TAMPEREEN TEKNILLINEN YLIOPISTO

TAMPERE UNIVERSITY OF TECHNOLOGY
Faculty of Computing and Electrical Engineering

Leyla S. Ghazanfari

DESIGN OF A RECONFIGURABLE MULTI-CORE ARCHITECTURE FOR STREAMING APPLICATIONS

Master of Science Thesis

Subject approved by Faculty Council
March 7th, 2012
Examiners: Prof. Jari Nurmi (TTY)
MSc Roberto Airoldi (TTY)

To my family...

Contents

List of Figures

List of Tables

List of Symbols and Abbreviations

Abbreviation	Description
ADI	Asynchronous Data Interface
AMPS	Advanced Mobile Phone System
ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
CCC	Communication and Configuration Controller
CISC	Complex Instruction Set Computer
COFFEE	A COre For FrEE
CREMA	Coarse-grain REconfigurable array with Mapping Adaptiveness
DMEM	Data MEMory
DSP	Digital Signal Processor
EPI	External Processor Interface
FIFO	First In First Out
FIR	Finite Impuls Response
FLASH	Fast Low-Latency Access with Seamless Handoff
FPGA	Field Programmable Gate Array
GENEPY	homoGENEous Processor arraY
GSM	Global System Mobile
HDL	Hardware Description Language
HW	HardWare
IMEM	Instruction MEMory
I/O	Input/Output
IP	Intellectual Property
IPI	Inter-picoArray Interface
ISA	Instruction Set Architecture
LSB	Least Significant Bit

Abbreviation	Description
LTE	Long Term Evolution
MIMO	Multi-Input Multi-Output
MPSoC	Multi-Processor System on Chip
NI	Network Interfacer
NoC	Network on Chip
OFDM	Orthogonal Frequency Division Multiplexing
PE	Processing Element
RAM	Random-Access Memory
RH	Reconfigurable Hardware
RISC	Reduced Instruction Set Computer
RP	Read Processes
RPE	Reconfigurable Processing Element
RPEC	Reconfigurable Processing Element Controller
RTL	Register Transfer Level
SIMD	Single Instruction Multiple Data
SME	Smart Memory Engine
SMEP	Smart Memory Engine and a Processor cluster with two DSPs
SoC	System on Chip
SRAM	Static Random Access Memory
TUT	Tampere University of Technology
UI	User Interface
VHDL	VHSIC Hardware Description Language
VLIW	Very Large Instruction Word
VLSI	Very Large-Scale Integration

Acknowledgements

I would like to express my deepest gratitude to my supervisor Prof. Jari Nurmi, who made me one of the members of the COFFEE family thanks for opening the door to a professional academic life and guiding me through this project. I would like to thanks Mr.Roberto Airoldi for first being a friend in need and second being a superb instructor, this thesis would not have been completed without his dedicated supports and guidance from the initial to the very final level of the project.

I'm thankful to all my fellow COFFEE group member since many parts of my project was based on their previous research and implementations; and especial thanks to Dr.Fabio Garzia and Mr.Waqar Hussain for helping me in understanding their previously developed designs.

I wish to express my love and gratitude to my mother Behjat who was my ideal in both social and academic life, my father Jalal who was full of encouragement, kindness and always made me believe in myself and my brothers Arash, Hossein and Ahmadreza who no matter the distances I have always felt their love and supports.

I would like to thanks my husband Ali who was always beside me with love and encouragement and also thanks to his family in Espoo and Tehran who brought more joy to my life.

Thanks to my Office mates Deepak and Roberto who made our office a cheerful place to work.

Lastly, I offer my regards and blessings to all who supported me in any respect during the completion of this project.

Leyla S. Ghazanfari

Abstract

Tampere University of Technology
International Master's Degree Programme in Information Technology
Leyla S. Ghazanfari: DESIGN OF A RECONFIGURABLE MULTI-CORE ARCHITECTURE FOR STREAMING APPLICATIONS
MSc Thesis, 57*pages*
March 2012
Major: Digital and Computer Electronics
Examiners: Prof. Jari Nurmi, MSc Roberto Airoldi
Keywords: MPSoC, Streaming applications, Streaming processors, Reconfigurable hardware

This thesis presents design of a reconfigurable multi-processor architecture. The architecture is composed of 9 nodes interconnected to each other through a 3x3 mesh-based Network-on-Chip. The central node of the architecture hosts a RISC processor. This node acts as master of the platform, taking care of the data and task scheduling. The surrounding nodes host a reconfigurable engine and do the actual processing.

The system was prototyped on an Altera FPGA device and RTL simulations of the architecture were carried out to ensure the correct functionality of the system. The platform was designed to process streaming applications. As an example of these applications, a finite impulse response filter was mapped on the system. Simulation results showed a speed-up of 6.8x over the same FIR filter implemented on a COFFEE RISC core, while requiring a 20% less resources of similar architecture composed by a homogeneous mesh of COFFEE RISC cores.

Chapter 1

Introduction

In the past decades Multi-Processor architectures have become an important class of very large scale integration (VLSI) systems and they have been widely utilized for the implementation of complex signal processing systems such as multimedia and wireless communications [20]. On one side there are heterogeneous Multi-Processors Systems-on-Chip (MPSoCs), which are systems composed of different types of processing elements (PE). They have the advantage of offering a high power efficiency and high performance but generally they have limited flexibility. Homogeneous MPSOC architectures (composed of similar PEs) are generally based on soft-cores. Although they allow a really high flexibility but they also have the drawback of limited performance.

In this context the utilization of recofigurable hardware could be exploited in order to achieve high flexibility and high computational power. At the same time the range of application of the platform is expanded to different domains. The goal of recongurable MPSoCs is to adjust the application to the multiprocessor architecture automatically [35].

The purpose of this thesis is to present a reconfigurable architecture as a processing engine for streaming applications. The MPSOC structure will increase the ability to process parallel tasks and the reconfigurability makes the architecture more suitable for streaming applications. The architecture that is going to be presented has almost a homogeneous structure but each of the reconfigurable nodes can function heterogeneously. This architecture can be used for multimedia applications in general or for instance applications that need to process vectors. This architecture is implemented in VHDL but receives its inputs in C code which made the UI much more user friendly. Many of the components that have been used in this work have been previously developed and tested here at TUT. This made the developing and testing process faster and more precise.

The thesis is organized as follows: In Chapter 1, we talk about the streaming

applications. Their main characteristic and applications domain is presented. In addition the reader will be familiarized with the concept of streaming processor a state of the art Imagine processor which is also a streaming processor is introduced; In Chapter 2 we speak about MPSoCs (Multi-Processor System-on-Chip) which are aimed for increasing the performance. The heterogeneous and homogeneous structures of MPSoCs are defined and the effect of having reconfigurable hardware in MPSoC architectures is realized. In this chapter the reader will also be familiarized with some state of the art MPSoC designs; In Chapter 3 the details of the new design of the reconfigurable MPSoC architecture for streaming applications is presented. After reviewing the overall structure of the system, the architecture is divided into three parts of central controller, reconfigurable node and the Network-on-Chip (NoC) interconnection and each part is explained separately. At the end of this chapter the compilation process for the simulator and the synthesis results are delivered; And finally in Chapter 4 we present an FIR filter application which is mapped to our architecture.

Chapter 2

Streaming Applications

Streaming applications have become increasingly important and widespread. There have been proofs that streaming media applications are already consuming most of the cycles on consumer machines and their use is continuing to grow [29].

In this chapter we define the properties of streaming applications then we introduce their domains with examples from research and industry. In the third section we present the state of the art streaming processor.

2.1 Characteristics of a Streaming application

In order to declare streaming applications we start by describing its characteristics. In this section we point out the properties that are common in steaming applications. They can be organized as follows [33]:

Large streams of data

The initial and most fundamental characteristic of streaming applications is that it operates on large amount of data. In these applications this data is referred to as stream data. Stream data enters the operation section from some external source. Each item of data is processed for limited amount of time and then it is discarded from the operation section. In streaming applications there is little data reuse.

Independent stream filters

In streaming computation, sequence of transformations are applied to the data stream. Every basic unit of transformation is referred to as a filter. In streaming application each execution step has a filter. These filters read one or more data

from input stream, then perform some operations over them and finally write one or more of the data results to the output stream. These filters are generally independent from each other but a chain of them makes the path of data through the system, meaning that output data of some filters are connected to the input data of other filters.

Occasional modification of stream structure

Although filters are connected to each other in an almost fixed chain sometimes the order of this chain can change. This means that when operations are done on stream data they might enter different set of filters. For example, if there is high noise on an input channel in a wireless network, it might reacted by adding some filters to clean up the signal. In some cases these re-initializations are synchronized with some data in the stream. An example of this situation is when a network protocol changes from Bluetooth to 802.11 at a certain point of a transmission.

Occasional out-of-stream communication

In addition to large-volume stream of data that is passed from filter to filter in streaming applications, it can also happen that filters are used to transfer occasional control signals. These control signals can happen out-of-stream and on random situations. They are also much less than stream data. Examples of these occasions can be changing the volume on the cell phone, error message printing on a screen or for instance updating a coefficient in an upstream FIR filter.

High performance expectations

Often streaming applications must be able to satisfy real-time constraints. So efficient executions in terms of both latency and throughput is very important. In addition to these concerns there are many embedded applications for instance mobile environment applications where power consumption, memory requirements, and code size are also important.

2.2 Streaming Applications Domain

Applications which make use of stream have diverse targets they can be used in:

- Embedded devices
- Consumer desktops
- High performance servers

Examples of such applications are Click modular router [24], the Spectrumware software radio [32, 12], specifications like the Bluetooth communications protocol [33], the GSM (Global System Mobile) Vocoder [27], and the AMPS (Advanced Mobile Phone System) cellular base station [2]. Almost any application developed with Microsoft's DirectShow library [9], Real Network's RealSDKSoftwareDevelopersKit2001 or Lincoln Labs Polymorphous Computing Architecture [8] are also streaming applications.

2.3 Streaming Processors

Stream processors are high performance processors that are optimized to run streaming applications. These processors are more efficient in area and energy saving in comparison with the conventional programmable architectures. The growing gap between arithmetic performance and bandwidth cause the gap between general purpose and special purpose processors to grow. General purpose processors are limited by bandwidth and so they only use a small portion of their die area for arithmetic unit. Thus they have relatively a performance which is low peak. On the other hand special purpose processors, reduce their demands of bandwidth by exploiting locality in their fixed applications. Graphic chips are examples of such special purpose processors. Hence these processors can efficiently exploit hundreds of arithmetic units. This means that they dedicate a much larger portion of their die area to arithmetic than general purpose processors. The gap between the special purpose solutions which are inflexible and general purpose processors which does not meet the computation and arithmetic demands in streaming applications is bridged by stream processing. Stream processing offers high performance without sacrificing programmability. Stream processing exploits locality and concurrency in the application by partitioning the communication and storage structure in order to reduce its demands for bandwidth similar to special purpose processors. Hence it can also support many ALUs efficiently. [21], [23]

2.3.1 State of the Art Streaming Processor

In this section we can study about a state of a art project that uses stream-based computation at the application, compiler, and architecture level.

Imagine

Imagine project was designed and prototyped at Stanford University. This architecture can execute stream-based programs. It has high performance with 48 floating-point units. The steam data is load and stored from the memory through a streaming memory system. This architecture can provide large amount

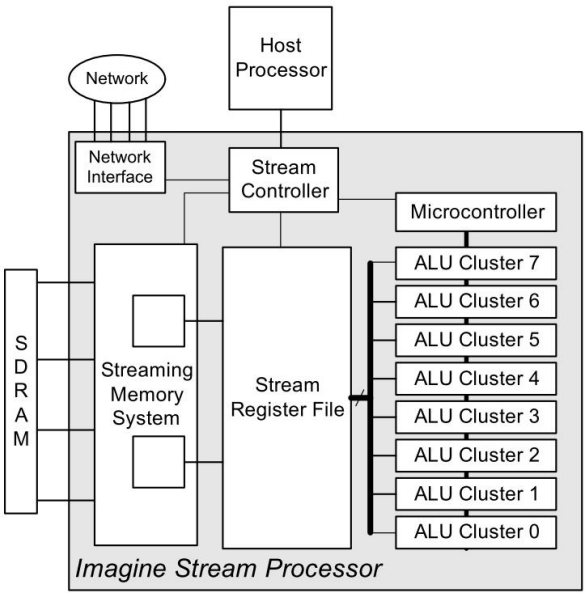

Figure 2.1: The Imagine architecture block diagram [22]

of on-chip intermediate storage using its stream register file. During kernal execution eight VLIW (Very Large Instruction Word) arithmetic clusters perform SIMD (Single Instruction Multiple Data) operations on data streams. A micro controller is used to sequence the data. In order to support the multi-Imagine systems and the I/O transfers, a network interface is used in the architecture. Finally the operation of all these units is managed by a stream controller. The applications that are programed on Imagine architecture use stream programming model. This model is composed of data streams and kernels. Kernels are small programs that use set of data stream as their inputs, operate on them and produce set of output data streams. Set of software tools and languages are used to implement the stream programming model. The programming languages are called StreamC and KernelC. StreamC is mapped to stream instructions using stream scheduler and KernalC is mapped to VLIW kernels using kernel scheduler. Imagine has high performance and is programmable and real-time. Thus applications such as programmable graphics pipelines have found their research path through this project. A prototype of this architecture was design and fabricated in 2002. Imagine contains 21 million transistors and has a die size of 16mm x 16mm in a 0.15 micron standard cell technology [3].

