Universitat Autònoma de Barcelona

DISSERTATION FOR THE DEGREE
DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE (SPECIALITY IN MICROELECTRONICS)

PARALLEL POST-PROCESSING SOLUTION FOR GNSS-R INSTRUMENT

Ph.D Dissertation by

Guo Yi
E-mail: guo@ieec.uab.es

CO-ADVISOR: Prof. Carles Ferrer

DEPARTMENT OF MICROELECTRONIC AND ELECTRONIC SYSTEM

UNIVERSITAT AUTÒNOMA DE BARCELONA (IEEC-UAB)

CO-ADVISOR: Prof. Antonio Rius

INSTITUT DE CIÈNCIES DE L'ESPAI (ICE-CSIC)
INSTITUT D'ESTUDIS ESPACIALS DE CATALUNYA (IEEC)

Bellaterra, December, 2011

Department of Microelectronic and Electronic System
Universitat Autònoma de Barcelona (UAB)

UNIVERSITAT AUTÒNOMA DE BARCELONA (UAB)
Department of Microelectronic and Electronic System

PARALLEL POST-PROCESSING SOLUTION FOR GNSS-R INSTRUMENT

Dissertation presented to obtain the degree of Doctor of Philosophy in Computer Science (Speciality in Microelectronics)

Author: GUO YI
Directors: PROF. CARLES FERRER AND PROF. ANTONIO RIUS

Prof. Carles Ferrer i Ramis, Professor from the Universitat Autònoma de Barcelona, Prof. Antonio Rius Jordán, Professor from Institut de ciències de l'espai (ICE-CSIC) and Institut d'Estudis Espacials de Catalunya (IEEC)

CERTIFY:

that the dissertation "**Parallel Post-processing Solution for GNSS-R Instrument**", presented by Ms. Guo Yi to obtain the degree of Doctor of Philosophy in Electrical Engineering, has been done under their direction at the Universitat Autònoma de Barcelona.

Barcelona, December, 2011

Prof. Carles Ferrer Prof. Antonio Rius

Preface

L-band signals transmitted by the Global Navigation Satellites Systems (GNSS) from its reflection off the Earth's surface allow for the inference of some of its geophysical properties. This concept is named GNSS-Reflectometry (GNSS-R), or PAassive Reflectometry and Interferometry System (PARIS).

The collected signals are processed by specialized GNSS-R receivers. This dissertation focusses on system design, which is primarily able to post-process the received GNSS-R data, with the purpose of reducing the sustained data throughput of the instrument, which is in the order of several Mbytes/sec. This amount of data poses very stringent requirements on GNSS-R designers .

In our study, we have taken as an example of GNSS-R receiver design, the GPS Open-Loop Differential Real Time Receiver (GOLD-RTR), which was designed, developed and built at the ICE (IEEC-CSIC). The problem that we faced could be stated thus: we have a system which produces 12.8 Mb/s in a sustained manner, and we need to reduce this rate by three orders of magnitude by applying suitable integration algorithms, to be discussed later.

The work towards my PhD has focused on one broad subject and applied this to the actual hardware design platform in order to study and address it. The subject was parallelism provision for the GNSS-R post-processing system, with special focus on the integration algorithms. The subject of parallelism provision is considered a multi-layer problem, the most discussed issues are related to the task-level and memory-level design.

The Symmetric Multi-Leon3 On Linux (SMLOL) platform, was developed to address the timing issues for the GNSS-R application. This subject was the parallelism provision for the task-level system, with special focus on the conventional Symmetric MultiProcessing (SMP) scheme.

As a multi-task problem, we used to assess the computational load, system performance and infer the system bottlenecks. However the unbalanced workload in the hardware design (among processors, cache, memory and bus) can not be fundamentally resolved through software methodology.

The Heterogeneous Transmission and Parallel Computing Platform (HTPCP) was later developed in order to balance the transmission and computing workload. This subject was the parallelism provision for the memory-level system. According to

the simulations results arrived at by MPARM emulator, we built and optimized the memory hierarchy system, in order to remove the bus busy ratio and memory access time between cache and main memory.

Moreover, dealing with the bus congestion issue, we implemented two types of element: Transmission Elements (TEs) and Processing Elements (PEs), as well as several interface designs: Massage Passing Interface (MPI) and Fast Simplex Link (FSL) in HTPCP.

The intended solution was to design, build and test a system with capacity to reduce the data flow three orders of magnitude by performing autonomous post-processing algorithms.

Prefacio

Las señales de banda trasmitidas por los sistemas de navegación global por satélite (GNSS, *Global Navigation Satellites Systems*) permiten averiguar algunas de las propiedades geofísicas de la Tierra al reflejarse en su superficie. Este concepto se llama reflectometría GNSS (GNSS-R) o sistema de interferometría y reflectometría pasivo (PARIS, *Passive Reflectometry and Interferometry System*).

Una serie de receptores GNSS-R especializados se encargan de procesar las señales recogidas. Esta tesis se centra en el diseño de dichos receptores, que permite principalmente procesar *a posteriori* los datos GNSS-R obtenidos, con el objetivo de reducir la tasa de transferencia de datos sostenida (*sustained data throughput*) del dispositivo, que es de alrededor de varios MB/s. Dicha cantidad de datos afecta enormemente al diseño de receptores GNSS-R.

En nuestro trabajo, hemos tomado como ejemplo de diseño de receptores GNSS-R el receptor GOLD-RTR (GPS *Open-Loop Differential Real Time Receiver*), diseñado, desarrollado y construido en el ICE (IEEC-CSIC). El problema al que nos enfrentamos es el siguiente: disponemos de un sistema que produce 12.8 Mb/s de forma sostenida y necesitamos reducir su magnitud tres veces mediante la aplicación de algoritmos de integración adecuados, que discutimos más adelante.

Las investigaciones realizadas durante mi doctorado, centradas en un tema muy amplio, las he aplicado al estudio y tratamiento de la plataforma de diseño del hardware correspondiente. El tema desarrollado fue el uso del paralelismo para el sistema de post-procesamiento GNSS-R, con especial atención a los algoritmos de integración. El tema del paralelismo se considera un aspecto problemático de múltiples dimensiones, siendo las más tratadas la del diseño de tareas y de memoria.

Se desarrolló una plataforma SMLOL (*Symmetric Multi-Leon3 On Linux*) para tratar los problemas de sincronización de la aplicación GNSS-R. Aquí se trató el uso del paralelismo para el sistema de tareas, con especial atención al esquema SMP (*Symmetric MultiProcessing*) convencional.

Como problema multitarea, evaluamos la carga computacional y el rendimiento del sistema y comprobamos las congestiones del sistema. Sin embargo, el desequilibrio en la carga de trabajo del diseño del hardware(en procesadores, memoria caché, memoria principal y buses) no se puede solucionar fundamentalmente mediante una metodología aplicada al software.

Posteriormente se desarrolló la plataforma HTPCP (*Heterogeneous Transmission and Parallel Computing Platform*) para equilibrar la carga de trabajo de transmisión y computacional. En este caso, se trató el uso del paralelismo con relación a la memoria del sistema. Según los resultados de simulación obtenidos con el emulador MPARM, construimos y optimizamos el sistema de jerarquía de memoria, para eliminar la tasa de ocupación del bus y el tiempo de acceso a la memoria entre la memoria caché y la memoria principal.

Asimismo, en relación con el problema de congestión en el bus, implementamos dos tipos de elementos: elementos de transmisión (TEs) y elementos de procesamiento (PEs), así como varios diseños de interfaces: interfaz MPI (*Massage Passing Interface*) e interfaz FSL (*Fast Simplex Link*) en HTPCP.

La solución deseada era diseñar, construir y probar un sistema con capacidad para reducir tres veces la magnitud del flujo de información mediante algoritmos de post-procesamiento autónomos.

To my parents and Lei,
To my son,

”Life is like a box of chocolate,you never know what you are gonna get.”
Movie Forrest Gump

Acknowledgments

The work presented in this thesis could not have been done without the aids and supports of many people. Therefore I have a great honor to express my sincere gratitude to all.

I would first like to thank my supervisors Carles Ferrer and Antonio Rius for everything, of which I would like to highlight all the support and help they provided me throughout the entire Ph.D. as well as encouragement in every endeavor. They were a big motivating force behind this herculean task I finished in last couple of years. Second, I would like to thank the David Atienza for all the support during my stage in EPFL, and as well as for all the collaborative work.

On a personal note, I must thank all my friends in Barcelona but special thanks goes to Lena Kanellou, L. Andrés Cardona for their helps and encouragement. I would also like to acknowledge my debt to Serni Ribó, without his help and patient, I can not go through the system debugging phase alone. I also want to thank Josep Sanz and Fran Fabra, without their wisdom, I can not fix the problem of ethernet and the endianess, and finally get the optimized result. Last but not least, I would like to thank Estel Cardellach for all the support documents she has given to me.

Finally, I am indebted to my husband Jiang Lei and my parents for their unconditional support and continuous encouragement throughout my work.

Guo Yi
Barcelona, November 18, 2011

List of Publications

This dissertation is based on the following Fourteen papers, referred to in the text by letters (A-N).

A. L. A. Cardona, J. Agrawal, Y. Guo, J. Oliver, C. Ferrer; "Performance-Area Improvement by Partial Reconfiguration for an Aerospace Remote Sensing Application", in *Proceedings of International Conference on ReConFigurable Computing and FPGAs*, Cancun, Mexico, November, 2011.

