATH_WIDTH-1 downto 0)



            );
```

```
end entity;
Architecture BEHAV of SPRD_INFACE is

begin


            process (RPU_IN_FACE_1,RPU_IN_FACE_2,RPU_IN_FACE_3)
      begin
            case
conv_integer(SPRD_INTR_FC_CONFIG_BITS(SPRD_INTR_FC_CONFIG_BITS_WIDTH-
1 downto 0))is
                  when 0 =>SPRD_DATA<= RPU_IN_FACE_1;  SPRD_PN<=
RPU_IN_FACE_2;
                  when 1 =>SPRD_DATA<= RPU_IN_FACE_1;  SPRD_PN<=
RPU_IN_FACE_3;
                  when 2 =>SPRD_DATA<= RPU_IN_FACE_2;  SPRD_PN<=
RPU_IN_FACE_1;
                  when 3 =>SPRD_DATA<= RPU_IN_FACE_2;  SPRD_PN<=
RPU_IN_FACE_3;
                  when 4 =>SPRD_DATA<= RPU_IN_FACE_3;  SPRD_PN<=
RPU_IN_FACE_1;
                  when 5 =>SPRD_DATA<= RPU_IN_FACE_3;  SPRD_PN<=
RPU_IN_FACE_2;


                  when others =>null;


            end case;
      end process;