Chapter 3

Trends in MPSOC and Reconfigurable Processors

After studying about streaming applications in chapter one, we realized the importance for high performance hardware that can manipulate the large amount of data stream. Multi-Processors System-on-Chips (MPSoC) are made to increase the performance and so they can be a very good choice for architectures that are aimed for streaming applications. That is why in this chapter we introduce MPSoCs. They are categorized and defined in to two different types of homogeneous and heterogeneous architectures. After defining these terminologies we also present examples of their state of the art architectures. In order to increase the performance of our system, reconfigurability can be added to it and so role of reconfigurable hardware in MPSoC is also explained. By the end of this chapter the reader will be introduced with the above mentioned concepts and their recent trends in research and technology.

3.1 MPSoC

MPSoCs are known as an important class in VLSI systems since the last decade. MPSoCs are composed of multiple components that are amid to process an application. All or most of the components needed for an application is encapsulated inside this VLSI system. Networking, signal processing, multimedia and communications are examples of different areas where MPSoCs are widely used [20]. MPSoC architecture consists of several Processing Elements (PE) connected by an interconnection structure. An example of the hardware organization of an MPSoC can be seen in figure 3.1.

MPSoC can be divided into two families: Homogeneous and Heterogeneous.

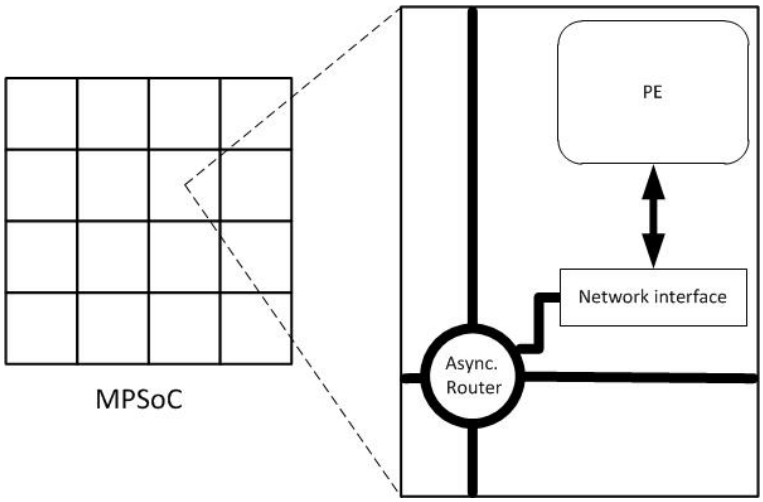

Figure 3.1: General MPSoC Architecture

Each of them are explained in the following sections.

3.2 Heterogeneous MPSoC

Heterogeneous systems structure is composed of different types of PEs. They are a set of cores that are connected to each other and can have different functionality. For instance a generic heterogeneous MPSoC can consist of one or more general purpose processors, accelerators which can be for (video, audio,..), DSPs, peripherals and memory. And these different blocks are connected to each other through a NoC (Network-on-Chip) interconnection infrastructure. Heterogeneous MPSoCs are strong in having specialized hardware for different tasks but their programming environment are usually more complex [18], [31].

3.3 Homogeneous MPSoC

In a homogeneous MPSoC the same programmable building block is instantiated a number of times. This model of architecture is also called parallel model. The basic concept of homogeneous models is that the capability should increase because the number of physical resources has increased. So compared to non-parallel models the execution time for each resource should divide to the number of PEs. For instance if now we have N, PEs the application should be N times faster. However this exact speedup is almost impossible to achieve in practice. Another benefit of parallel models is that run time frequency decreases and so dynamic power consumption also decreases significantly. Homogeneous MPSoCs have an easier programming environment but they also lack in having specialized

hardware for different tasks. These MPSoCs can also better provide flexibility, fault tolerance and scalability [18], [31], [19].

3.4 State of the Art in MPSoCs

In this section we introduce the reader with examples of state of the art heterogeneous MPSoCs.

3.4.1 Picochip

PicoArray is designed for signal processing applications. Figure 3.2 shows an image of the Pico Array below the concept of the architecture and communications methods of this device is describe.

The picoArray Architecture

PicoArray has a parallel, MIMD (Multiple Instruction Multiple Data) architecture. In picoArray different types of PEs are connected to each other through picoBus. The 16-bit Harvard architecture processors are the basic building block of the array architecture. Each building block has its own three-way LIW (Long Instruction Word) and a local memory. To speedup some specific tasks FAUs (Functional Acceleration Units) and hardwires execution blocks are also added to this structure. A complete ARM9 processor subsystem can also be added to this device. [7]

Inter-processor Communications

Inside the picoArray each device is organized in a two-dimensional grid, its communication is done via a network of 32-bits buses and a programmable bus switches. Ports connect the array elements together. These ports act as nodes on the picoBus and provide an interface to the bus depending on put and get instructions. The inter processor communication protocol is based on TDM (Time Division Multiplexing). In this protocol, data transfers between processor ports occur during time slots scheduled automatically by the tools and controlled using the bus switches [7], [15].

External Communications

Pico Array has three methods of external communications:

- External Processor Interface (EPI)
- Inter-picoArray Interface (IPI)
- Asynchronous Data Interface (ADI)

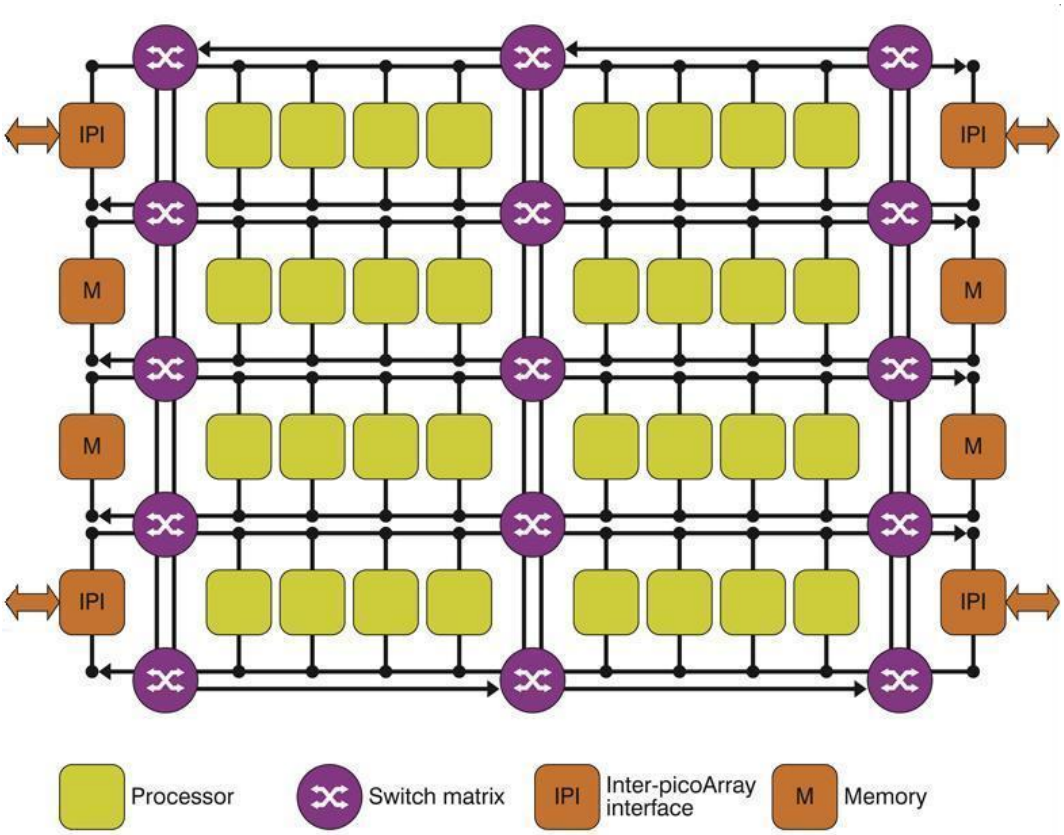

Figure 3.2: The PicoArray concept [5]

The EPI can be used to configure picoArray devices and it can also be used by debugging tools for input and output of (2.5 Giga-bits per second) information. The IPI is used to connect picoArray devices together. Using IPI multiple picoArray devices can be connected to each other and form a system that contains thousands of processos. The ADI is used for exchanging data with high bandwidth (5 Giga-bits per second) external asynchronous data streams. Each device has a one EPI and four other interfaces each of these interfaces can be configured as either an IPI or an ADI [15].

3.4.2 MAGALI

MAGALI is a heterogeneous Telecom chip. It supports OFDMA/MIMO TX/RX baseband algorithms. This chip is composed of different IPs (Intellectual Property) which can communicate with each other through a mesh-based NoC. An image of MAGALI structure is shown in figure 3.3. These IPs are not identical but each type can be available several times and used when they are needed by the application. All the resources are associated to a Communication and Configuration Controller (CCC) and a Network Interface (NI). The CCC and NI

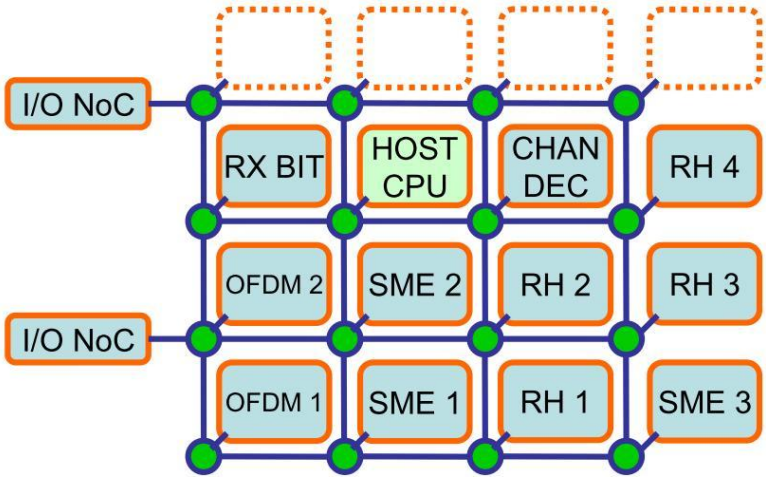

Figure 3.3: MAGALI Telecom chip resources [13]

control and configure the resources through the system's NoC. As it is shown in the figure there are different resources inside the MAGALI chip. The RH (Reconfigurable Hardware) resources are coarse-grain reconfigurable cores that are used for complex matrix computations. The SMEs (Smart Memory Engine [26]) are reconfigurable memories resources which include data re-arrangement functions trough virtual buffers and configuration servers. The RH and SME type have no predetermined position in telecommunication chain. The other resources are OFDM, RX_BIT (bit operations) and CHAIN_DEC (channel decoding). These three type are specific reconfigurable IP cores and directly support functions of the chain for receiving data [13].

3.5 Reconfigurable HW in MPSoC

The goal of Reconfigurable MPSoC is to adjust the application to the multi-processor architecture automatically [14]. Hardware demands can differ from on application to another. These demands can also change at runtime. The runtime property changes happen because the application reacts to environment requests. For this reason, new degree of freedom in system designs and runtime supports is made. In programming flexible multi-processor systems, it is very important that the complexity of the underlying hardware is hidden. Furthermore there is a need of runtime operating system to take care of the resource management and runtime scheduling of the applications. Nowadays FPGAs are used in a wide area of applications. Previously, FPGAs were used as rapid prototyping systems to integrate test systems. After the testing was done ASICs were used to make the mass production of those prototypes. But recently things have changed. FPGA prices are decreasing while the mask-costs of ASICs are still high. In addition

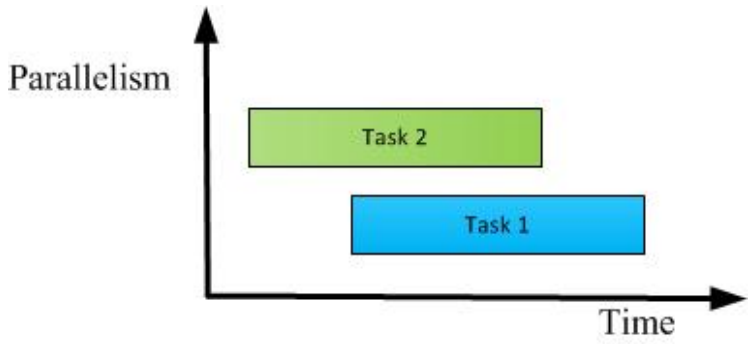

Figure 3.4: Parallel task processing

novel reconfigurable hardware have lower power-consumption then before. These properties in addition to the flexibility have opened a market in industry and a wide research area in scientific researches. Especially the ability to have runtime reconfiguration which is supported by some state of the art FPGAs is a door to new ideas for adaptive hardware. As long as FPGA architecture is SRAM or FLASH based it can be reconfigured many times. Xilinx Virtex FPGA is one of modern state-of-the-art FPGA devices which support a partial dynamic runtime reconfiguration. This provides new aspects for designers who want to develop applications which need reconfigurable and flexible hardware.

Figure 3.4 shows an example of running tasks which are executed in parallel. The attributes shown in this diagram does not make difference between a hardware reconfigurable system and microprocessor based system approach.