B. Y. Guo, S. Ribó, J. Sanz, A. Rius, C. Ferrer; "HTPCP: Real-Time Post-Processing Solution for GNSS-R Instrument", in *Proceedings of 3rd Int.Colloquium on Scientific and Fundamental Aspects of the Galileo Programme*, Copenhagen, Denmark, August. 2011.

C. Y. Guo, A. Rius , S. Ribó, C. Ferrer; "Heterogeneous Transmission and Parallel Computing Platform (HTPCP) for Remote Sensing Applications", in *Proceedings of SPIE, Microtechnologies*, Prague, Czech Republic, April. 2011.

D. L. A. Cardona, Y. Guo, C. Ferrer; "Partial reconfiguration of a peripheral in an FPGA-based SoC to analyse performance-area behaviour ", in *Proceedings of SPIE, Microtechnologies*, Prague, Czech Republic, April. 2011.

E. Y. Guo, A. Rius , S. Ribó, C. Ferrer; "On-board real-time parallel processing for GNSS-R Instrument GOLD-RTR", in *GNSSR-10 Workshop* , Barcelona, Spain, October 21-22. 2010.

F. Y. Guo, D. Atienza, A. Rius, S. Ribó, C. Ferrer; "HTPCP:GNSS-R multi-channel correlation waveforms post-processing solution for GOLD-RTR Instrument", in *Proceedings of NASA/ESA Conf. on Adaptive Hardware and Systems (AHS-2010)*, Anaheim, CA, USA, pp. 157- 163, Jun. 2010.

G. Y. Guo, D. Atienza, A. Rius, S. Ribó, C. Ferrer; "GNSS-R multi-channel correlation waveforms post-processing solution for GOLD-RTR Instrument" in *PHD FORUM of Design, Automation & Test in Europe (DATE-2010)*, Dresden, German March 15-18, 2010.

H. Y. Guo, E. Kanellou, L. A. Cardona, A. Rius, and C. Ferrer; "Parallel workload analysis in SMP platform: a new modelling approach to infer the hardware efficiency for remote sensing application " in *Proceedings of SPIE, VLSI Circuits*

and Systems IV, Dresden, German vol. 7363, DOI: 10.1117/12.821549. May 2009.

I. A. Garcia-Quinchía, Y. Guo, E. Martín, C. Ferrer; "A System-On-Chip (SOC) Platform to Integrated Inertial Navigation Systems & GPS " in *Proceedings of international Symposium on Industrial Electronics (ISIE 2009)*, Seúl-Corea July 5-8, 2009.

J. C. Ferrer, Y. Guo, X. Wang, E. Kanellou; "Fault Tolerant NoCs architectures for aerospace applications" in *Forum on specification and Design Languages (FDL-2007): Workshop on System Design in Avionics & Space Industry*, Barcelona, Spain 18-20 of September 2007.

K. Y. Guo, C. Ferrer; "IMS/GPS integration with a novel real-time system platform for inertial data estimation" in *IEEE International Conference on the Computer as a Tool (EUROCON-2007)*, Warsaw,Poland pp. 2503-2583,ISBN: 1-4244-0813-X September 9-12, 2007.

L. Y. Guo, X. Fitó, C. Ferrer; "A novel real-time system platform development applied to an integrated inertial navigation system" in *Proceedings of the 15th IEEE Mediterranean Conference on Control and Automation (MED- 2007)*, Athens, Greece Paper T11-002,ISBN:978-960-254-664-2 June 27-29, 2007.

M. X. Fitó, E. Kanellou, Y. Guo, C. Ferrer; "Designing an inertial measuring system using system-on-chip and sensor microsystems integration" in *Proceedings of the IEEE International Sysposium on Industrial Electronics Conference (ISIE 2006)*, Montreal, Quebec, Canada pp.3285-3263, ISBN: 1-4244-0497-5 July 9-13, 2006.

N. X. Fitó, F. Lleixa, Y. Guo, C. Ferrer; "Microsystems and System-on-Chip Integration applied to Inertial Measuring System Development" in *Proceedings of the 9th IEEE International Workshop on Advanced Motion Control (AMC-2006)*, Istanbul, Turkey pp. 488-493, vol. 1 & 2, ISBN: 0-7803-9511-1 March 27-29, 2006.

Contents

List of Figures

List of Tables

Abbreviations

ALU	Arithmetic Logic Unit
AMBA	Advanced Microcontroller Bus Architecture
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
BBD	Black Box Definition
BRAM	Block RAM
CC-WAV	Cross-Correlation WAVeform
CPU	Central Processing Unit
Dcache	Data Cache
DCM	Digital Clock Manager
DCR	Device Control Register
DDM	Doppler Delay Map
DM	Delay Map
DPRAM	Dual-Port RAM
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processing
DSU	Debug Support Unit
EDAC	Error Detection and Correction
EDK	Embedded Development Kit
ESA	European Space Agency
FIFO	First In First Out
FPGAs	Field Programmable Gate Arrays
FPU	Floating Point Unit
FSL	Fast Simplex Link
FSM	Finite State Machine
FTP	File Transfer Protocol
GDB	GNU DeBugger
GNSS	Global Navigation Satellite System
GNSS-R	Global Navigation Satellite System - Reflectometry
GOLD-RTR	GPS Open-Loop Differential Real-Time Receiver
GPS	Global Positioning System
GUI	Graphical User Interface
HDL	Hardware Description Language
HTPCP	Heterogeneous Transmission and Parallel Computing Platform
HW	HardWare
ICACHE	Instruction Cache

ILP	Instruction-Level Parallelism
IP	Internet Protocol
IPC	Inter-Process Communication
JTAG	Joint Test Action Group
LMB	Local Memory Bus
LUT	LookUp Table
MB	MicroBlaze
MHS	Microprocessor Hardware Specification
MII	Media Independent Interface
MMPs	Massively Multi-Processing
MMU	Memory Management Unit
MPD	Microprocessor Peripheral Definition
MPI	Massage Passing Interface
MPs	Multi-Processors
MPSOC	Multi-Processors System-On-Chip
MSS	Microprocessor Software Specification
NOC	Network on Chip
NUMA	Non-Uniform Memory Access time
OPB	On-chip Peripheral Bus
OS	Operation System
PAO	Peripheral Analyze Order
PARIS	Passive Reflectometry and Interferometry System
PC	Personal Computer
PDF	Probability Density Function
PEs	Processing Elements
PLB	Processor Local Bus
PM	Processing Module
POPI	POlarimetric Phase Interferometry
POSIX	Portable Operating System Interface of Unix
PPC	PowerPC
PPS	Pulse-Per-Second
PUs	Processor Units
RAM	Random Access Memory
RF	Radio Frequency
RISC	Reduced Instruction Set Computer
RMII	Reduced Media Independent Interface
ROM	Read-Only Memory
SCI	Single Correlation Integration
SDRAM	Synchronous Dynamic Random Access Memory
SDK	Software Development Kit
SEU	Single Event Upset
SMLOL	Symmetric Multi Leon3 On Linux
SMP	Symmetric Multi-Processing
SNAPGEAR	SnapGear's embedded Linux distribution
SOC	System-On-Chip
SOW	Second Of Week

SRAM	Static Random Access Memory
SRMMU	SPARC Reference MMU
SW	SoftWare
TCP	Transmission Control Protocol
TEs	Transmission Elements
TLP	Thread-Level Parallelism
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
UMA	Uniform Memory Access
UP	Uni-Processor
UTP	Unshielded Twisted Pair
XCL	Xilinx Cache Link
XMP	Xilinx Microprocessor Project
XPS	Xilinx Platform Studio

Part I

Introduction

Introduction

The European Space Agency (ESA) proposed a method for extracting altimeter information from GPS signals reflected off the surface of the sea, called the Passive Reflectometry and Interferometry System (PARIS) in 1993 [1] or the GNSS-R scenario. There is a list of potential applications that has increased considerably since then, including the remote sensing of the state of the sea (surface roughness and its salinity), soil moisture levels, sea-ice characterization, and snow structures [2] which we review in Chapter 2. Because of the mostly non-coherent nature of the reflected signals, they cannot usually be tracked by standard GNSS receivers such as GPS, GLONASS, the future GALILEO and COMPASS. In order to conduct experimental work, a new instrument was designed by ICE (IEEC-CSIC) to compute and store the cross-correlation waveform (CC-WAV) in real time. This device is called the GPS Open-Loop Differential Real-Time Receiver (GOLD-RTR) [3]. Since 2005, this GNSS-R hardware receiver has been used in more than 42 campaign flights and in over 250 days of continuous ground-based observations.

In order to meet the portable and timing requirements for airborne missions, using FPGA as a post-processing platform will be a new approach. FPGA is well suited to form a parallel architecture since it incorporates several soft-cores. Taking into account time-to-market issues and rapid prototype developments, a shared memory Symmetric Multiprocessing (SMP) mounted with embedded OS appears as the first option to explore as we review in Chapter 3. However, the bus-based SMP has limited scalability due to bus congestion and shared memory issues [4]. A new approach is necessary to analysis the parallelism system from the workload perspective, particularly to provide a new emulation framework for addressing situations where is a bottleneck in different layers of Multi-Processing System-On-Chip (MPSOC) design.

The importance of post-processing the recorded waveforms of GOLD-RTR is that it provides diversity to the post-processing algorithm, and thereby reduces the load for the instrument downlink. A novel parallel platform HTPCP is designed to permit leverage of the processing capability and transmission load and this is reviewed in Chapter 4. Moreover, with regard to the post-processing algorithm, Coherent/Incoherent integration is introduced in Chapter 5, combined with the real campaign data to verify the correctness of the HTPCP design in GNSS-R post-processing systems.