      end BEHAV;
```

References

[1] R. Weiss, “FPGA + Technology = DSP,” DSPnet, www.techonline.com DSP featured article no. 7120, Nov. 2000.

[2] R. Weiss, “DSPs are Hot,” DSPnet, www.techonline.com DSP featured article no. 7119, Nov. 2000.

[3] E. Mirsky and A. DeHon, “MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources,” IEEE Symposium on FPGAs for Custom Computing Machines, April. 1996, Napa, CA, pp. 157 -166.

[4] B. Radunovic and V. Milutinovic, “A survey of Reconfigurable Computing Architectures,” Proc. of FPL'98 Eighth International Workshop on Field Programable Logic and Application, Tallinn, Estonia, Sept. 1998.

[5] J.Villasenor, “*The Flexibility of Configurable Computing*,” IEEE Signal Processing Magazine, Vol. 15, No. 5 , Sept. 1998, pp. 67 -84.

[6] Xilinx data book sheets, http://www.xilinx.com/partinfo/databook.htm.

[7] Altera data book sheets, http://www.altera.com/products/prd-index.html.

[8] R. Sutton, V. Srini, and J. Rabaey, “A Multiprocessor DSP System Using PADDI-2,” Proc. of IEEE Design Automation Conference (DAC), 1998, San Francisco, CA, pp. 62 -65.

[9] S. Goldstein et al. “*PipeRench: A Reconfigurable Architecture and Compiler*,” IEEE Computer Magazine, Vol. 33, No. 4, pp. 70 -77, April 2000.

[10] G. Lu, et al. “The Morphosys Parallel Reconfigurable System,” Proc. of Euro-Par 99, Toulouse, France, Sept. 1999.

[11] T. Callahan, et al. “*The Grap Architecture and C Compiler,*” IEEE Computer Magazine, April 2000.

[12] Ray Bittner, and Peter Athanas, “Wormhole Run-time Reconfiguration,” Proc. of FPGA'97, Monterrey, CA, February, 1997.

[13] R. Hartenstein, “Coarse Grain Reconfigurable Architectures,” Coarse Grain Reconfigurable Architecture ASP-DAC 2001, Asia and South Pacific Design Automation Conference 2001, Jan. 2001, Yokohama, Japan, pp. 564 -569.

[14] R. Abielmona, “Alphabetical List of Reconfigurable Computing Architectures,” http://www.site.uottawa.ca/~rabielmo/personal/rc.html

[15] R. Hartenstein, “A Decade of Reconfigurable Computing: A Visionary Retrospective,” Proc. DATE 2001 Conf., Munich, Germany, pp. 642, Mar., 2001.

[16] K. Compton, and S. Hauck, “Configurable Computing: A Survey of Systems and Software,” Northwestern University, Dept. of ECE, Technical Report, 1999.

[17] R. Hartenstein “Using The KressArray for Reconfigurable Computing,” Conf. on Configurable: Technology and Applications, Boston, Nov. 1998.

[18] M. Lee, et al. “*Design and Implementation of the MorphoSys Reconfigurable Computing Processor*,” Journal of VLSI and Signal Processing Systems, Mar. 2000.

[19] M. Wan, et al. “*Design Methodology of a Low-Energy Reconfigurable Single-Chip DSP System*”, Journal of VLSI Signal Processing, Mar. 2000.

[20] J. Rabaey, “Reconfigurable Processing: The Solution to Low Power Programmable DSP,” Proc. of 1997 ICASSP Conference, Munich, Vol. 1 , pp. 275 - 278, April 1997.

[21] Chameleon Systems Corp. web site http://www.chameleonsystems.com/

[22] MorphICs web site, http://www.morphics.com.

[23] G. Smit, et al. “Chameleon Reconfigureability in Hand-held Multimedia Computers,” Proc. First International Symposium on Handheld and Ubiquitous Computing, HUC'99, Karlsruhe, Germany, September 1999.

[24] R. Hartenstein, et al. "*A Novel Machine Paradigm to Accelerate Scientific Computing*," Special issue on Scientific Computing of Computer Science and Informatics Journal, Computer Society of India, 1996.

[25] A. Alsolaim, J. Becker, J. Starzyk, and M. Glesner, "Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems," IEEE FCCM'00 San Jose, CA, pp. 205 -214, May. 2000

[26] A. Alsolaim, J. Becker, M.Glesner, and J. Starzyk, "A Dynamically Reconfigurable System-on-a-Chip Architecture for Future Mobile Digital Signal Processing," European Signal Processing Conf. EUSIPCO2000, Nov. 1999.

[27] A. Alsolaim, and J. Starzyk, "Dynamically Reconfigurable Solution in the Digital Baseband Processing for Future Mobile Radio Devices," SSST'2001, Athens, Ohio, pp. 221 -226, Mar. 2001.

[28] J. Becker, A. Alsolaim, M. Glesner, and J. Starzyk, "Fast Communication Mechanisms in Coarse-grained Dynamically Reconfigurable Array Architecture," The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000), Las Vegas, Nevada, June 26 2000.

[29] 3GPP Web Site, http://www.3gpp.org.

[30] R. Parsad, W. Mohr, and W. Konhauser, **Third Generation Mobile Communication Systems**, Artech House Publisher, Boston - London, 2000.