Another dimension is added in runtime reconfigurable systems. In the figure 3.5 axes X and Y represent chip area dimensions. For instance here we can see that task 4 needs a lager amount of chip area compared to task 3 also task 1 needs more chip area than task 2. Figure 3.6 shows a cut through XY axes at t1 and t2 of Fig 3.5. Here we can see more clearly which tasks need more chip area. So it is realized that chip area can be reused for different tasks processes that begin and finish at different points of time. These reusable chip areas can be reconfigurable hardware that is used applications which their data processing is on-demand. It means that the processing of data is initiated either by external triggers or internal requirements. This way of approach has a benefit of reducing chip size because only currently required tasks are using the HW and the idle functions can be loaded from external memory when needed. [18]

3.6 State of the Art in MPSoCs with Reconfigurable HW

In this section GENEPY (homoGENEous Processor arraY) architecture is presented. GENEPY is state of the are for homogeneous MPSoCs with reconfig-

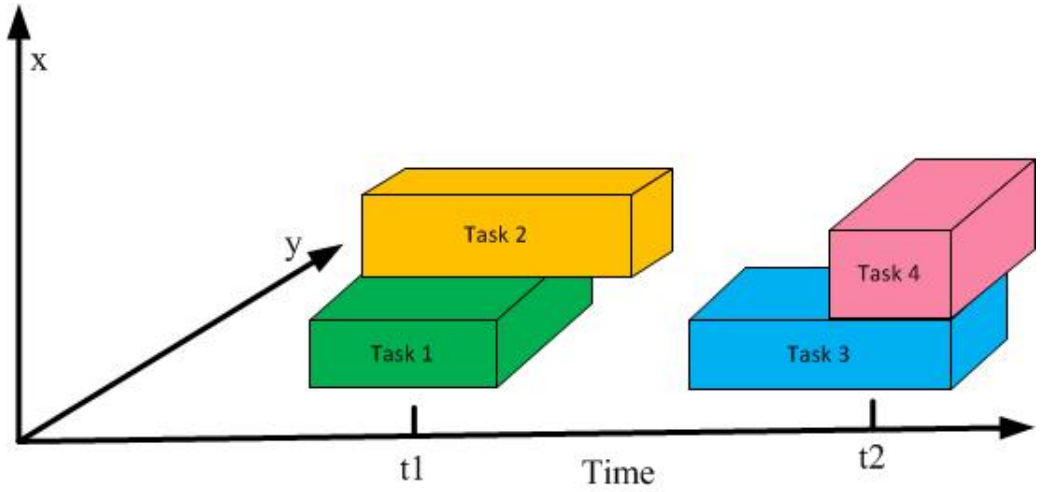

Figure 3.5: Parallel task processing in time and space

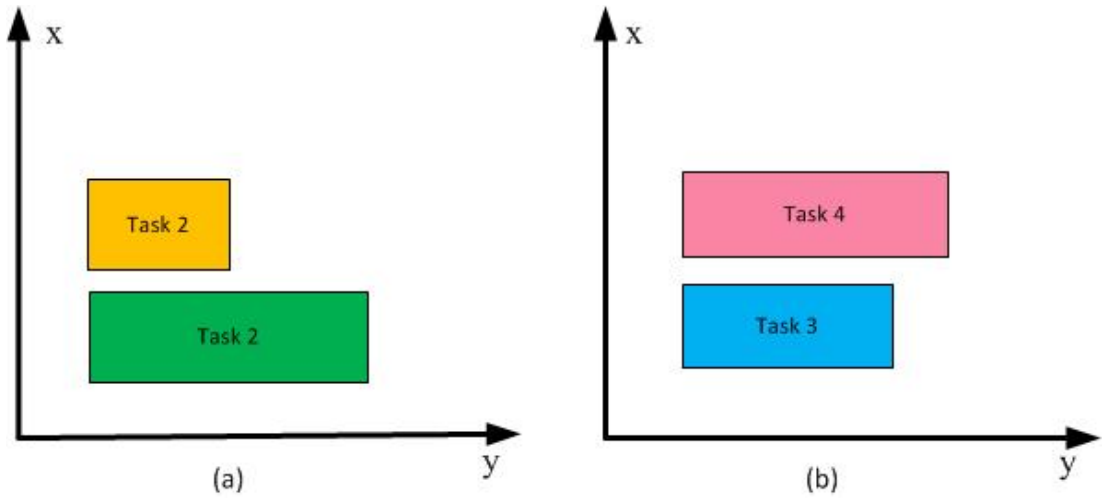

Figure 3.6: Cut parallel to the XY-plane at time point t1 and t2

urable hardware.

3.6.1 GENEPY

GENEPY [19] is a platform for 4G applications. This platform is built with Smart ModEm Processors (SMEP) and it is interconnected with a NoC. The SMEP used in this platform has been implemented in 65nm low-power CMOS and it can perform 3.2 GMAC/s with 77 GBits/s internal bandwidth at 400MHz. Two versions of GENEPY are presented in this sub-section.

GENEPY v0

GENEPY v0 is a homogeneous processor array with a host processor. SMEP v0 is a Smart Memory Engine (SME) and a Processing unit with two DSPs. Figure 3.7 shows the structure of this unit. This unit is used in GENEPY v0 and is used to support both data manipulation and data processing. The SME handles four logical buffers that are mapped on the same 32KB local memory (RAM data). Four Read Processors (RP) are used to manipulate data on the four buffers. A RP is in charge of executing microinstructions to read data from a buffer. As a result of this action it generates read addresses, writes data to a specific target and it is also responsible for handling synchronizations between RPs. The write target of RPs can be the Network Interface to access other units or another buffer in the local memory. The inner communications of the SMEPs are supported by a 6x6 crossbar. This interconnect can support six parallel 32-bits transfers at 400MHz. There are two DSPs inside the processing cluster and each of them reads incoming data from an input FIFO, the local memory stores the intermediate processing values and the results are written into an output FIFO. The read process in the SME reads the output data-flow from the output FIFO. This SMEP unit of the processor array is highly programmable. The computing is DSP based. Figure 3.7 also shows the CCC block. This block has the same functionality as the one that was described in MAGALI and is used to perform the reconfiguration and the scheduling of the system. GENEPY v0 has a host processor and N SMEP units which are interconnected by an asynchronous NoC (see figure 3.8). The main advantage of this solution is a reduced NoC size (number of routers) for the same computing power compared to the MAGALI structure.

GENEPY v1

GENEPY v1 has a homogeneous platform with a single units type which has been instantiated several times. Each of these units is responsible for processing, configuration and scheduling. GENEPY v1 has the same processing and data management blocks as GENPY v0. The new architecture that is used in this

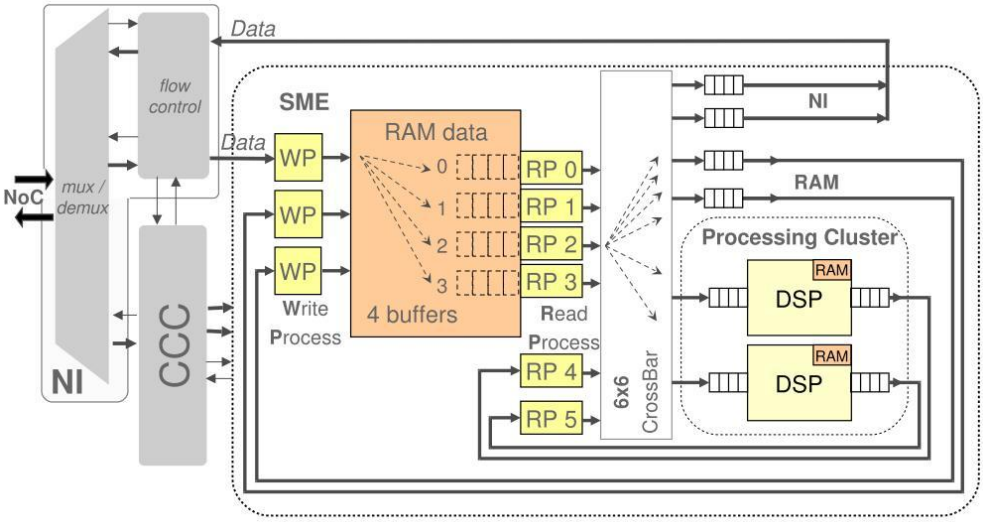

Figure 3.7: Elementary unit (SMEP v0) of the homogeneous processor array with host processor [19]

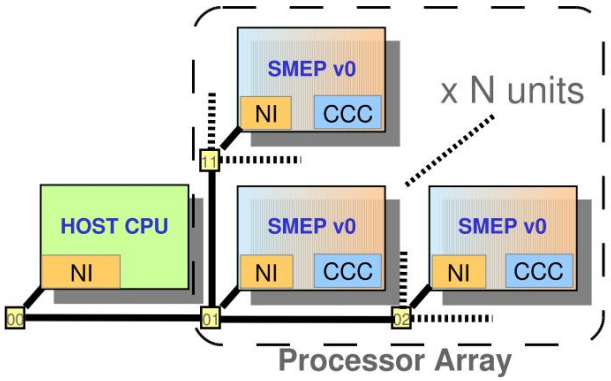

Figure 3.8: GENEPY v0 platform [19]

system is SMEP v1. Figure 3.9 shows the structure of this unit. Using a Control Processor instead of the previous CCC, the system can be fully distributed control. This control processor is a MIPS processor which is in charge of dynamic reconfigurations, real-time scheduling and synchronizations. Several extensions are used to improve the efficiency of the CPU. They are:

- Input/Output extension to manage a control flow between units
- Timer extension to handle real-time constraints
- Configuration handler to improve reconguration speed

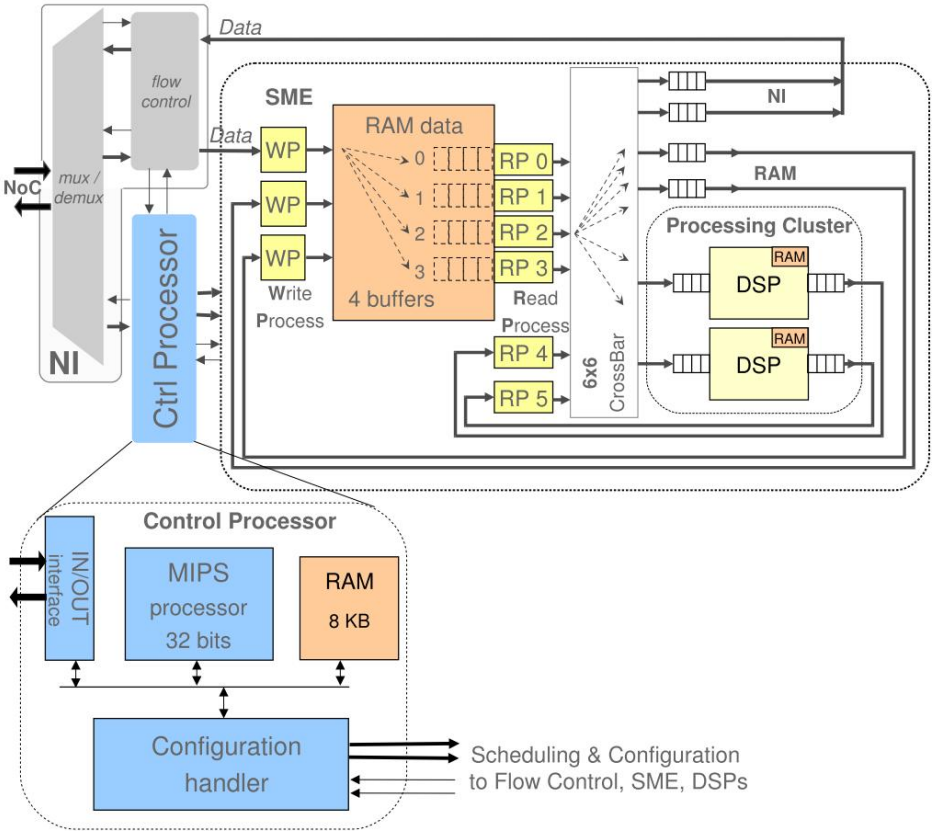

Figure 3.9: Elementary unit (SMEP v1) of the fully homogeneous processor array [19]

The NI, the SME and the processing cluster is managed by the control processor. One difference between CCC and the control processor is that the management of the control processor is software-based which causes increase in the flexibility and the autonomy of each unit and there will be no need for a host processor. Figure 3.10 shows the fully homogeneous processor array which was described here. GNEPY v1 platform is a NoC interconnection of SMEP v1 units. At this level, the platform is homogeneous and fully distributed. To increase the computing capacity of GENEPY v1, SMEP unit can be replicated as needed.

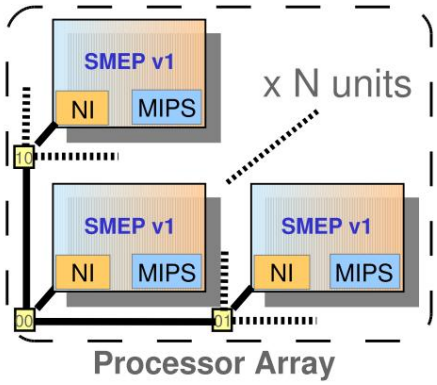

Figure 3.10: Fully homogeneous processor array: GENEPY v1 platform [19]

Chapter 4

Architecture Proposal and Synthesis Procedure

In the previous chapter the reader was introduced with the concept of MPSoC and reconfigurable hardwares. In this chapter we present a detailed picture of the proposed architecture, its components and discuss about the procedure that should be taken in order to synthesis our hardware. We will also present figures to show the hardware usages of our design.