1.1 Motivation

The aim of this PhD is to design a post-processing development board, where we can carry out a series of post-processing algorithms for the collected Cross-Correlation WAVforms (CC-WAVs) of the GOLD-RTR. With the development of SMP and NOC architecture, an on-board parallel processing system becomes possible. With sophisticated design tools, FPGA provides the perfect platform for interaction between software programs and hardware designs. For this reason we consider taking advantage of SMP especially in terms of its computing power, and of NOC especially in terms of its transmission power, and so minimize the shortcomings of each architecture, in order to develop a novel real-time post-processing system which can meet the actual bandwidth requirements of the GNSS-R application. In this way we hope to achieve a long term effective campaign and post-processing the CC-WAV in real time during that campaign.

The first challenge in this work is to achieve the system real-time processing features in the multi-processor hardware architecture design. This work will try to address the conflict between the intensive computational load and the transmission load in the SMP design.

The second challenge is the co-design process of the hardware and software. We attempt to find the system design bottlenecks by testing and analyzing the workload model at different levels. The chosen methodology for addressing this problem is the use of memory hierarchy designs and bus designs in order to leverage the workload between the computation and transmission systems. Basically the redesign of these two parts can solve the issues of bus congestion and memory allocation. Moreover, we provide a timing controller to monitor the timing of the input and output data.

Last but not least, in order to study the GNSS-R scenario, an incentive for working the post-processing algorithms for GNSS-R instrument is considered as a scientific experiment for the geophysical exploitation. One example, the Coherent/Incoherent Integration algorithm, has been studied as part of this PhD. This concept has been tested and proven for the future measurements of altimetry experiments, sea roughness caused by wind, soil moisture and sea ice etc..

1.2 Objectives

The main objectives of this PhD are summarized as follows:

- Identify the timing constraints of the post-processing design of GOLD-RTR, emphasizing the basic scientific demonstration of a parallel system.
- Test and simulate the post-processing algorithm on SMP platforms, in order to find out the system bottlenecks in the architecture design.

- Develop the post-processing algorithms to realize a fine-grained parallelism application on the target board, i.e., HTPCP. Fill the gap between the processing time and the storage of the CC-WAV for GNSS-R applications.
- Develop a novel architecture, HTPCP, which has been proposed as a promising technique for GNSS-R post-processing systems, which could meet the real-time requirement and the diversity requirement of processing the scientific demonstrations.
- Verify the HTPCP with the real campaign experiment, in order to demonstrate the correctness of the system design.

The fundamental platform for the development of this thesis is the research project of National Space Plan (CICYT ESP2005-03310), the project entitled ”ASAP: Altimetric and Scatterometric Applications of the PARIS Concept”, which was developed by ICE (IEEC-CSIC) and INTA. The platform will be used for future campaigns of the GOLD-RTR Instrument.

1.3 Document Structure and Context

This document reflects the line of my PhD research towards obtaining the doctorate. At the beginning of my doctorate course, the subject was decided by implementing the post-processing algorithm of GNSS-R application by FPGA. With the gradual deepening of the study, my research subjects constantly expanded from a general signal processing problem to a concrete multi-core processing, and finally reached a parallel system design. This document is not only the result of this project, but rather an experimental platform. It covers the HTPCP hardware design, post-processing algorithm design, laboratory readiness tests, and the aircraft campaign result experiments.

The rest of the thesis is organized as follows.

- Chapter II discusses the state of the art, focussing on setting out the latest situation in two fields: GNSS-Reflectometry (GNSS-R) technique and Parallel system development.
- Chapter III describes the parallel system based on the SMP scheme, namely the Symmetric Multi-LEON3 On Linux (SMLOL) platform, which is designed and analyzed to implement the post-processing algorithm over a task-level parallelism system. The in-depth analysis of timing performance is based on the MPARM emulator, in order to discover the bottleneck of the SMLOL platform. It provides a framework for applying the post-processing algorithm as well as testing its performance in a realistic scenario.
- Chapter IV focusses on the novel parallel platform, called HTPCP, which is presented in order to carry out the real time post-processing for GNSS-R

applications. Moreover, two problems are proposed and solved, 1) Parallelize the inherent serial output of the GOLD-RTR instrument and 2) Post-process the multi-channel (I and Q) correlators in parallel. The seven laboratory readiness tests demonstrate the correctness of the IP cores design in HTPCP.

- Chapter V presents two experiment results based on the real campaign environment in order to demonstrate the correct operation of the entire system (GOLD-RTR + HTPCP + Control PC). Moreover a detailed description of the post-processing algorithm - coherent/incoherent integration is introduced, to deal with the data reduction and the data mass storage issues.
- Chapter VI concludes the thesis.

Bibliography

[1] M. Martín-Neria, "A passive reflectometry and interferometry system (paris):application to ocean altimetry," *ESA Journal*, vol. 17, no. 4, pp. 331–355, 1993.

[2] E. Cardellach, F. Fabra, O. Nogués-Correig, S. Oliveras, S. Ribó, and A. Rius, "Gnss-r ground-based and airborne campaigns for ocean, land, ice and snow techniques: application to the gold-rtr datasets," *RADIO SCIENCE*, 2011.

[3] O. Nogués-Correig, E. Cardellach-Galí, J. Sanz-Camderrós, and A. Rius, "A gps-reflections receiver that computes doppler-delay maps in real time," *IEEE Transactions on Geoscience and Remote sensing*, vol. 45, no. 1, pp. 156–174, 2007.

[4] Y. Guo, L. Kanellou, A. C. Luis, A. Rius, and C. Ferrer, "Parallel workload analysis in smp platform: a new modelling approach to infer the hw efficiency for remote sensing application," in *Proceedings of of SPIE–VLSI Circuits and Systems IV (SPIE'09)*. Dresden, Germany: SPIE, May 2009.

Part II

State of the art

State of the art

2.1 Introduction

This dissertation is based on the previous knowledge in two fields: GNSS-Reflectometry (GNSS-R) and parallel system design. This Chapter reviews the state of the art of both fields: *a)* the GNSS-R technique and *b)* the parallel system development. Here we cover the theoretical basis and technology development related with our project. The first category is described as the GNSS-R post-processing relevant design, it focuses on the implementation of the GNSS-R instrument and its performance for geophysical exploration. The second category is described as the parallel system design based on the basic principles of the parallel system design, and it focuses on the exploration of the parallel system, the definition of parallelism, parallel architecture, parallel algorithm and parallel programming model, in order to argue that they represent an important and distinct category of computer architectures.

2.2 GNSS-R Post-Processing Relevant Design

In this Section, we summarize the basic principles of the GNSS-R technique. We mainly focus on the GNSS-R scenario, GOLD-RTR instrument and GNSS-R post-process applications. It is organized as following:

- **The GNSS-R Scenario**, define the GPS transmission signal structure, frequency carriers, carrier modulation, cross-correlation function, and the physical significance of the reflected signal.

- **The GOLD-RTR Instrument**, briefly introduce the composition of the instrument and the work principle of GOLD-RTR.

- **The GNSS-R Post-Process Applications**, devoted on the description of the basic aspects involved in the technique of using GNSS signals reflected off the ocean surface to infer its physical properties.

2.2.1 GNSS-R Scenario

In 1993, the European Space Agency (ESA) proposed a method for extracting altimeter information from GPS signals after their reflection off the sea surface. This concept is created by the Passive Reflectometry and Interferometry System (PARIS) as passive multistatic radar to monitor mesoscale ocean altimetry [1]. This theoretical notion has turned into one of the most recent applications for the Global Navigation Satellite System (GNSS), which is the used of reflections for oceanographic remote sensing purposes. So far it mainly extends to the use of GPS signals combined with airborne receivers to form a type of bistatic radar. The idea is to capture both the GPS signals coming directly from the transmitting satellites and the ones that have been reflected off the ocean surface as shown in Fig.2.1. The delay between them can be used to determine aircraft altitude, and extract the ocean scattered signal information about wave height, wind speed and wind direction. These techniques allow further processing of the real-time computed 1-ms waveforms in a flight campaign over the ocean, ice, or ground, which can be used to obtain geophysical parameters such as sea level and tides, sea surface mean-square slopes, ice roughness and thickness, soil moisture and biomass, or future applications.