[31] W. Granzow, “3rd Generation Mobile Communications Systems,” Course notes for Mobile Communication II. Friedrich-Alexander-Universität Erlangen-Nürnberg.

[32] E. Dahlman, “*WCDMA-The Radio Interface for Future Mobile Multimedia Communications,*” IEEE Trans. on Vehicular Technology, Nov. 1998.

[33] T. Ojanpera, and R. Parasad, **Wideband CDMA for Third Generation Mobile Communications**, Boston-London, Artech House Pub., 1998.

[34] S. Ohmori, Y. Yamao, and N. Nakajima, “*The Future Generations of Mobile Communications Based on Broadband Access Technologies,*” IEEE Communications Magazine, Vol. 38, No. 12, Dec. 2000, pp. 134 -142.

[35] R. Steele, and L. Hanzo, **Mobile Radio Communications Second and Third Generation Cellular and WATM Systems**, John Wiley and Sons, New York, 1999.

[36] 3GPP2 Web Site, http://www.3gpp2.org.

[37] [3GPP Technical Specification, “Spreading and Modulation (FDD),” 3GPP Document no. 3G TS 25.213, Ver. 3.2.0, Mar. 2000.

[38] 3GPP Technical Specification, “UE Radio transmission and Reception (FDD),” 3GPP Document no. 3G TS 25.101, Ver. 3.2.2, Apr. 2000.

[39] 3GPP Technical Specification, “Physical Channels and Mapping of Transport Channels onto Physical Channels (FDD),” 3GPP Document no. 3G TS 25.211, Ver. 3.2.0, Mar. 2000.

[40] 3GPP Technical Specification, “Multiplexing and Channel Coding (FDD),” 3GPP Document no. 3G TS 25.212, Ver. 3.2.0, Mar. 2000.

[41] 3GPP Technical Specification, “Physical Layer Procedures (FDD),” 3GPP Document no. 3G TS 25.214, Ver. 3.2.0, Mar. 2000.

[42] 3GPP Technical Specification, “Physical Layer General Description,” 3GPP Document no. 3G TS 25.201, Ver. 3.2.0, Mar. 2000.

[43] N. Parameshwar, R. Rajagopalan, “A comparative study of cdma2000 and W-CDMA.” Wireless Communications and Systems 2000. 1999 Emerging Technologies Symposium, 1999, pp. 15.1 -15.9

[44] V. Garg, *IS-95 and cdma2000: Cellular/PCS System Implementation*, Prentice Hall communications engineering and emerging technologies series, Upper Saddle River, NJ, 2000.

[45] M. Zeng, A. Annamalai, and V. Bhargava, “*Advances in Cellular Wireless Communications*,” IEEE Communications Magazine, Sept. 1999.

[46] A. Samukic, “UMTS Universal Mobile Telecommunications System: development of standards for the third generation,” IEEE Global Telecommunications Conference (GLOBECOM), 1998. The Bridge to Global Integration, Vol. 4, pp. 1976 -1983.

[47] M. Progler, “*Air Interface Access Schemes for Broadband Mobile System,*” IEEE Communication Magazine, Sept. 1999.

[48] A. Gatherer, “*DSP-Based Architectures for Mobile communications: Past, present and future,*” IEEE Communication Magazine, Vol. 38, No. 1, Jan. 2000, pp. 84 -90.

[49] D. Coons, and J. Oba, “*Wideband-CDMA-System Design Become More Complex,*” EDN. Magazine, Sept., 1999. http://www.ednmag.com.

[50] G. Cortez, “3G Wireless Design Verification: What Are The Options?“, Wireless System Design, Feb. 2001. http://www.wsdmag.com.

[51] Altera Corp.,”Implementing a W-CDMA System with Altera Devices & IP Functions,” Sept. 2000. http://www.altera.com.

[52] Texas Instruments “Implementation of a WCDMA Rake Receiver on a TMS320C62x DSP Device,” Application Report, Jul. 2000.

[53] S. Morris, “*Signal Processing Demands Shape 3GBase Stations,*” Wireless Systems Design Magazine, Nov. 1999.

[54] P. Jung, and J. Plechinger, “M-GOLD: a Multi-mode Baseband Platform for Future Mobile Terminals,” IEE International Conference on Communications, CTMC'99, Vancouver, Jun. 1999.

[55] M. Sturgill, and S. Alamouti, “Simulated Design Methodology Targets 3G Wireless Systems,” Wireless System Design, Nov. 1999. http://www.wsdmag.com.

[56] J. Proakis, **Digital Communications**, 4th ed. McGraw Hill, 2001.

[57] E. Nikula, “*Frames Multiple Access for UMTS and IMT-2000,*” IEEE Personal Communication Magazine, Vol. 5, No. 2, April 1998, pp. 16 -24.

[58] U. Fawer, “A Coherent Spread-Spectrum RAKE Receiver With Maximum-Likelihood Frequency Estimation,” Proc. of ICC '92, Chicago, IL, vol. 1, pp. 471-475, Jun. 1992.

[59] L. Harju, M. Kuulusa, and J. Nurmi, “A Flexible Rake Receiver Architecture for WCDMA Mobile Terminals,” IEEE Third Signal Processing Workshop Advances in Wireless Communication, Taiwan, pp. 9 -12, Mar. 2001.