4.1 System overview

The proposed architecture is composed of 9 nodes that are connected to each other in a 3x3 mesh topology. This architecture is based on Silicon Café [28]. Figure 4.1 is an overall view of the system. The central node is system's controller (COFFEE [25]) which is surrounded by eight other nodes that are reconfigurable. The communication between these nodes is supported by a Network-on-Chip (NoC) platform [11]. In the following sub-sections we introduce these parts in more detail.

4.1.1 COFFEE

COFFEE is a general-purpose 32-bit RISC (Reduced Instruction Set Computer) processor. This processor was developed at TUT. This core was designed to be used in embedded systems and it was aimed for applications related to telecommunication and multimedia and its instruction set has 66 instructions [30]. The external interface of this core is illustrated in figure 4.2. Having an overall view of what are the main characteristics of the COFFEE core, now we are ready to explain its role in our architecture.

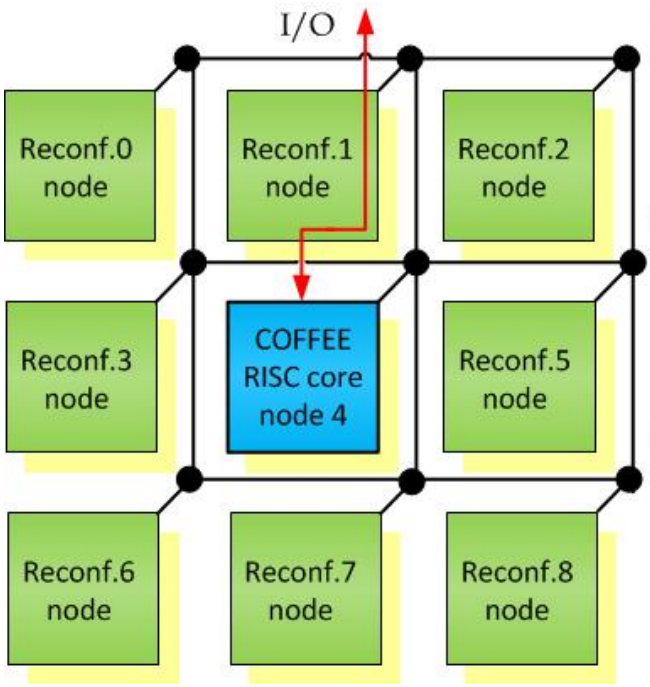

Figure 4.1: System overview

The central node acts as the system's controller. We have chosen COFFEE (a Core For FrEE [25]) RISC core to be the central node. This node is the only node that is directly connected to the I/O ports. The C code which is the input of the COFFEE core determines the data and configuration words that should be passed from the controller node to the corresponding reconfigurable node through the NoC.

4.1.2 Network

In this architecture, the interface between the central controller and the reconfigurable nodes is a NoC-Based platform which has been also developed at TUT. Figure 4.3 shows a global view of this network.

This platform has a hierarchically heterogeneous architecture which can increase bandwidth inside processing clusters by using its local switches that replace shared buses. The NoC platform has priority-based low-latency arbitration logic with a memory space conserving programming model. In short we can say that our NoC can efficiently utilize the communication resources through the bus oriented standard [11].

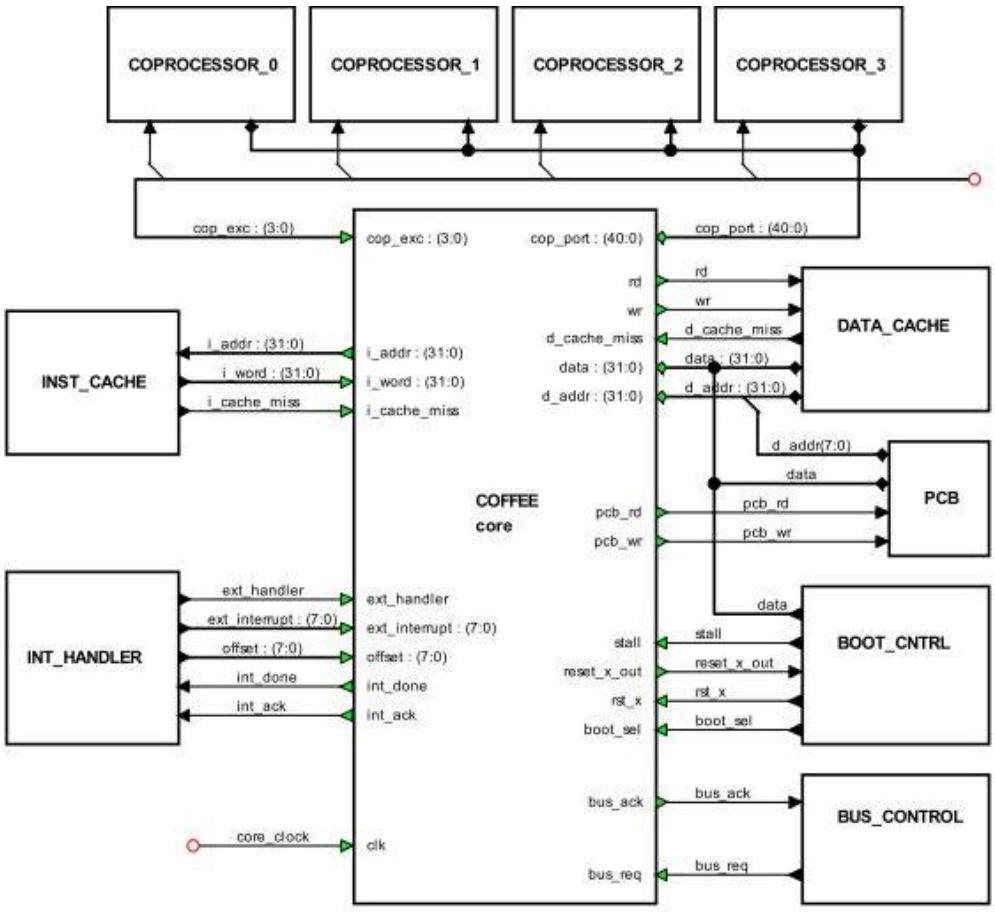

Figure 4.2: Interface of COFFEE Core [1]

Figure 4.3: Global view of the NOC-based platform [11]

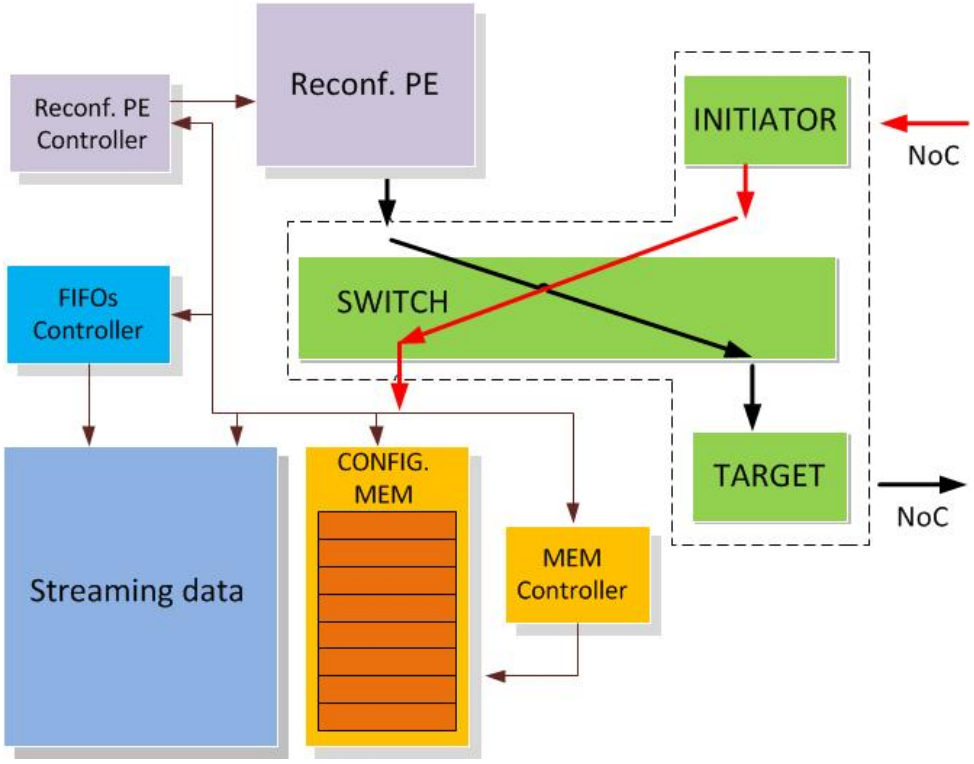

Figure 4.4: Reconfigurable Node Structure

4.1.3 Reconfigurable Node

Previously we discussed the overall view of our multi-core system, the central core and the network was explained. Now we are ready to take a deeper look to the structure of the surroundings reconfigurable nodes. Figure 4.4 shows an abstract view of the components and connections that are inside of each reconfigurable node. Each reconfigurable node can be divided in to five main sections that are:

- NoC interface
- Streaming data
- Processing element
- Configuration memory
- Controllers

Next, these parts will be explained with more details.

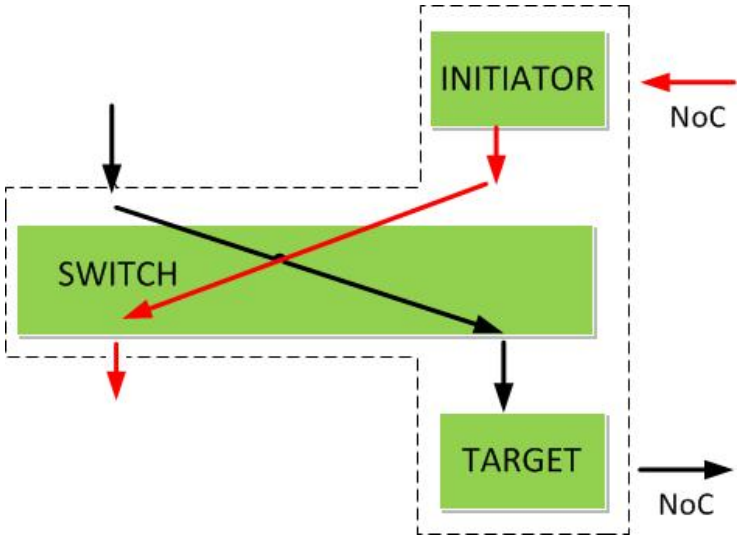

Figure 4.5: NOC interface structure

NoC Interface

In figure 4.5 the NoC interface is shown. The initiator, switch and target blocks, are the main components of our network interface. These components have been previously developed here at TUT. The design aim of this switch is to replace shared buses to increase bandwidth in the local clusters of large systems-on-chip [10]. In the figure we can see two sets of arrows. The black arrows are the routes for receiving the data from the NoC into the initiator, from the initiator to the switch and from there to the other components. The red arrows show the routes that data is sent to the NoC. After the processed data is ready it is sent to the switch and from there to the target component. With every data there is an address, using this address the data will be sent to the predefined core in the network.

Streaming Data Component

Streaming data component is composed of four FIFOs (First In First Out). The role of this component is to receive the data from the switch and keep them in the FIFOs until the processing time. In figure 4.6 an abstract view of this component is shown.

The FIFOs that are used in the streaming data component was previously implemented as a gate-level net-list in our department. One of the important features of this structure is that it is an asynchronous FIFO, meaning that it can have different frequency clock cycles for writing and reading [34]. This feature will be used in our implementation. When data is received by the reconfigurable core, it is sent to the streaming one by one on every clock. This clock frequency is

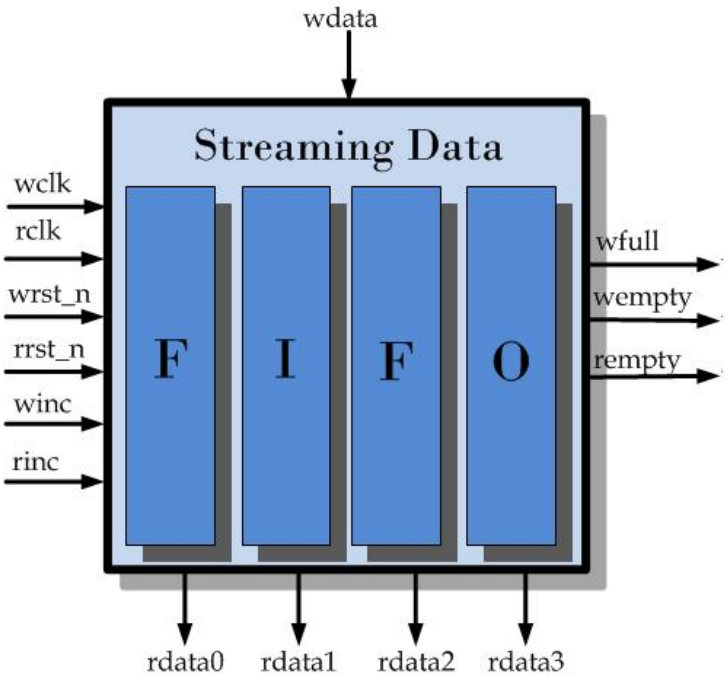

Figure 4.6: Streaming data component

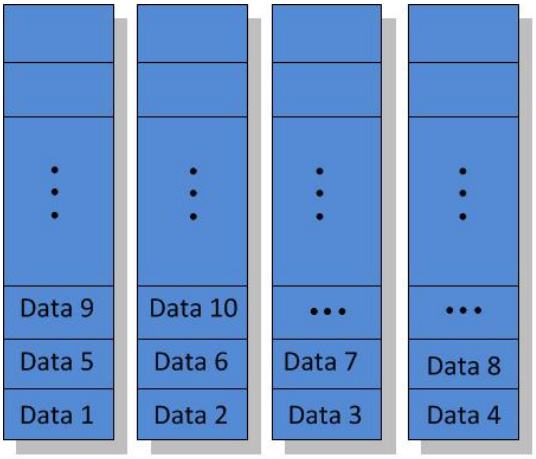

Figure 4.7: The four FIFOs are filled row by row

the same frequency which the total system and central core is working with. The entering data will fill the FIFOs row by row. The order with witch the FIFOs are filled can be observed in figure 4.7.