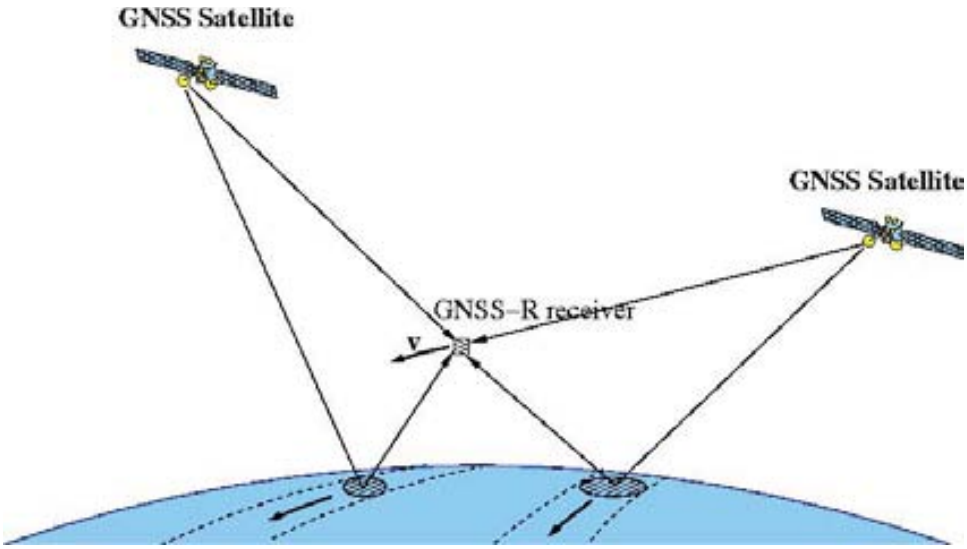


Figure 2.1: GNSS-R scenario: GNSS signals reflected on the ocean surface are used to gather their properties like roughness or level. (Adapted from Nogués-Correig et al.[2])

As shown in Fig.2.2, satellite transmissions occur using two different frequency carries, L1 = 1575.42 MHz and L2 = 1227.60 MHz, both obtained by multiplication of the fundamental frequency (10.23 MHz) with the factor 154 and 120, respectively. These frequencies are suitable for satellite transmissions because in this range both rain effects and galactic noise are minimized. The low Signal-to-Noise (SNR) L-band (L1 = 1575.42 MHz, λ = 0.1905 meter wavelength) electromagnetic field is transmitted by satellite orbiting at $\sim$ 20000 km altitude. The signals are transmitted at Right-Hand Circular Polarization (RHCP) continuously. A detailed description of the GPS can be found in Spilker et al. [3] or Misra and Enge et al. [4].

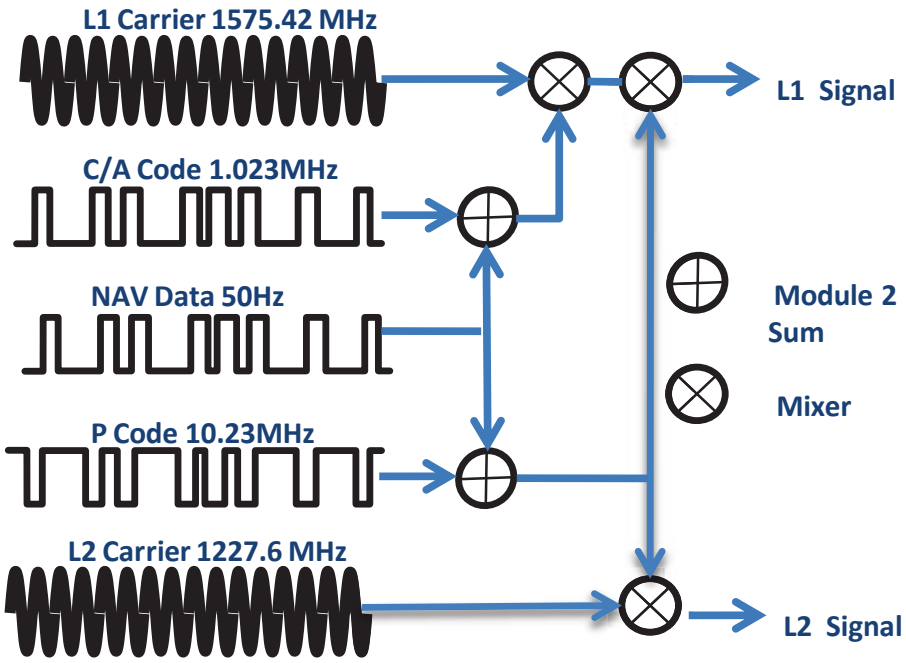


Figure 2.2: GNSS-R TX signal model.

The carriers modulate two types of codes, called C/A code (Coarse/Acquisition) and P code (Precise). The former is used for civil applications within the Standard Positioning Service (SPS). The latter allows access to the Precise Positioning Services (PPS), available for military purpose. These codes modulate the signal by introducing 180^o phase-shifts. One of the codes is the navigation code, used to help real-time positioning. The other codes are designed to isolate the signal transmitted by one particular GPS space vehicle from the others received simultaneously which build up a Code Division Multiple Access (CDMA) scheme that spreads the band-width of the data signal and drops down its power spectral density. Each satellite accesses the transmission medium using its own pseudorandom sequence, and it can be distinguished and identified from the rest because it assigned sequences exhibits good cross-correlation properties with those of the other satellites.

These cross-correlation functions are also called waveforms, or delay maps (DM). A detailed description of cross-correlation functions can be found in Cardellach et al. [5], where we acquire the Fig.2.3 to describe the correlation function. As shown, it is an ideal triangle in amplitude units in direct propagation conditions (no reflection). The half-width of the triangle is the chip-length, which means the time interval for potential phase-shifts. For the C/A code, this is $\sim$ 300 meter. The delay of the signal is measured by either displacements of the peak of this triangle function (group delay), or more precisely by tracking the changes in the phase of the carrier. The ionospheric and lower atmospheric conditions, together with multi-path environment introduce some additional delays. When the signal reflected off the Earth surface is collected, (1) the polarization is mostly swapped, from RHCP to Left-Hand Circular Polarization (LHCP). This effect depends on the dielectric properties of the surface

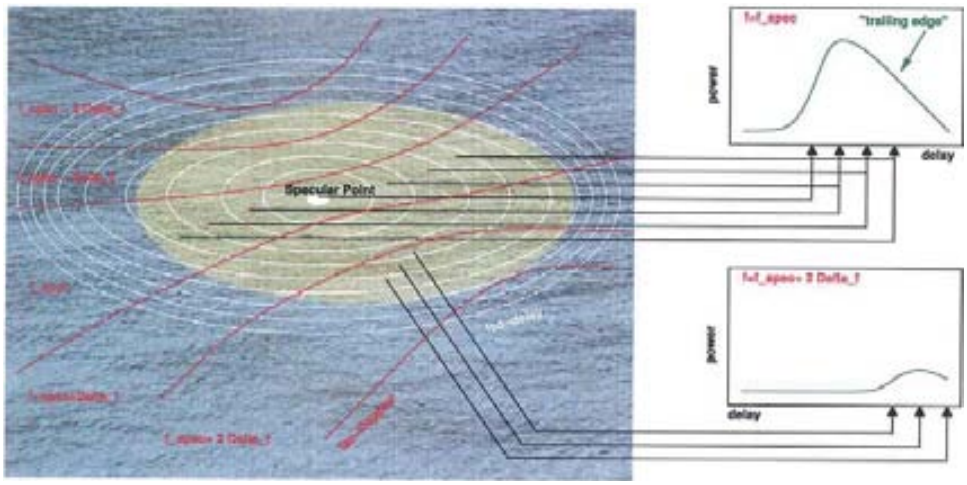


Figure 2.3: The composition of all the DM-slices generates the Delay-Doppler Map (DDM). (Adapted from Cardellach et al. [5])

as well as geometry of the reflection; (2) the correlation function is distorted, since the diffuse scattering, the antenna collects signals reflected at the specular point as well as at other points within the surface, which add contributions to the cross-correlation function at longer delays than the specular one. The waveform is not a triangle function any more as shown in Fig.2.3, but the trailing edge decays at lower slope than the leading edge. Moreover, in certain dynamic conditions, the Doppler frequency suffered by the off-specular reflections differs significantly from the Doppler frequency at the specular point. These signals are then filtered out by the cross-correlations and coherent integration process. In those cases to retrieve the complete glistening zone, a set of cross-correlations must be performed at slightly different frequencies around the specular one, generating a Delay-Doppler Map (DDM). All these waveforms can be provided as a set of complex number (C-DM, C-DDM) given in amplitude-units, or non-coherently integrated in time to reduce the noise, given in power-units (P-DM, P-DDM).

2.2.2 GOLD-RTR Instrument

The GOLD-RTR instrument [*GOLD-RTR*] is designed by the ICE (IEEC-CSIC). A detailed description of the GOLD-RTR can be found in [2]. Here we briefly introduce the composition of the instrument and the work principle of GOLD-RTR. As shown in Fig.2.4, the GOLD-RTR includes two parts: the RF front-end and the signal-processing back-end. The front-end is composed of three I and Q direct downconversion chains with their corresponding three antenna inputs. The local oscillator synthesizer that feeds them coherently with a common-tone down-shifted 300 KHz from L1 carrier. The system reference oscillator operates at 40 MHz, therefore, the data sample frequency is 40MHz. The back-end is composed of a one-bit A/D conversion stage that converts the three I and Q Base-Band pairs to digital and six comparators performing the sign extraction. After sampling, the bus of six