[60] L. Zhenhong, “Tap Selection Scheme in a W-CDMA System Over Multipath Fading Channels,” IEEE International Conference on Communication Technology, Beijing, China, Oct. 1998.

[61] A. Szabo. A. Manuela, and A. Alsolaim, “Performance Simulation of A Rake Receiver for Direct Sequence Spread Spectrum Communication System,” Annual Conference on Semiconductors, Bucharest, Romania, Oct. 2000.

[62] A. Szabo. A. Manuela, and A. Alsolaim, “Direct Sequence Spread Spectrum Communications System In A Multipath Fading Channel.rake Receiver Performance Analysis,” IEEE Proc. Communications, Circuits and Systems, Istanbul. Turky, 2000.

[63] W. Gross, and P. Gulak, “*Simplified MAP Algorithm Suitable for Implementation of Turbo Decoders*,” Electronics Letters, Vol. 34. No. 16. Aug. 1998. pp. 1577-1578.

[64] J. Rabaey, “Beyond the Third Generation of Wireless Communications,” Keynote Presentation, ICICS 99, Singapore, Dec. 1999.

[65] K. Keutzer “*System-Level Design: Orthogonalization of Concerns and Platform-Based design*,” IEEE Trans. on CAD of Integrated System, Dec. 2000.

[66] E. Dahlman “*UMTS/IMT-2000 Based on Wideband CDMA*,” IEEE Communication Magazine, Vol. 36, No. 9, Sept. 1998, pp. 70 -80.

[67] R. Enzler and M. Platzner, “Application-Driven Design of Dynamically Reconfigurable Processors,” Technical Report, Swiss Federal Institute of Technology (ETH) Zurich, Mar. 2001.

[68] P. Chow, et al. "*The Design of SRAM-Based Field-Programmable Gate Array, Part I: Architecture,*" IEEE Trans. on VLSI Systems, pp. 191-197, June 1999.

[69] G. Masera, "*VLSI Architectures for Turbo Codes,*" IEEE Trans. on VLSI Systems, Sept. 1999.

[70] S. Linghao, "*Mobile Terminal Implementation for Third Generation Applications,*" IEEE Communication Magazine, May. 2000.

[71] M. Ding, A. Alsolaim, J. Starzyk, "Designing and Mapping of a Turbo Decoder for 3G Mobile Systems Using Dynamically Reconfigurable Architecture," The 2002 International Conference on Engineering of Reconfigurable Systems and Algorithms ERSA'02, Las Vegas, June 24-27, 2002.

Abstract

Alsolaim, Ahmad Mohammed. Ph.D. Aug. 2002

Electrical Engineering and Computer Science

Dynamically reconfigurable architecture for third generation mobile systems (309 pp.)

Director of Dissertation: Professor Janusz Starzyk

A third generation mobile system is scheduled for launch in 2002. The system provides a high data rate that can be used for multimedia and internet services. As the diversity of required services and performance of the mobile units increases, the traditional hardware implementation of the mobile terminal will fall short of providing the required flexibility and performance. This justifies the use of reconfigurable hardware.

The typical implementations of the current mobile systems are a mixture of ASICs and DSPs. ASICs are used for their high performance and low power. DSPs are used for their flexibility. The two implementation paradigms are combined in this research to reach a compromise between low cost, low power consumption (long battery life), flexibility, and performance.

A dynamically reconfigurable computing architecture is an ideal implementation solution for the third and future generations of mobile systems. This dissertation delineates the design and simulation of a new dynamically reconfigurable architecture called DRAW. DRAW is a hardware fabric specially designed for the third generation wireless mobile systems.

Reconfigurable computing lowers the cost of the final product by shortening the time to market period through the reduction of the design flow steps. Additionally, it reduces the power consumption through the dynamic switching on the required hardware logic, while avoiding the excessive area requirements of dedicated ASICs.

A Matlab simulation test bed was written and used to extract the main characteristics of the target application. Once a broad guidelines of the design process were available, a synthesizeable VHDL description of the architecture was written. The design of the architecture was further optimized through iterative post-synthesize simulations and redesign.

A further work is needed in three main points, firstly, optimize the architecture for power, secondly, develop an automated mapping tools for mapping different baseband algorithms onto the architecture, thirdly, construct a set of intellectual property blocks for wireless communication to be mapped into the designed hardware fabric

Approved:____