After all the four FIFOs are filled with data, the reading procedure can start to stream data to the processing section. Reading the data from the FIFOs will be done four times slower than writing the data to the streaming data component. Reading the data will be four times slower but instead we will read four data on

every cycle.

Reconfigurable Processing Element

The Reconfigurable Processing Element (RPE) is composed of four reconfigurable cells attached to multiplexers, data and control signals. The structure of the RPE is shown in figure 4.8. RPE speed of processing is four times slower than the system clock cycle. This means that RPE works at the same pace as the streaming data read procedure does.

Each reconfigurable cell is one CREMA (Coarse grain REconfigurable array with Mapping Adaptiveness [16], [17]) cell. All the CREMA cells can be configured to do different operations such as addition, subtraction, multiplication, etc. depending on the value of their Configuration signal (CONF). The select signals (sel) choose the inputs operands for each reconfigurable cell. By looking at figure 4.8 we can see that RPE has four inputs and four outputs. The four inputs are the four outputs of the streaming data component which was described previously.

The outputs of the RPE should be sent back to the network. So first they should be sent to the NoC interface. At every cycle, four data is generated as RPE output but the switch can transfer one data at a time. This problem is solved by our multi clocks which have different frequencies. The output data is sent to the switch four times faster than it is generated as RPE output.

In figure 4.9 we can see that RPE outputs will first enter a four-input multiplexer. With every RPE clock cycle four different values can be sent to the switch via this multiplexer.

By looking at figure 4.10 the RPE clock process can be described easier. We can see that on every RPE clock cycle four data (for example: D0,D1,D2 and D3) will enter the RPE. Since there are two levels of reconfigurable cells the final results (meaning: R0,R1,R2 and R3) will be ready after two RPE clock cycles delay. These results can enter the switch from our four-input multiplexer that was explained from figure 4.9 on four consecutive system clock cycles.

Configuration Memory

The fourth important section of the Reconfigurable cell is the configuration memory. An image of this memory is shown in figure 4.11. The RPE configuration signals are stored in the first four space of the memory. In the fifth space an address is stored, which refers to the address of the core that the RPE outputs are sent to. In the sixth space the REP select signals are stored. Each select signal is connected to a two to one multiplexer meaning that each of the four select signals is one bit. They can be seen in figure 4.8. These four bits are stored in the LSBs (Least Significant Bits) of the sixth word in the memory. The order of the select signals has been specified in figure 4.12.

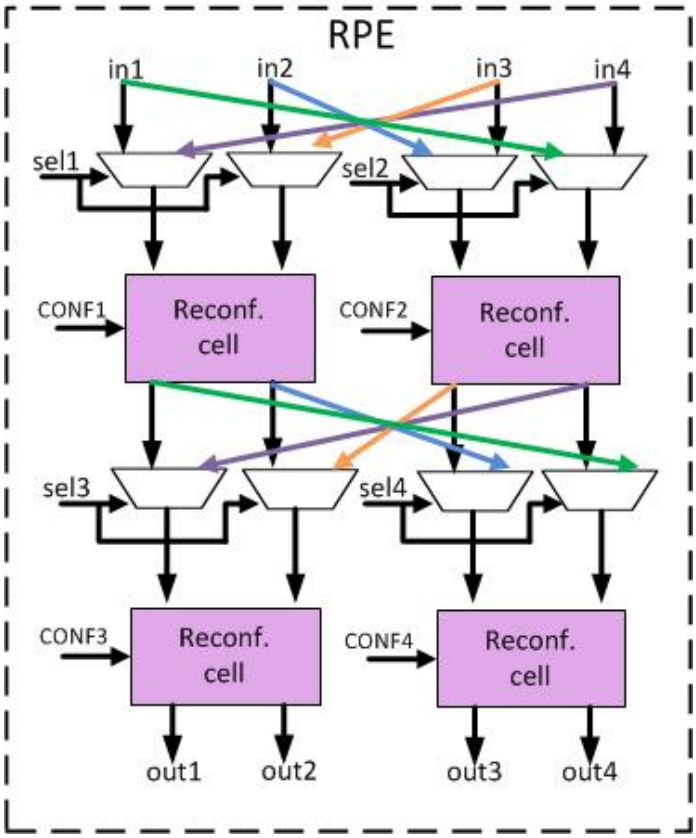

Figure 4.8: Reconfigurable Processing Element

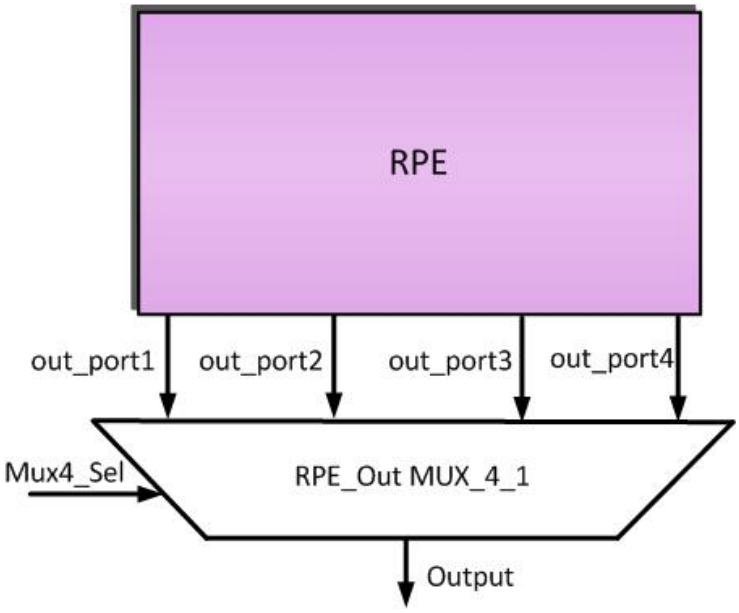

Figure 4.9: RPE output selection

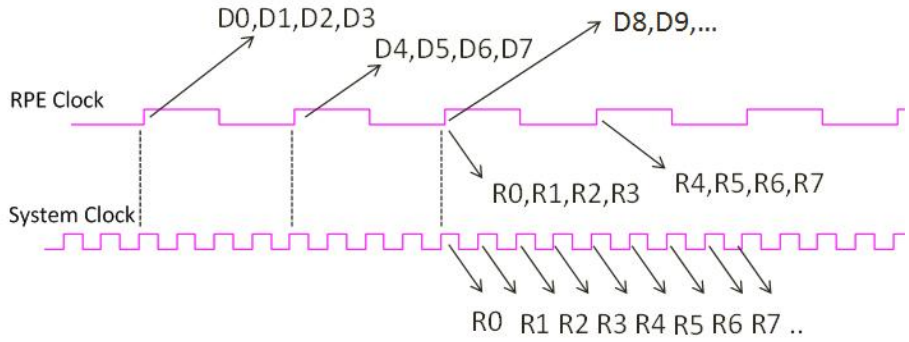

Figure 4.10: Clocking system

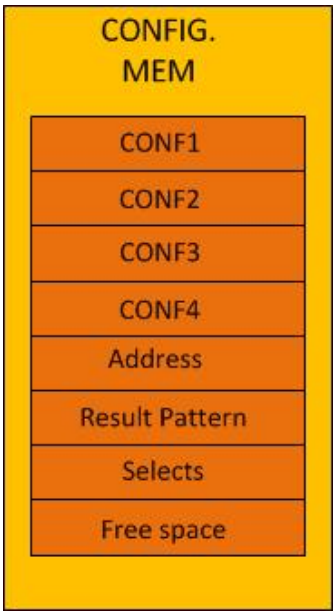

Figure 4.11: Reconfigurable Processing Element

In order to increase the reconfigurability of the architecture the seventh space of the configuration memory is where the result pattern is stored. The result pattern word specifies the order in which our four RPE outputs are sent to the network. The Result pattern is sent to the reconfigurable processing element controller. Depending on the result pattern value the RPE controller assigns different values to Mux4_Sel signal of the RPE output multiplexer. The pattern for different values of the result pattern is specified in table 4.1.

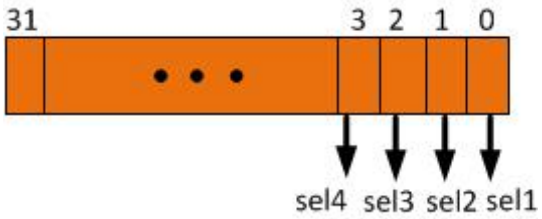

Figure 4.12: Select signals order in the configuration memory.

Table 4.1: Pattern specifications

Result Pattern	Order of Results entering the Network in four consecutive clock cycles
X 00000000	R0, R1, R2 and R3
X 00000001	R0, R2, R0 and R2
X 00000002	R1, R2, R1 and R2
X 00000003	R0, R0, R0 and R0
X 00000004	R1, R1, R1 and R1
X 00000005	R2, R2, R2 and R2
X 00000006	R1, R3, R0 and R2

4.1.4 Controllers

In each reconfigurable cell there are three controllers. They are as follow:

- Memory Controller
- FIFOs Controller
- Reconfigurable Processing Element Controller (RPEC)

An abstract view of them were shown if figure 4.4. In the following parts we will look at them with more details.

Memory Controller

Figure 4.13 shows the logic of the memory controller. Our memory controller is a simple 3 to 8 decoder which can be activated with an enable (en) signal. As it is shown in the figure Imem-request-link is a bus link that is received from the network interface (NOC_UI). This bus is 68 bits. The 32-bits configuration data, 32-bits address of the configuration data and 1-bit write enable (wren) control signal are also in this link. The fourth to second bits of the address is the input of the decoder. The first two bits of the address are left out because every data is 32-bits meaning that its four bytes so the first two address for every new data will be "00". When the Imem-link-wren signal is active it means a data should

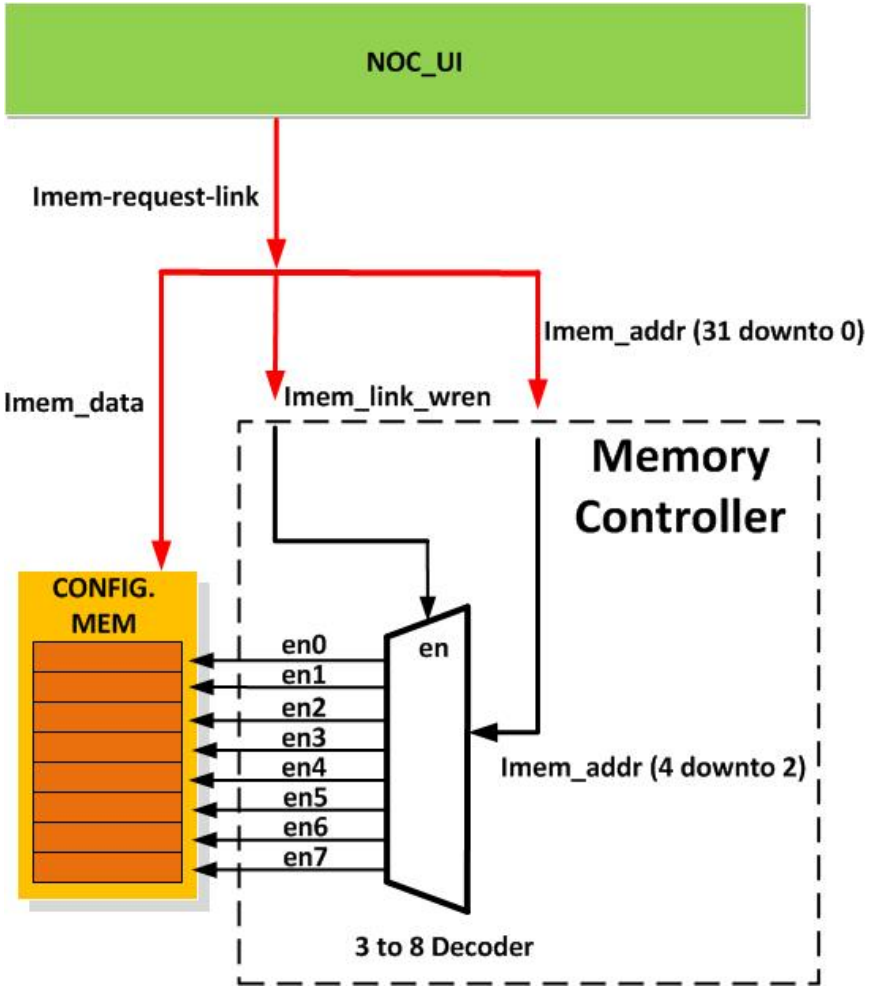

Figure 4.13: Memory controller

enter our configuration memory block. Using the address bits we can select the memory block to which the data will locate.