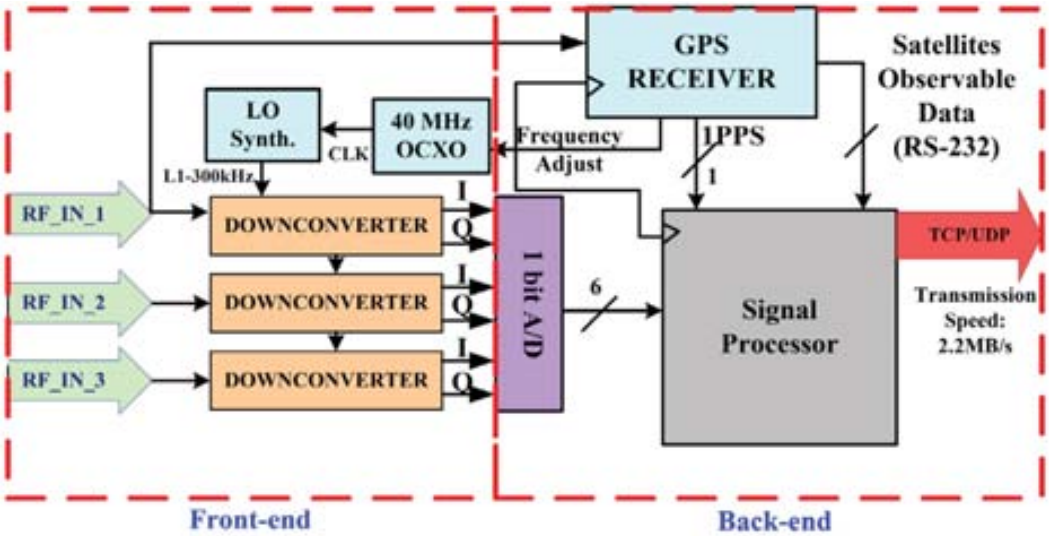


Figure 2.4: Block diagram of GOLD-RTR instrument.(Adapted from Nogués-Correig et al.[2])

base-band 1-bit signals is split into **ten** equal copies, which are input to the ten correlation channels. Each correlation channel contains 64 single-lag correlators in order to provide **64-lag** waveforms at **50 ns** spacing. The instrument has a set of 640 single-lag IQ correlators, grouped into ten 64-lag independent correlation channels that work simultaneously. The real-time signal processor computes the waveforms. A commercial GPS receiver card (Novatel) computes the navigation solution from antenna input 1. The GOLD-RTR communicates with Control PC through a full-duplex Ethernet link at 100 Mbps, using a standard Unshielded Twisted Pair (UTP) cable ended with RJ-45 connectors. An important issue is that the two down-looking antenna used to collect GPS reflected signals, and the up-looking antennas used to collect the GPS direct signals. The geometrical projection of such a baseline into the scattering direction enters, straightforward, as a relative phase between both LH and RH signals.

An important issue is that there is flexibility in the loaded signal models in order to force offset-values in delay or frequency, for both direct and reflected signals. All these features make it possible to assign different satellites to different channels, or to assign the same satellite to all the channels, but with slight differences in the frequency model to produce a Doppler / delay map (DDM), or DDM of two polarizations of a satellite signal, or other possible combinations.

2.2.3 Examples of GNSS-R Post-Processing Applications

Nowadays, there are many applications of the GNSS-R post-processing. These applications are applied to the different experiments, and the results put forward in the campaigns. Cardellach et al. [5] has concretely described and classified the GNSS-R applications. The GNSS-R applications tend to be classified according to the observed surface: (a) Ocean; (b) Land; and (c) Ice and Snow. Another classification

Table 2.I: List of the GNSS-R techniques identified in the literatures.

Technique:	Bibliography:
	GROUP-ALTIMETRY
Peak-Delay	Martín-Neira et al. [6] (2001)
Retracking	Lowe et al.[7] (2002); Ruffini et al. [8] (2004)
Peak-Derivative	Hajj and Zuffada et al. [9] (2003); Rius et al. [10] (2010)
	PHASE-ALTIMETRY
Interferometric-beats	Cardellach et al. [11] (2004); Helm et al. [12] (2004)
5-Parameter DM Fit	Treuhaft et al.[13](2001)
Separate Up/Down Channels	Fabra et al. [14] (2011); Semmling et al. [15] (2011)
	OCEAN ROUGHNESS
DM-fit	Garrison et al. [16] (2002); Cardellach et al. [17] (2003); Komjathy et al. [18] (2004)
Multiple-satellite DM-fit	Komjathy et al. [18] (2004)
DDM-fit	Germain et al.[19](2004)
Trailing-edge	Zavorotny and Voronovich [20] (2000a), Garrison et al. [16] (2002)
Delay and Doppler spread	Elfouhaily et al.[21](2002)
Scatterometric-delay	Nogués-Correig et al.[2](2007), Rius et al. [10](2010)
DDM Area/Volume	Marchan-Hernandez et al.[22](2008); Valencia et al.[23](2009)
Discrete-PDF	Cardellach and Rius et al.[24] (2008)
Coherence-time	Soulat et al.[25](2004); Valencia et al.[26] (2010)
	OCEAN PERMITTIVITY
Polarimetric-ratio	
POPI	Cardellach et al.[27](2006)
	LAND
Soil-moisture cross-polar	Masters et al.[28](2004); Manandhar et al.[29](2006); Katzberg et al.[30] (2005); Cardellach et al. [31](2009)
Soil-moisture polarimetric-ratio	Zavorotny and Voronovich el at.[32](2000b),Zavorotny et al.[33](2003)
Object-identification	Lei-Chung et al.[34](2009)
	SEA-ICE
Permittivity by peak-power	Komjathy et al.[35](2000); Belmonte et al.[36](2007)
1st-year thickness VH-phase	Zavorotny and Zuffada et al.[37](2002)
Permittivity polarimetric ratio	
Permittivity POPI	Cardellach et al.[27](2006), Cardellach et al [31](2009)
Sea-Ice roughness	Belmonte et al.[36](2007)
	SNOW
Volumetric-scattering	Wiehl et al.[38]

regards the final product: (i) altimetry; (ii) roughness; (iii) permittivity parameters (such as temperature, salinity, or humidity), which determine the reflectivity level of the surface materials. Still, the observables used in the technique might also characterize the classification: (1) integrated power observables solely; or (2) high sampling complex correlation functions (amplitude and phase). Finally, we might need to distinguish between applications that require polarimetric or non-polarimetric observables. The data obtained in the experimental campaigns cover all the cases. Therefore, we focus on the applications that can be tested with the released data set, that is: applications of both integrated and complex-field observables, under categories (a) to (c), (i) to (iii), either polarimetric or not. Table 2.I summarizes the GNSS-R techniques and the reference literatures for each method.

2.2.3.1 Altimetry

The altimetric techniques can in principle be applied to reflections off any surface, but their performance will depend on the signal-to-noise ratio of the scattering. Therefore, GNSS-R altimetry has only been conducted on strongly reflecting surfaces/geometries, such as waters and smooth ice, or land at near-surface receiver altitudes [39].

The altimetry aims to resolve the vertical distance between the receiver and the surface, and/or the vertical location of the specular point (with respect a reference ellipsoid/geoid). The observables to deal with are the distances between transmitter, receiver and/or surface. They can be given in the space-domain (called *ranges*) or in the time-domain (called *delays*). They are related to each other by the speed of light, and from here on we will call them *range* or *delays* indistinctly. Because the GNSS-R observations are bi-static, the way to relate the ranges with the surface level will depend on the geometry (incidence angle), as well as other systematic effects (atmospheric, instrumental, antennas set up, etc). Here we focus on the *observable*, that is, the *altimetric range*. The altimetric range is the distance traveled by the reflected signal with respect to the one traveled by the direct radio-link. When the range is measured through the delay of the code (delay of the correlation function), it tends to be called *group-delay* or pseudo-range. When the carrier-phase can be tracked, variations in the range can be monitored with much better precision, since an entire cycle corresponds to $\lambda \sim$20 cm change ($\sim$0.5 mm/carrier-phase degree). This latter approach is called *carrier-phase altimetry*.

- Group-Delay Altimetry Observables

  In a smooth-surface reflection event, the distance between the path traveled by the reflected signal and the path traveled by the direct link would simply correspond to the range between the peaks of the two correlation functions. Nevertheless, and since most reflections occur off rough surfaces, this approach cannot generally be used. Rius et al. [40] shows that variations in the delay of the reflected peak mostly account for changes in the surface roughness.

  A few techniques have been used to identify the specular point delay in the waveform:

  Peak-Delay: the altimetric range is taken as the peak-to-peak delay. This peak delay can be directly extracted by combining some fields provided in the data.

  Re-tracking: which consists on fitting a theoretical model to the data. The best-fit model indicates the delay where the specular point lies.

  Peak-Derivative: identifies the maximum of the derivative of the leading edge as the specular point delay. The peak-derivative delay can be directly extracted by combining some fields provided in the data.

- Phase-Delay Altimetry Observables

  The phase at which the direct and reflected signals (ϕ_d and ϕ_r respectively) reach the receiver depends on the range between the transmitter and the direct and the reflected antenna phase centers respectively (r_d and r_r), through the terms $\phi_d \propto k \cdot r_d$ and $\phi_r \propto k \cdot r_r$, where k states for the carrier's wavenumber. The reflected-phase, given with respect to the direct one, thus becomes $\phi_{r-d} \propto k \cdot (r_r - r_d)$, where $(r_r - r_d)$ is the *altimetric range*, r_{r-d}. The phase-delay altimetry approach aims to measure the altimetric range by means of carrier-phase observations

ϕ_{r-d}. When the signals reflect off rough surfaces, the received phase ϕ_r becomes too noisy (*non-coherent*) and the technique cannot be applied.

In general, it can be used in very smooth surfaces (some ice surfaces [41]); from very low altitudes (a few meter); and over very slant geometries (a few degrees elevation). Some of the released data sets are suitable for phase-altimetry, following the techniques below:

Interference beats: When the altitude is low enough, or the observation at grazing angles, the delay between the reflected and direct signals is short and their correlation functions overlap, producing interference-beats. These beats are oscillations of the amplitude and phase of the total field (sum of the two signals), and they occur at the frequency $(1/\lambda)d(r_r - r_d)/dt$. [11] obtained phase-delay altimetry by analyzing the interferences found in radio-occultation data from the CHAMP Low-Earth Orbiter. Similarly, [12] used an equivalent approach to perform phase-delay altimetry off of an Alpine lake.

5-parameters: A more robust fit is performed in [13], where the whole complex RHCP (direct+reflected) waveform is used to extract five parameters , among them the altimetric range.