FIFOs Controller

FIFOs controller primer task is to assign appropriate values to the streaming data component control signals. Figure 4.14 shows the signals connected to this controller. The inputs are received from NoC_UI and streaming data components and the output signals are sent to streaming data and RPEC component which will be explain after this part. FIFOs controller main structure is a state machine.

As it can be seen in the figure dmem-request-link is a bus that contains 32-bits data write enable signal (dmem-link-wren) and 32-bits address of the data. The data is sent to the Streaming data component. When the write enable (en) signal is activated it means that the data should be written in one of the streaming data

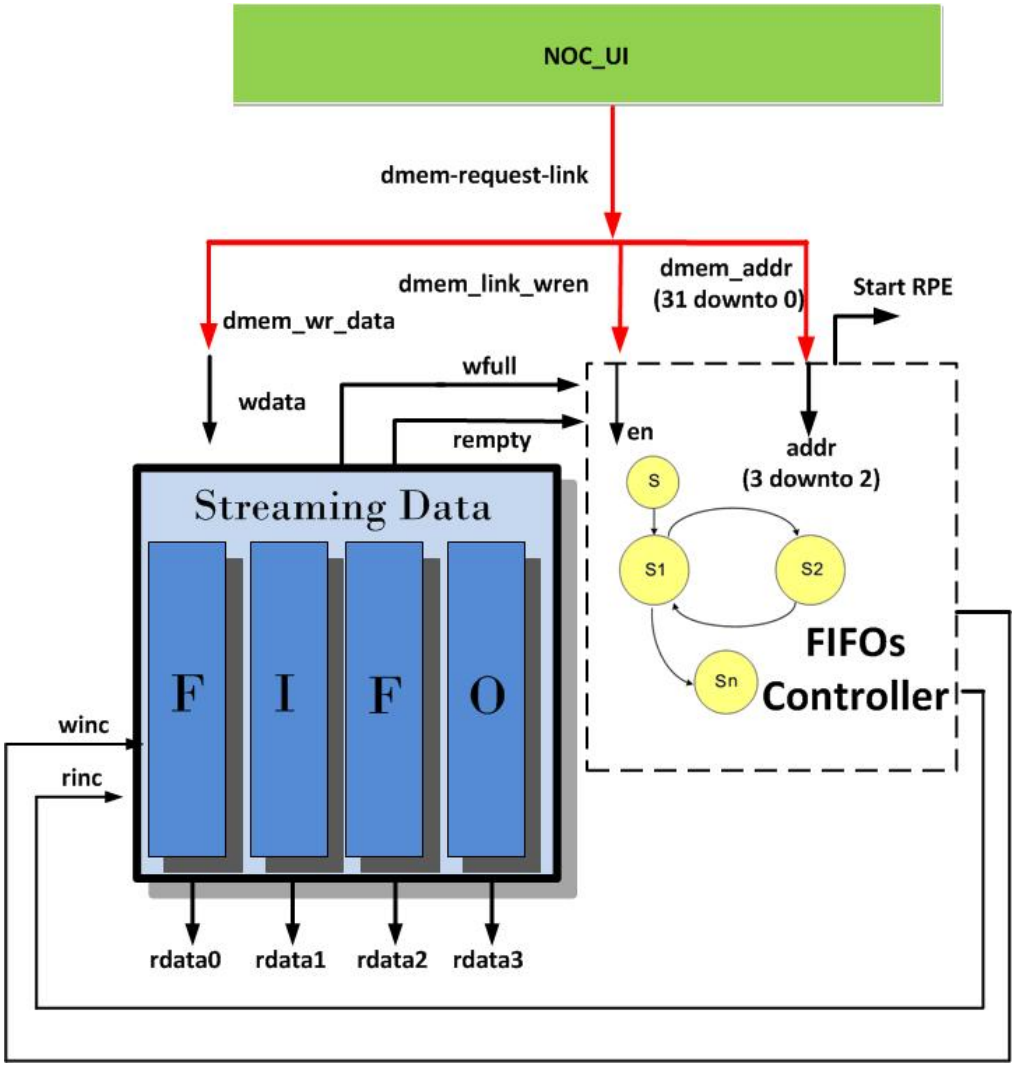

Figure 4.14: FIFOs controller

FIFOs. The destination FIFO is selected through the write increment (winc) signal with the third and second bit of the address. The first two bits of the address are left out because the data are 32-bits. In the table 4.2 this selection pattern is explained.

When the FIFOs become full their write full (wfull) signal is activated. At this point no more data is written to the streaming data component which means all the bits of the winc signal are zero. Now its time to start processing these data and activate the RPE component. So "Start RPE" is activated and the data are read from the FIFOs four at a time by activating the read increment (rinc) signal. The reading procedure is done until the FIFOS are empty. The "rempty" signal points out whether the FIFOs are empty or not.

Table 4.2: FIFOs selection to write data

Address (3 downto 2)	winc	Description
"00"	"0001"	Data is written to the FIFO number 1
"01"	"0010"	Data is written to the FIFO number 2
"10"	"0100"	Data is written to the FIFO number 3
"11"	"1000"	Data is written to the FIFO number 4

RPEC

Figure 4.15 shows the Reconfigurable Processing Element Controller (RPEC) and its connections. RPEC receives its input signals from configuration memory, FIFOs controller and streaming data components. The clock signals are the inputs of the reconfigurable core. RPEC outputs are sent to RPE, RPE output multiplexer and NoC_UI.

The "Reconfigurable Cells_en" signal is connected to the enable input of all four reconfigurable cells inside the RPE. When the "Start RPE" signal is activated by the FIFOs controller, this signal is also activated.The reconfigurable cells enable signal will be deactivated when there are nomore data to process. This means that no more data are left in the streaming data component. So when "rempty" signal is active the reconfigurable cells enable signal is deactivated.

The "Mux4_Sel" signal select the order with which we want RPE outputs to be sent to the NoC_UI. Result pattern word from the configuration memory is received as input to select the four consecutive values of the Mux4_Sel signal. The clock frequency with which the four input multiplexer select signal will change is same as the "System clk" frequency. More details of this can be found in table 4.1.

For every final result data that is generated from the RPE an address should be assigned. RPEC has a counter that counts the number of RPE outputs data. RPEC adds the counter value to the "Core Address" that was received from the configuration memory and generates the proper address.

"wr_en" signal is active when a data is ready to be sent to the NoC_UI. This signal is activated and deactivated same as reconfigurable cells enable signal, but the only difference is that "wr_en" turns one two "RPE clk" cycles after "Reconfigurable Cells_en". This is done because RPE outputs are generated with two REP cycles delay from the time their inputs were assigned.

The NoC_UI input is "cpu-request-link" bus. This bus is a combination of the RPE output data, the write enable signal and the result data address. These links can be seen in the figure.

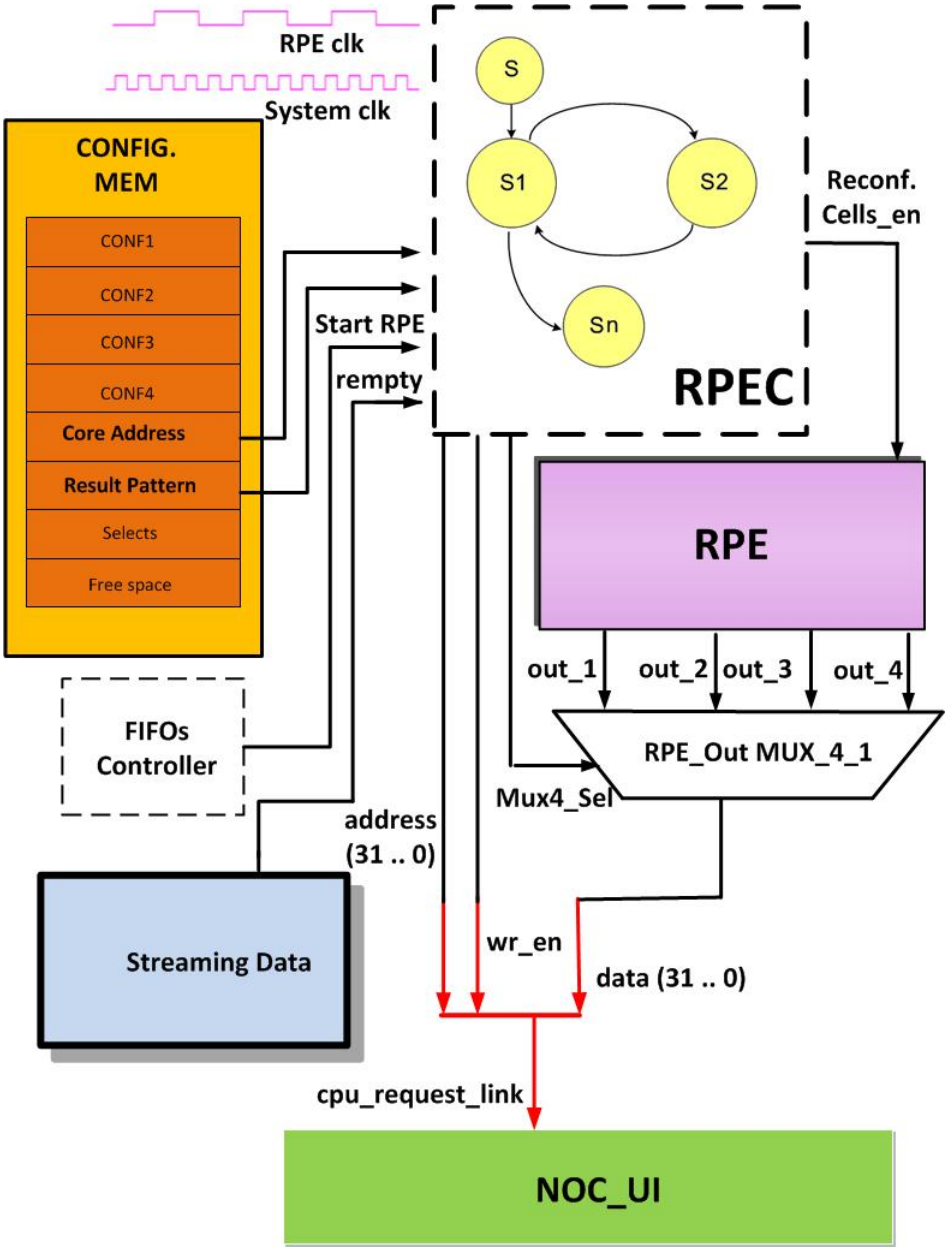

Figure 4.15: RPEC

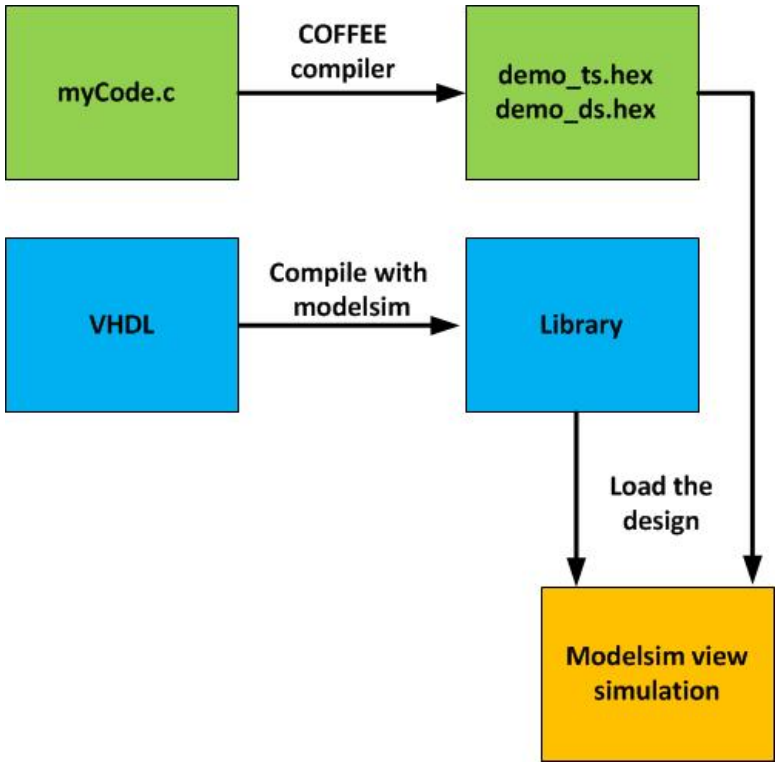

Figure 4.16: Design compilation procedure

4.2 Compilation and Simulation

All the components explained in chapter 3 were coded in VHDL (VHSIC Hardware Description Language). In order to test and observe the simulation of the architecture a c code was implemented and fed into the system. This procedure can be explained using the diagram in figure 4.16.

The C code is compiled using COFFEE compiler. This compiler has been developed with the COFFEE core here at TUT. By compiling the C code two memory image files are generated. "demo_ts.hex" is memory image file which stores the instructions and "demo_ds.hex" is data memory image file.

The VHDL codes were compiled using Modelsim software. For compiling, all the components were organized in bottom to top level in our script file. After the compilation of the VHDL codes their library files were generated and stored in a predefined location.

For compiling the C code and VHDL files Cygwin which is a Linux-like environment for windows was used. Through "vsim" command, the top entity is called and the design is loaded in to the Modelsim.

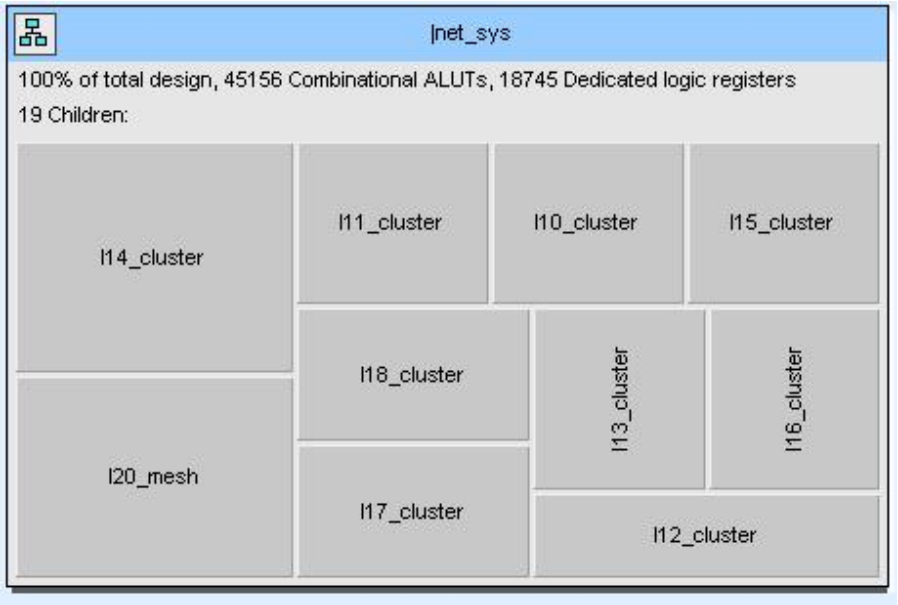

Figure 4.17: Design partition planner

4.3 Synthesis

The design was synthesized using Altera Quartus software. Stratix IV was chosen as the target FPGA. Figure 4.17 shows the design partition. The clusters from I10 to I18 are the 9 nods of the overall system. We can see that I14_cluster has a larger partition compared to the other nodes, and this is because it is COFFEE core. The other 8 clusters are our reconfigurable nodes. Other than these nine nodes, we can view I20_mesh which represents the network.

Table 4.3 presents a summary of hardware resources that our architecture consumes. The register and logic elements numbers also show that COFFEE node consumes more hardware than the Reconfigurable node.

The number of DSPs used in this system is 166 and the table shows that each of the nine cores used sixteen DSPs.

Table 4.3: FPGA synthesis results of the proposed Reconfigurable MPSOC architecture

	# Registers	#Logic Elements	# DSP
System	18925	45154	144
NOC	4261	5295	0
COFFEE Node	5454	5708	16
COFFEE Core	5087	5177	16
Other peripherals	367	521	0
Reconf. Node	1143	4260	16
RPE	289	3479	16
Steaming Data	288	262	0
CONF. MEM	90	0	0

Chapter 5

Application Mapping Example

In the previous chapter the proposed architecture was explained. In this chapter we map an FIR filter application to our architecture. We will go through the applications theory and present an introductory section about FIR filters next we focus on the solution we have chosen to map this application in to our architecture. Finally in the last section the performance of the application on this architecture is compared to the same application on a single COFFEE core processor.

5.1 FIR Filter

In digital signal processing "filtering" is a common term that is applied to many applications. Any operation performed to extract wanted information from a digital signal is called filtering in general.Finite Impulse Response (FIR) filters is a filter whose impulse response is of finite duration that is because it settles to zero in finite time.

For a discrete-time FIR filter, the output is a weighted sum of the current and a finite number of previous values of the input. The operation can be described by the following equation. This equation defines the output sequence y[n] in terms of its input sequence x[n]:

$$Y_n = \sum_{i=0}^{N} X_{n-i} \cdot b_i \tag{5.1}$$

in this equation :
Y[n] is the output signal,
X[n] is the input signal,
b_i are the filter coefficients,

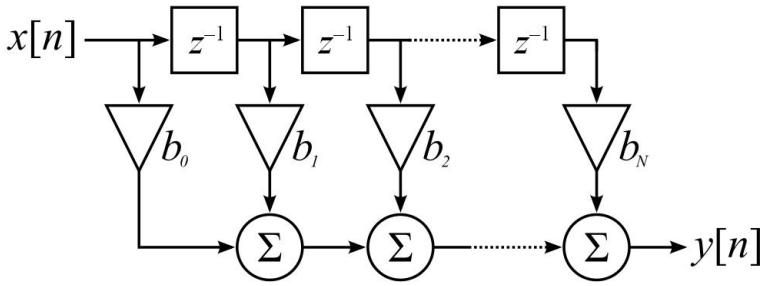

Figure 5.1: FIR filter diagram [5]

The coefficients make up the impulse response. N is the order of the filter; an N^{th}-order filter has $(N + 1)$ terms on the right-hand side. $X[n - i]$ in these terms are commonly called taps based on the structure of a tapped delay line that in many implementations or block diagrams provides the delayed inputs to the multiplication operations [6], [4].

In the next section we explain how an FIR filter is mapped to our architecture.

5.2 Mapping The Application

Figure 5.2 shows our nine-core architecture. As we described in the previous chapter, only the central controller is connected to system's I/O. We receive the X input signals and the coefficients b array from the input and we send the final Y array result to the output (see figure 5.2).

The central controller sends data and configuration that are required to each node. We decided to have 8 number of coefficients.

At the start the first 512 data of the X array and the coefficients are sent to the first node. When the reconfigurable nodes apply the process on the data the products that are as below are send back to the central controller.

$X_0 * b_0, X_1 * b_1, .., X_7 * b_7, .., X_{511} * b_7$

In the central core the total sum of these products are for calculation and these output signals can be generated:

$Y_0, Y_8, .., Y_{64}$.

At the same time that $node_0$ is sending the products to the central controller, the shifted X array is also send to the next node. In order for the next node to starts it process, the controller will send the coefficients plus the next X value (X_{512}) to $node_1$. The same process as the first node is operated to produce and $Y_1, Y_9, .., Y_{65}$.

Figure 5.3 shows the transmission of data that needs to be done between in the network in order to generate the first three results. Figure 5.5 and 5.6 present the order which the four FIFOs in the streaming data component are filled in

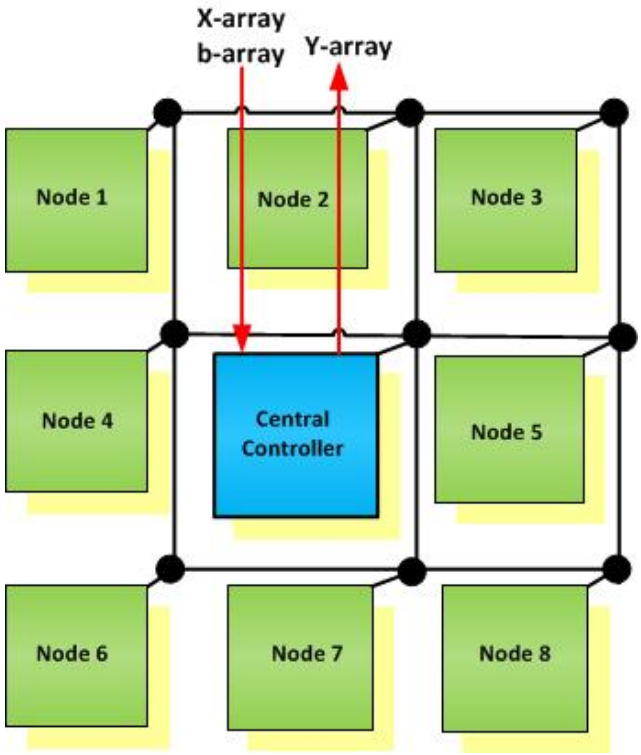

Figure 5.2: FIR filter I/O

$node_0$ and $node_i$ respectively. The rows of data that are processes to generate each Y output are also illustrated in these two figures.

In addition to the streams of data that are transfered in our MPSoC we also have configuration memory words which needs to be set for the application. Figure 5.4 shows configuration memory component of each reconfigurable node for our filter. So as it can also be seen in figure 5.7 the first to reconfigurable cells in the RPE act as a multiplier and the second two are repeated . The "Select" word in configuration memory is zero which means that the second row of reconfigurable cells receive their inputs directly from the outputs the cells above them. The result pattern is assigned to value six and that means that the pattern which the results are send to the NI is in this order :

R1,R3,R0 and R2.

This order can be seen from left to right in figure 5.7 where the data and address bus are shown. The central controller (COFFEE core) address is loaded in the memory and as it can also be seen in figure 5.4 the free space is used for the next node address. This addresses are send to RPEC then they are synchronized with their corresponding data and finally sent to the NoC.

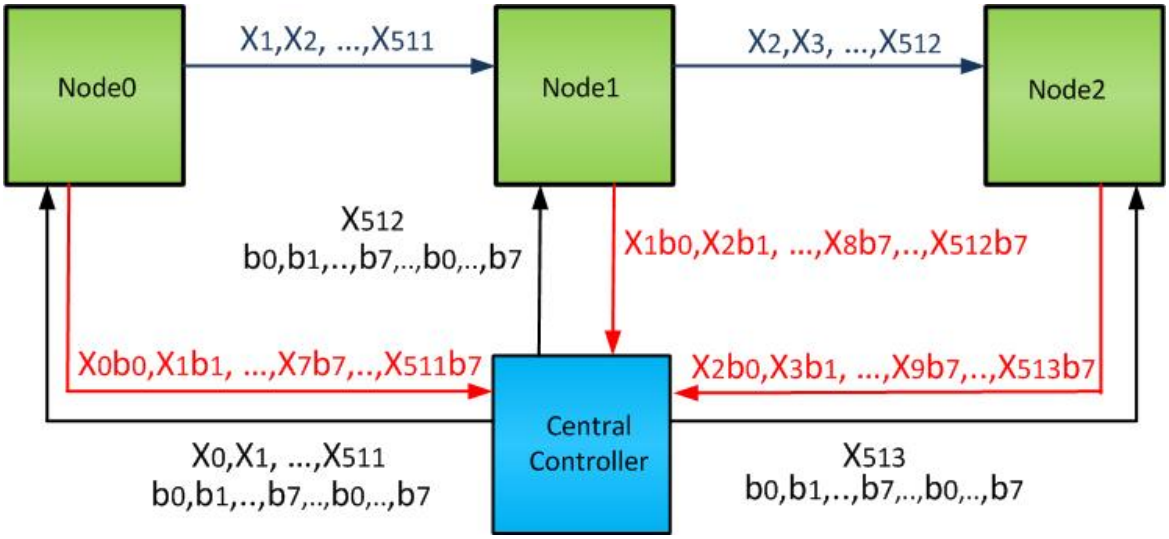

Figure 5.3: Stream of data for the first three results in the FIR filter application

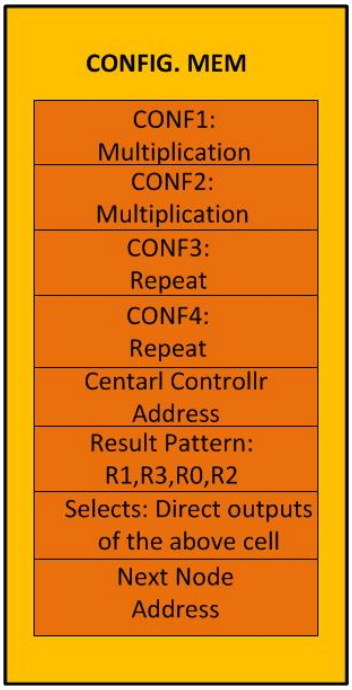

Figure 5.4: Configuration memory of each reconfigurable node in the FIR filter application

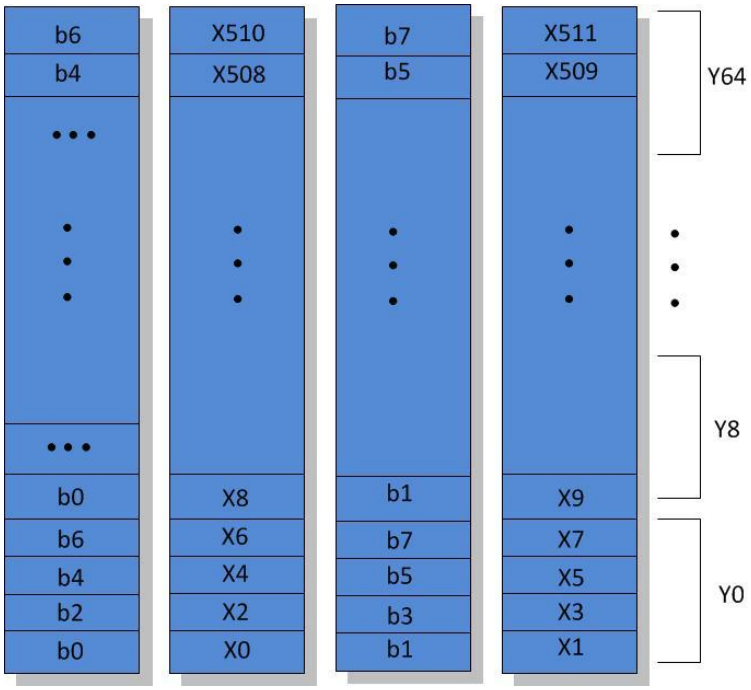

Figure 5.5: The order which the streaming data component of Node 0 is filled in the FIR filter application