Separate channels: When the delay between the two radio-links is longer, and their correlation functions do not or just weakly overlap, [14; 15] used separate up/down channels to measure the relative phase ϕ_{r-d}, obtaining sea-ice phase-altimetry which clearly reproduced the tidal signatures.

2.2.3.2 Ocean Wind and Roughness

As the GNSS link reflects off the sea surface, its roughness might scatter the signal in a wide range of output directions. As a result, the reflection is not specular (mirror-like), but it spreads in a *scattering pattern*: contribution from sea surface patches (*facets*) that have different orientation deviate the signals, and introduce further delays (longer ray-path distances than the nominal transmitter-specular-receiver radio-link). This changes the properties of the received signal and thus those of its correlation function. In general, the reflected waveforms present lower amplitudes when roughness increases, and the shape is also distorted: the leading edge (before peak) elongates, the peak gets further delayed, and the trailing edge (after peak) persists longer, with slower decay rate. Information about the surface roughness can be obtained from the analysis of these distortions.

The L-band navigation signals, of $\sim$0.2 meter electromagnetic carrier wavelength are not sensitive to sea surface roughness of spatial scales much smaller than the electromagnetic wavelength (such as wind instantaneously induced ripple). As a consequence, the GNSS-R ocean scattering observations can inform about intermediate roughness scales, which do not necessarily relate to the wind conditions. One of the open questions is a better understanding and modelling of this roughness term e.g.[42], and researchers are encouraged to use this data set for these

investigations. Two L-band radiometers have recently been and will be soon launched: SMOS and Aquarius respectively, which shall provide sea surface salinity and soil moisture measurements. Because of them, it is nowadays essential to understand and properly model the L-band roughness, and the bi-static scattering of L-band signals off the Ocean. The reason is that these issues are required for the proper modeling of the L-band emissivity, and the separation between the effects of the permittivity of the surface (salinity and temperature) and the roughness corrections.

The approaches to untangle roughness information that have been identified in the literature as suitable for being applied to this data set are codified and summarized in Table 2.I. Brief explanations are given below:

DM-fit: After re-normalising and re-aligning the delay-waveform, the best fit against a theoretical model gives the best estimate for the geophysical and instrumental-correction parameters [e.g. 16]. Depending on the model used for the fit, the geophysical parameters can be 10-meter altitude wind speed, or sea surface slopes' variance (mean square slopes–MSS). Note that the provided data sets, the time-delay alignment between the data and the model, *re-alignment*, is given by the variable `MaxDerDelay`, which identifies the specular-point.

Multiple-satellite DM-fit: extends the DM-fit inversion to several simultaneous satellite reflection observations, which resolves the anisotropy (wind direction or directional roughness [18]).

DDM-fit: The fit is performed on a delay-Doppler waveform [19]. In this way, anisotropic information can be obtained from a single satellite observation.

Trailing-edge: As suggested from theoretical models in [20], [16] implements in real data a technique in which the fit is performed on the slope of the trailing edge, given in dB.

Delay and Doppler spread: [21] developed a stochastic theory that results in two algorithms to relate the sea roughness conditions with the Doppler spread and the delay spread of the reflected signals.

Scatterometric-delay: For a given geometry, the delay between the range of the specular point and the range of the peak of the reflected delay-waveform is nearly linear with MSS. This fact is used to retrieve MSS [2]. The scatterometric-delay can be directly obtained in our data set by combining some of the provided fields.

DDM Area/Volume: Simulation work in [22] indicates that the volume and the area of the delay-Doppler maps are related to the changes in the brightness temperature of the ocean induced by the roughness. The hypotesis has been experimentally confirmed in [23].

Discrete-PDF: When the bi-static radar equation for GNSS signals is re-organised in a series of terms, each depending on the surface's slope s, the system is linear with respect to the Probability Density Function (PDF) of the slopes. Discrete values of the PDF(s) are therefore obtained. This retrieval does not require an

analytical model for the PDF (no particular statistics assumed). In particular, when the technique is applied on delay-Doppler-maps, is it possible to obtain the directional roughness, together with other non-Gaussian features of the PDF (such as up/down-wind separation [24]).

Coherence-time: Finally, when the specular component of the scattering is significant (very low altitude observations, very slant geometries, or relatively calm waters), the coherence-time of the interferometric complex field depends on the sea state. It is then possible to develop the algorithms to retrieve significant wave height [25; 26].

2.2.3.3 Ocean Permittivity

Polarimetric measurements are sensitive to the permittivity of the reflecting surface. For the Ocean surface, the permittivity at the L-band of the electromagnetic spectrum is essentially given by the salinity and the temperature [43].

Polarimetric ratio: the ratio between the co-polar (RHCP) and the cross-polar (LHCP) Fresnel reflection coefficients differs up to 10% between different salinity conditions. The direct inversion of the polarimetric ratio is neverthless an open question, since it will not only depend on the Fresnel coefficients, but on a more complex scattering process, for which the co-polar term is not properly modelled yet.

POlarimetric Phase Interferometry (POPI): Similarly, the difference between the phase of the complex co- and cross-polarized components of the reflected fields (RHCP and LHCP respectively) depends on the permittivity of the surface [27]. This is simply the phase of the polarimetric interferometric field, complex-conjugate multiplication between the co- and the cross-polar complex components. The long coherence time, of the order of minutes, of the polarimetric interferometric field, increases significantly the precision of the POPI measurement, which requires a theoretical sensitivity of a few degrees-phase. *Phase wind-up* [44] affects twice the POPI phase, and it must be corrected.

The data taken up today with the GOLD-RTR cannot provide absolute POPI values, because of instrumental issues, but they can provide POPI-variations. The GOLD-RTR has recently been modified to allow absolute POPI measurements, and the data taken during 2010 (27 flights) and later campaigns, to be posted in the server, will be ready for absolute POPI.

2.2.3.4 Land and Hydrological Applications

Several techniques to extract soil moisture information contents can be found in the literature. They are mostly sensitive to the 1-2 cm upper layer [30]:

Soil-moisture cross-polar: uses the LHCP SNR as the observable, from which the surface reflectivity is extracted. It can be normalized by the direct power level or even calibrated with observations over smooth water bodies [e.g. 28].

Soil-moisture polarimetric-ratio: Another method assumes that the received signal power is proportional to the product of two factors: a polarization sensitive factor dependent on the soil dielectric properties and a polarization insensitive factor that depends on the surface roughness. Therefore, the ratio of the two orthogonal polarizations excludes the roughness term and retains the dielectric effects [32; 33]. The same references note that real data did not support this hypothesis. Some of the assumptions might be too crude, and better modelling is required.

Object-identification: approach suggested in [34], based on a combination of computing the GNSS-R derived total reflectivity together with the carrier-phase positioning of both up- and down-looking antennas.

2.2.3.5 Ice and Snow Applications

For altimetric measurement over ice, the techniques intended to characterize several other aspects of the ice/snow, such as its permittivity (brine/temperature); texture; or sub-structure:

Permittivity by peak-power: obtains the effective dielectric constant empirically, as a function of the peak-power [e.g. 35].

Vertical and the horizontal polarizations: [37] suggested to infer the 1st-year thickness from the phase difference between the vertical and the horizontal polarized components.

Polarimetric Ratio: Similarly to polarimetric ratio, the ratio between the amplitudes of both polarizations relates to variations in the permittivity of the sea-ice (temperature and brine), especially at relatively low elevation angles of observation, around the Brewster angle. There is no algorithm in the literature, but the evolution of the polarimetric ratio captured during $\sim$200 days, including the freezing and melting process of the sea-ice in Disko Bay, Greenland.

Permittivity POPI: As in POPI, the technique uses the phase difference between the co- and cross-polar circular polarized components.

Sea-ice roughness: [36] obtained the sea-ice roughness by fitting the waveform shape.

Volumetric-scattering: [38] suggested a volumetric-scattering approach to model reflections produced in the sub-surface firn layers of dry snow.

The PARIS or GNSS-R techniques were suggested in early 90ies, whereas most theoretical and experimental research about their potential applications emerged

some years later. In this section, the ICE/CSIC-IEEC designed and manufactured a dedicated GNSS-R hardware receiver, the GOLD-RTR, with which more than forty air-borne flights and eight months ground-based campaigns were conducted over Ocean, Land, Sea-Ice and Dry-Snow. Several aspects of the GNSS-R require further investigation, and new applications or approaches might be envisaged. For these reasons, the GNSS-R data collected since 2005 with the GOLD-RTR are made available to the research community. With the aim of encouraging new users and new research, the paper sought to review in an understandably manner the GNSS-R applications, techniques and algorithms that could be potentially applied to these data sets, together with the data structure, and the suitable models to deal with them.

2.3 Parallel System Design

In this Section, we summarize the basic principles of the parallel computing, including the parallel system design, the parallelism definition, the parallel architecture, the parallel algorithm and the parallel programming model. It is organized as the following:

- **The Parallel System**, explores the main issues in the parallel system design.
- **The Parallelism**, describes that four types of parallelism should support in parallel system: Bit-Level Parallelism (BLP), Instruction-Level Parallelism (ILP), Data-Level Parallelism (DLP), Task-Level Parallelism (TLP).
- **The Parallel Architecture**, roughly classifies the parallel architecture into multi-processing system and multi-core system, and indicates two models for communication and memory architecture in parallel systems: Heterogeneous architecture and homogeneous architecture.
- **The Parallel Algorithm**, devotes to the description of Flynn's taxonomy, in order to decompose a given problem in many different ways and orders.
- **The Parallel Programming Model**, presents the three types of parallel programming models: shared address space, message passing, and data parallel programming. Moreover, presents the synchronization issue in the parallel programming design.