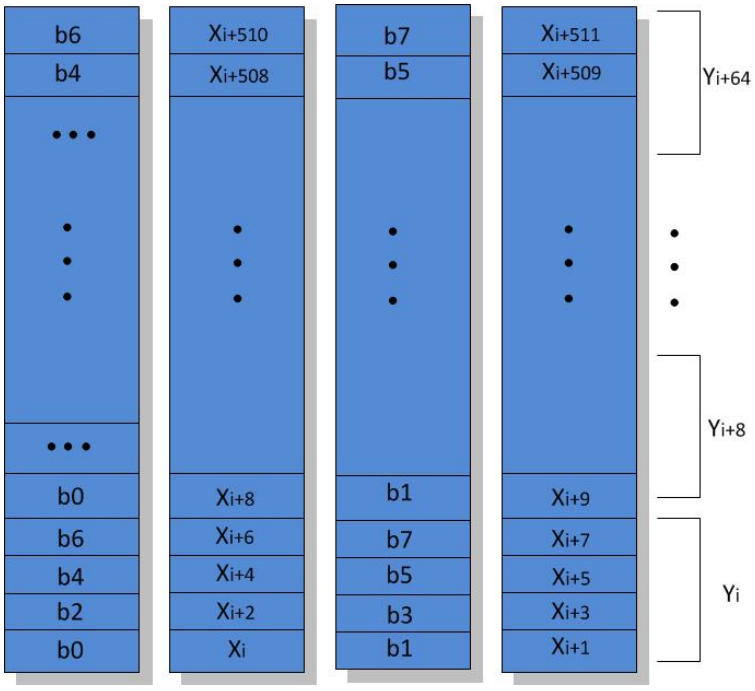

Figure 5.6: The order which the streaming data component of Node i is filled in the FIR filter application

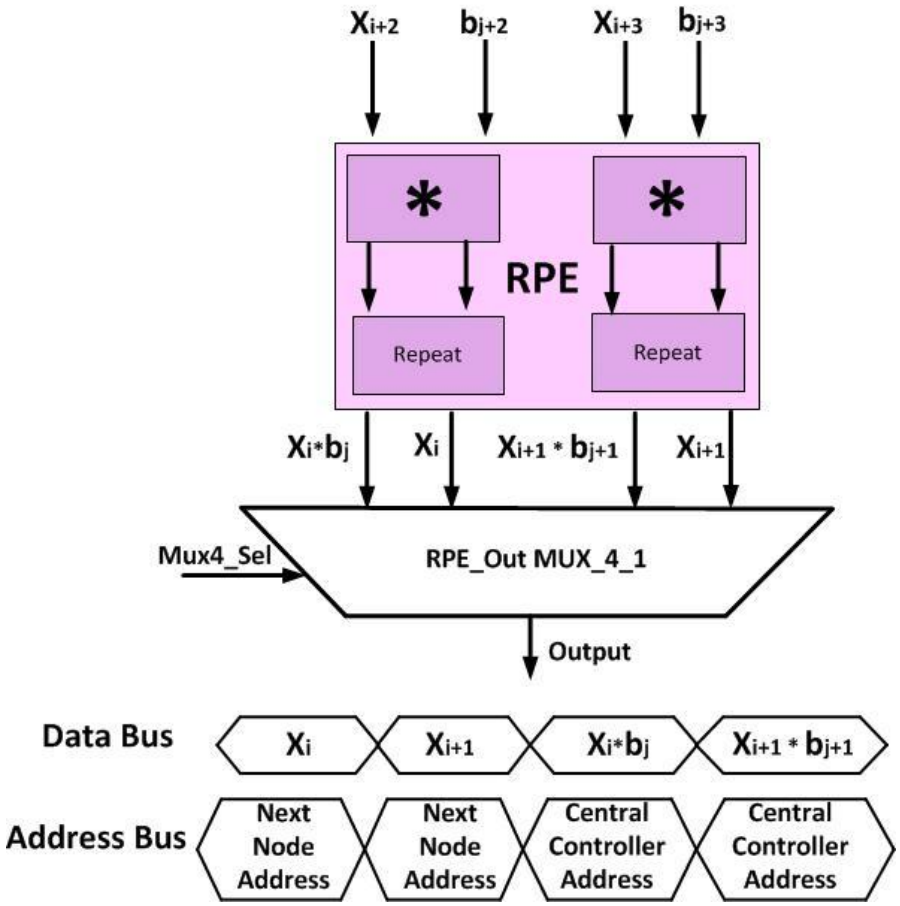

Figure 5.7: RPE process in FIR filter application

5.3 Performance analysis

After mapping the FIR filter on our design the same filter was implemented on a single COFFEE core processor. As the equation 5.2 shows, it takes 144 clock cycles to generate each sample.

$$FIR_{COFFEE} = 144_{Cycles/Sample} \tag{5.2}$$

In our architecture in every 16 clock cycles the $X_i * b_j$ products of 8 out put signals are ready and it take T_{sum} clock cycles to sum up every set of products for each output sample.Hence, we have:

$$8 * FIR_{multi-core} = 16 + 8 * T_{sum} \tag{5.3}$$

Simulations show that T_{sum} is equivalent to 19 system clock cycles. As a result :

$$FIR_{multi-core} = (16 + 8 * 19)/8 = 21_{Cycles/Sample} \tag{5.4}$$

$$\frac{COFFEE_{Performance}}{Multi - Core_{Performance}} = 144/21 = 6.857 \tag{5.5}$$

By comparing the two final results we can conclude that the application performance has a speed up of 6.8 in the recent architecture in comparison with a single COFFEE core.

Conclusion

In this thesis we discussed the characteristics of streaming applications and their wide range of domain. We saw that streaming processors fill up the gap between general purpose processors and specific purpose processors meaning that they can perform wide range of applications and also have high performance. It was realized that choosing a multi-processor will gain our goal to have higher performance and to facilitate the use of processor for general purposes, reconfigurability can be introduced in the design.

Hence, after introducing multi-processor system-on-chips and use of reconfigurable hardware in the state of the art processors we presented our own design. The proposed architecture is a MPSoC composed of nine cores interconnected to each other in a 3x3 mesh topology NoC. The central node is a COFFEE RISC core responsible for data and configuration words streaming and the eight other surrounding nodes act as reconfigurable processors. A streaming data component is located in the reconfigurable nodes to handle the data stream that enters the nodes. The RPE component is configured through the configuration word which were previously loaded in the configuration memory and made their process on the data stream.

After developing the architecture, it was simulated in modelsim and synthesized on a Startix IV Altera FPGA. The architecture of this design was coded in VHDL and the application was fed to the COFFEE core in C code. Using many pre-developed components in our architecture speed up the progress of system implementation and testing. The correctness of the design was first checked through simple test codes and finally an FIR filter application was mapped to the architecture. The FIR filter was also implemented on a single COFFEE core. The obtained speed-up for the proposed architecture was 6.8x, when compared to the COFFEE implementation. Also from the synthesis result we could observe that this architecture is about 20% smaller in size compared to a similar nine core MPSoC were all nodes are COFFEE RISC cores.

Bibliography

[1] *COFFEE Core User Manual - 2007.* [cited at p. iii, 21]

[2] Eia/tia: Mobile station-land station compatibility spec. tech. rep. 553 (1989). [cited at p. 5]

[3] *http://cva.stanford.edu/projects/imagine/.* [cited at p. 6]

[4] *http://en.wikipedia.org/.* [cited at p. 38]

[5] *http://images.google.com/.* [cited at p. iii, iv, 10, 38]

[6] *http://www.edaboard.com/thread180310.html.* [cited at p. 38]

[7] *http://www.picochip.com/page/42/multi-core-dsp-architecture.* [cited at p. 9]

[8] Lebak, j.: Polymorphous computing architecture (pca) example applications and description. external report, mit lincoln laboratory (august 2001). [cited at p. 5]

[9] Microsoft corporation: Microsoft directshow. online documentation (2001). [cited at p. 5]

[10] T. Ahonen and J. Nurmi. Programmable switch for shared bus replacement. In *Proc. Ph Research in Microelectronics and Electronics 2006 D*, pages 241–244, 2006. [cited at p. 24]

[11] Tapani Ahonen and Jari Nurmi. Hierarchically heterogeneous network-on-chip. In *EUROCON 2007 The International Conference on Computer as a Tool*, 2007. [cited at p. iii, 19, 20, 22]

[12] V. Bose, M. Ismert, M. Welborn, and J. Guttag. Special issue on software radios. In *Virtual radios. IEEE/JSAC*, 1999. [cited at p. 5]

[13] Fabien Clermidy, Romain Lemaire, Xavier Popon, Dimitri Ktenas, and Yvain Thonnart. An open and recongurable platform for 4g telecommunication: Concepts and application. 2009. [cited at p. iii, 11]

[14] Linfeng Ye Jean-Philippe Diguet and Guy Gogniat. Reconfigurable mpsocs for on-demand computing. In *Groupe d Etudes du Traitement du Signal et des Images*, 2009. [cited at p. 11]

[15] Andrew Duller, Daniel Towner, Gajinder Panesar, Alan Gray, and Will Robbins. picoarray technology: the tools story. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE05)*, 2005. [cited at p. 9, 10]

[16] F. Garzia, W. Hussain, and J. Nurmi. Crema: A coarse-grain reconfigurable array with mapping adaptiveness. In *Proc. Int. Conf. Field Programmable Logic and Applications FPL 2009*, pages 708–712, 2009. [cited at p. 26]

[17] Fabio Garzia. *From Run-Time Reconfigurable Coarse-Grain Arrays to Application-Specific Accelerator Design*. PhD thesis, Tampere University of Technology, 2009. [cited at p. 26]

[18] Michael Hubner and Jurgen Becker. *Multiprocessor System-on-Chip: Hardware Design and Tool Integration*. Springer, 2011. [cited at p. 8, 9, 12]

[19] Camille Jalier, Didier Lattard, Ahmed Amine Jerraya, Gilles Sassatelli, Pascal Benoit, and Lionel Torres. Heterogeneous vs homogeneous mpsoc approaches for a mobile lte modem. *DATE*, 2010. [cited at p. iii, 9, 14, 15, 16, 17]

[20] W. Wolf A .A. Jerraya and G. Martin. Multiprocessor system-on-chip (mpsoc) technology. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2008. [cited at p. 1, 7]

[21] Ujval J. Kapasi, William J. Dally, Scott Rixner, John D. Owens, and Brucek Khailany. Programmable stream processors. In *IEEE Computer*, 2003. [cited at p. 5]

[22] Ujval J. Kapasi, Peter Mattson, William J. Dally, John D. Owens, and Brian Towles. Stream scheduling. In *MICRO-34*, 2001. [cited at p. iii, 6]

[23] Brucek Khailany, William J. Dally, Scott Rixner, Ujval J. Kapasi, John D. Owens, and Brian Towles. Exploring the vlsi scalability of stream processors. In *In International Conference on High Performance Computer Architecture*, 2003. [cited at p. 5]

[24] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M.F. Kaashoek. The click modular router. *ACM Trans. on Computer Systems 18*, pages 263–297, 2000. [cited at p. 5]

[25] J. Kyllainen and J. Nurmi. Coffee - a core for free. *International symphosium on System-on-Chip. Procedings*, 2003. [cited at p. 19, 20]

[26] Jerome Martin, Christian Bernard, Fabien Clermidy, and Yves Durand. A microprogrammable memory controller for high-performance dataflow application. In *IEEE International Symposium on Networks-on-Chips (NoCs)*, 2009. [cited at p. 11]

[27] M. Mouly and M. Pautet. The gsm system for mobile communications. In *Cell&Sys*, 1992. [cited at p. 5]

[28] J. Nurmi. Silicon cafe : a heterogeneous multi-processor platform based on coffee (risc core). In *8th International Forum on Application-Specific Multi-Processor SoC*, 2008. [cited at p. 19]

[29] Scott Rixner, William J. Dally, Ujval J. Kapasi, Brucek Khailany, Abelardo Lpez-Lagunas, Peter R. Mattson, and John D. Owens. A bandwidth-efficient architecture for media processing. In *Micro-31*, 1998. [cited at p. 3]

[30] Piia Saastamoinen. *Program Code Compression on Single-and Multi-Core Embedded Systems*. PhD thesis, Tampere University of Technology, 2012. [cited at p. 19]

[31] Hao Shen and Frederic Petrot. Novel task migration framework on congurable heterogeneous mpsoc platforms. In *IEEE*, 2009. [cited at p. 8, 9]

[32] D. Tennenhouse and V. Bose. The spectrumware approach to wireless signal processing. In *Wireless Networks*, 1999. [cited at p. 5]

[33] William Thies, Michal Karczmarek, and Saman Amarasinghe. Streamit: A language for streaming applications. *Springer-Verlag*, pages 179–196, 2002. [cited at p. 3, 5]

[34] Xin Wang, T. Ahonen, and J. Nurmi. A synthesizable rtl design of asynchronous fifo. In *Proc. Int System-on-Chip Symp*, pages 123–128, 2004. [cited at p. 24]

[35] Linfeng Ye, Jean-Philippe Diguet, and Guy Gogniat. Modeling of reconfigurable mpsocs for on-demand computing. In *GRETSI*, 2009. [cited at p. 1]