2.3.1 Parallel System

Parallel computing is a form of computation in which many calculations are carried out simultaneously, thus, large problems can be divided into smaller ones, which are then solved concurrently. Note that parallel system cannot be addressed as only a multi-processor design or double the number of cores on a chip design. The target should be easy to write programs that execute efficiently on highly parallel computing systems. Due to the gap of software and hardware design, we need to provide

convenient functionalities to higher levels and permit an efficient implementation at lower levels.

Over last years, several embedded solutions for parallel processing are ready available: SMP [45], ASymmetric Multi-Processing (ASMP), Network-On-Chip (NoC) [46–48], Graphics Processing Unit (GPU)[49]. Indeed, higher timing performance is achieved, but the following issues have emerged with these new embedded solutions:

1. The Taxonomy of Multiprocessors

   We argue that MPSoCs from two important branches in the taxonomy of multiprocessors: the **homogeneous** model pioneered by the Lucent Daytona (i.e.ARM MPCore [50]; Intel IXP2855 (cluster) [*Intel IXP2855 Network Processor*]; Freescale MC8126; Samdbridge Sandblaster processor [51][52] and the **heterogeneous** model pioneered by the Nokia C5, NXP Nexperia and TI OMPA (i.e. Seiko-Epson inkjet printer "Realoid" SoC; Infineon music processor; IBM Cell processor [53] multiprocessors. In 2002, Hammond et al. [54] proposed a Chip MultiProcessor (CMP) architecture before the appearance of the Lucent Daytona. Their proposed machine included eight CPU cores, each with its own first-level cache (L1) and a shared second-level (L2) cache. They used architectural simulation methods to compare their CMP architecture to simultaneous **multithreading** and **superscalar**. They found that the CMP provided higher performance on most of their benchmarks and argued that the relative simplicity of CMP hardware made it more attractive for VLSI implementation. In 2005, the Intel Core Duo processor [55] combines two enhanced Pentium M cores on a single die. The processors are relatively separate but share an L2 cache as well as power management logic. The AMD Opteron dual-core processor provides separate L2 caches for its processors but a common system request interface connects the processors to the memory controller and HyperTransport. Also the Niagra SPARC processor has eight symmetrical four-way multithreaded cores as another multi-core architecture.

2. Instruction Sets:

   Instruction sets that are designed for a particular application which are used in many embedded systems. Processor configuration is usually of two type: **Coarse-grained** structural configuration and **Fine-grained** instruction extensions. Coarse-grained structural includes the presence or absence of a variety of interfaces (bus, local memory, direct first-in first-out interfaces or accelerating block) to associated objects. Fine-grained instruction adds extra instructions directly into the datapath of the processor that are decoded in the standard way, and may even be recognized by the compiler automatically or least as manually invoked intrinsics. The architectural optimization is often in conjunction with configurable processor or coprocessors, which is supported by the CPU tools. The related CPU design tools are MIMOLA system [56], ASIP Meister [57], LISA [58], XPRES tool [59] etc.

3. Interconnections:

Most interconnect choices are based on conventional bus concepts. Thus, the two best known SoC buses are **ARM AMBA** (ARM processor and LEON3 processor) and **IBM CoreConnect** (PowerPC and IBM POWER processors). It has long been predicted that the current bus-based architectures will run out of performance and desirable to achieve the required on-chip communications and bandwidth, as SoCs grow in complexity with more and more processing blocks. [60].

4. Alternative Architectures:

   The search of alternative architectures has led to the concept of **NoC** and **GPU**. For a good survey, see the book edited by De Micheli and Benini [61]. The key idea behind a NoC design is to use a hierarchical network with routers to allow packets to flow more efficiently between originators and targets, in order to provide additional communications resources so that multiple communications channels can be simultaneously operating. Sonics Silicon Backplane was the first commercial NoC [62], which offers TDMA-style interconnection network. Today, several other vendors provide NoCs, including Arteris [63] and Silistix [64] etc.. Shuai et al. [49] also proves that GPU could accelerate the routing processing by one order of magnitude.

5. Inter-Processor Communication:

   Inter-Process Communication (IPC) is a set of methods for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network. IPC methods are divided into methods for message passing, synchronization, shared L2 memory (Intel core 2 Duo, Xbox 360's custom PowerPC), and remote procedure calls. Regarding inter-processor communication in a multi-core design, Chin et al. [65] applies the Message Passing Interface (MPI) and Open Multi-Processing techniques for parallel computation implementation. This parallel method could put the mutli-cores microprocessors to be efficiently used in GNSS signal acquisition and tracking processes.

6. Harware/Software Co-design:

   Hardware/Software
   codesign was developed in the early 1990s. Hardware/Software cosynthesis can be used to explore the design space of heterogeneous multiprocessors. Cosyma [66], Vulcan [67], and system of Kalavade and Lee [68] were early influential cosynthesis systems, with each taking a very different approach to solve all the complex interactions due to the many degrees of freedom in hardware/software codesigns. Cosyma's heuristic moved operations from software to hardware; Vulcan started with all operations in hardware and moved some to softwares; Kalavade and Lee used iterative performance improvement and cost reduction steps. Cost estimation is one very important problem in cosynthesis, because area, performance, and power must be estimated both accurately and quickly.

7. Software Methods:

Software methods can be used to find and eliminate such conflicts but only at noticeable cost [69]. A variety of algorithms and methodologies have been developed to **optimize the memory system** by the behavior of software [70]. De Greef et al. [71] developed an influential methodology for memory system optimization in multimedia systems. A great work at the IMEC, such as that by Franssen el al. [72], De Greed et al. [71], and Masselos et al. [73], developed algorithms for data transfer and storage management. Panda el al. [74] developed algorithms that place data in memory to reduce cache conflicts and place arrays accessed by inner loops. Kandemir et al. [75] combined data and loop transformations to optimize the behavior of a program in cache.

8. Memory Allocation:

   Furthermore, access to any memory location in a block may be sufficient to disrupt real-time access to a few specialized locations in that block. However, if that memory block is addressable only by certain processors, then programmers can much more easily determine what tasks are accessing the locations to ensure proper real-time responses, here commonly concern about L2 cache design [76] [77], manage the cache hierarchy [78] or compress the main memory [79]. Also scratch pad memories have been proposed as alternatives to address-mapped caches. A scratch pad memory occupies the same level of the memory hierarchy as a cache does, but its contents are determined by software, giving more control of access times to the program. Panda et al. [80] developed algorithms to allocate variables to the scratchpad. They defined a series of metrics, including variable and interference access counts, to analyze the behavior of arrays that have intersecting lifetime.

2.3.2 Parallelism

The promise of parallelism has fascinated researchers for at least three decades. Across many technologies, bandwidth improves by at least the square of the improvement in latency. Therefore, increasing parallelism is the primary method of improving processor performance. In general, the programming models should support a wide range of parallelism: Bit-Level Parallelism (BLP), Instruction-Level Parallelism (ILP), Data-Level Parallelism (DLP), Task-Level Parallelism (TLP), whereas the uni-processor only relies on available ILP for high performance, which is limited.

1. Bit-Level Parallelism (BLP)

   Historically, 4-bit microprocessors were replaced by 8-bit, then 16-bit, this trend generally came to an end with the introduction of 32-bit processors, which has been a standard in general-purpose computing for two decades. Not until recently, with the advent of x86-64 architectures, 64-bit processors become common. Thus increasing the word size of the processor can reduce the number of instructions per cycle. For example, an 8-bit processor must add two 16-bit integers, the processor must first add the 8 lower-order bits from each

integer by the standard addition instruction, then add the 8 higher-order bits by an add-with-carry instruction, and then add the carry bit from the lower order addition. Thus, an 8-bit processor requires two instructions to complete this single operation, where a 16-bit processor would be able to complete the operation with a single instruction.

2. Instruction-Level Parallelism (ILP)

Figure 2.5: Canonical five-stage pipeline in a RISC machine. (IF = Instruction Fetch, ID = Instruction Decode, EX = EXecute, MEM = Memory Access, WB = Write Back)

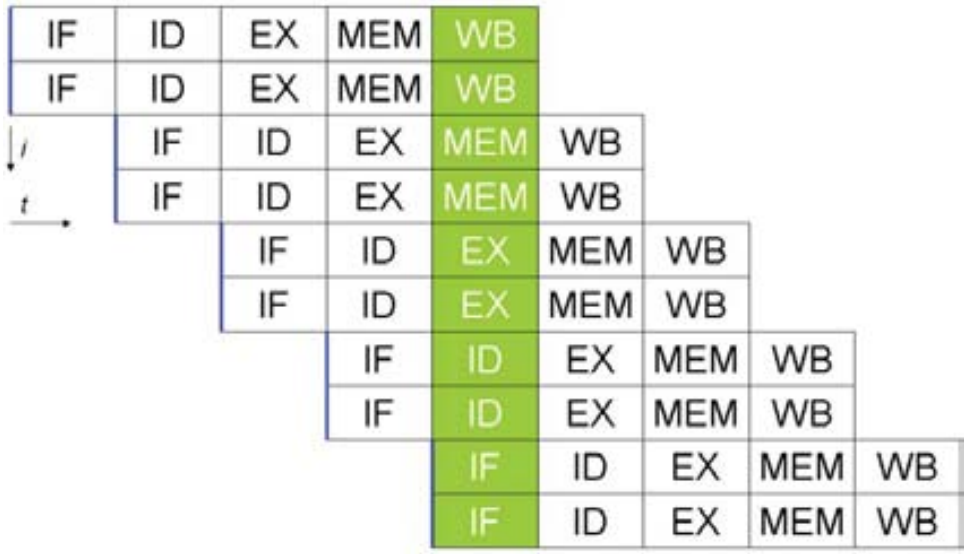


Figure 2.6: A five-stage pipelined superscalar processor, capable of issuing two instructions per cycle. It can have two instructions in each stage of the pipeline, for a total of up to 10 instructions being simultaneously executed.

Modern processors have multi-stage pipelines of instruction. Each stage corresponds to a different action of processor. A processor with an N-stage pipeline can have up to N different instructions performing at different stages. The canonical example of five stages pipelined processor is a RISC processor with the stages of fetch, decode, execute, memory access, and write back shown in Fig.2.5. In addition to multi-stage pipeline, some processors can execute more than one instruction during a clock cycle by simultaneously dispatching

multiple instructions to redundant functional units on the processor. These are known as superscalar processors shown in Fig.2.6. However the complexity of the dispatcher and the associated dependency checking logic limit the amount of ILP.

3. Data-Level Parallelism (DLP)

   Data parallelism is parallelism inherent in program loops (e.g. BogoMIPS), which focuses on distributing the data across different computing nodes to be processed in parallel. As the size of a problem getting bigger, we could use BogoMips to present the performance of CPUs as well as the data parallelism. BogoMips is an unscientific measurement of CPU speed made by the Linux kernel when it boots, to calibrate an internal busy-loop.

   Conventional programs are written by assuming the sequential execution model, under this model instructions execute one after the other automatically. However, parallel execution of multiple instructions may induce the dependency problems. Executing multiple instructions without considering related dependencies may cause the wrong results, namely **hazards**. For example, consider the following pseudo-code that computes the first few Fibonacci numbers. This loop cannot be parallelized because *c* depends on itself, *a* and *b*, which are computed in each loop iteration. Since each iteration depends on the result of the previous one, they cannot be performed in parallel.

```
1 a :=1; b :=0; c :=1;
2
3 do:
4    c := a + b;
5    b := a;
6    a := c;
7
8 while{c<10}
```

4. Task-Level Parallelism (TLP)

   Applications are often classified according to how often their subtasks need to synchronize or communicate with each other. An application exhibits fine-grained parallelism if its subtasks must communicate many times per second; it exhibits coarse-grained parallelism if they do not communicate many times per second. The ratio between computation and communication is known as **granularity**. Large number of small tasks (short computation cycles) are executed between long communication cycles defined by **fine-grain parallelism** (Computation $<$ Communication). Typified by small number of large tasks (long computation cycles) are executed between short communication cycles defined by **coarse-grain parallelism** (Computation $>$ Communication).

   For example, in the reconfigurable computing, granularity refers to the data path width, 1-bit wide PEs like the Configurable Logic Blocks (CLBs) in an FPGA called fine-grained computing; 32-bit wide resources, like microprocessor CPUs or data-stream-driven Data-Path Units (DPUs) called coarse-grained computing.

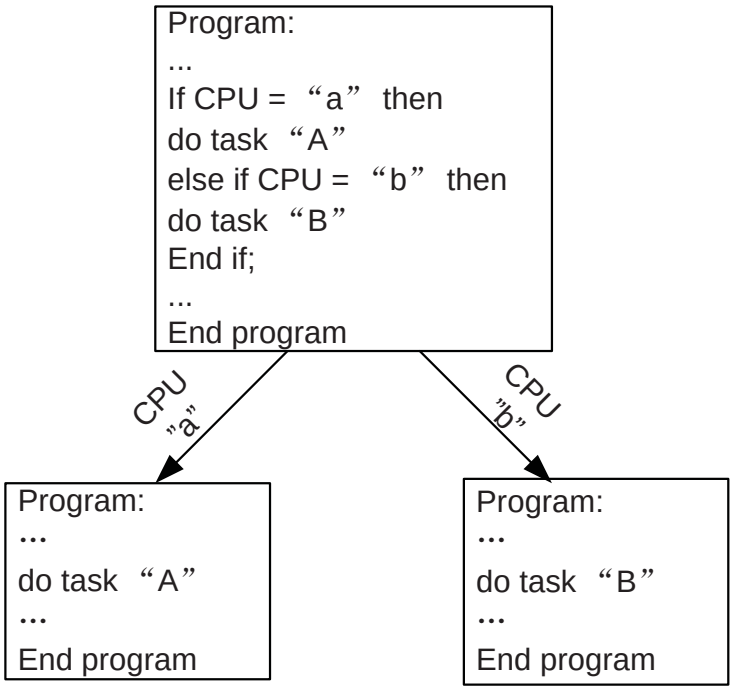


Figure 2.7: Task-level parallelism.

Nollet et al. [81] introduces how the fine-grain configuration hierarchy improves the task assignment performance in multimedia applications.

Task parallelism focuses on distributing execution processes (threads) across different parallel computing nodes. Contrasts with data parallelism, where the same operation is performed on the same or different sets of data. Task parallelism is a form of parallelization of computer code across multiple processors in parallel computing environments. For example, the program is to do the net total task (A+B) as shown in Fig.2.7. If we write the code as launch it on a 2-processor system, then both CPUs (a,b) could execute separate code blocks simultaneously performing different tasks simultaneously.

2.3.3 Parallel Architecture

The parallel architecture can be roughly classified into multi-core and multi-processor architecture, which has multiple Processing Elements (PEs) within a single machineor clusters. And these PEs are the most efficient in Million Instructions Per Second (MIPS). Our view focuses on the evolutionary approach to parallel hardware and software work on two to ten processors system. The parallel architecture can be seen as the extension of the conventional computer architecture to address issues of communication and cooperation among PEs.

Multiprocessing systems are less complex than multi-core systems, because they are essential single chip CPUs connected together. CPUs have their own caches or

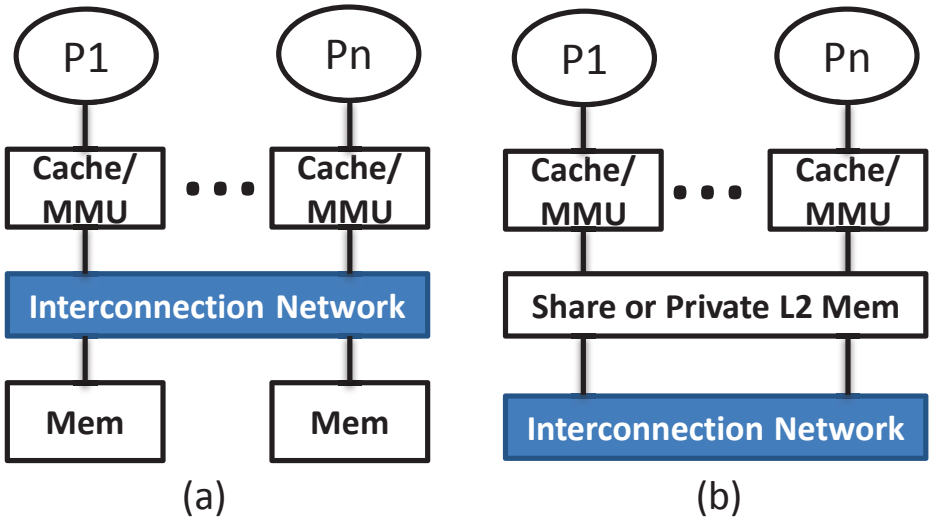


Figure 2.8: Multiprocessing systems: a) The multiprocessor with local cache or Memory Management Unit (MMU), but shared memory by long interconnects; b) The multi-core processors with local cache or MMU, but private or shared L2 memory by short interconnects.

Memory Management Units (MMUs), but shared the main memory. Typically it is named SMP. It is easy to built up the higher the Operation System (OS) level and SW programming. Fig.2.8-a shows that the multiprocessor system has local cache and MMU with long-interconnects. Its communication occurs through a shared address space via loads and stores. The challenge of SMP is that the shared resource and scheduling issues may reduce the system gain.

Multi-core systems are a family of processors that contain numbers of multiple CPUs on a single chip, such as 2, 4, and 8. Fig.2.8-b shows that the multi-core processors with private L1 caches or MMUs but with short interconnects of memories. Its communication occurs by explicitly passing messages among the processors: message-passing multiprocessors, shared L2 caches, or shared memory inter-core communication methods. The challenge with multi-core processors is in the area of software development. The speed-up performance is directly related to how parallel the source code of an application was written through multi-threading.

There are two models of the communication and memory architecture in parallel systems: heterogeneous architecture and homogeneous architecture. The important motivation of design new parallel architectures is the real-time performance. **Heterogeneous architectures**, although they are harder to program, it can provide improved real-time behavior by reducing conflicts among processing elements and tasks [82]. For example, considering a shared memory multiprocessor in which all CPUs have access to all parts of memory. The hardware cannot directly limits accesses to a particular memory location, therefore, noncritical accesses from on processor may conflict with critical accesses from another [83]. In [84], the authors indicate that the memory architecture dictates most of the data traffic flow in a design, which in turn influences the design of the communication architecture. The

main challenge will be the data migration and recomputation technique design [85], instruction cache locking issue [86] etc.. Different applications have different data flows that suggest multiple different types of architectures, especially for scientific computing [87]. **The homogeneous architectural** style is generally used for data-parallel systems, in which the same algorithm is applied to several independent data streams, i.e. wireless base stations.

2.3.4 Parallel Algorithm

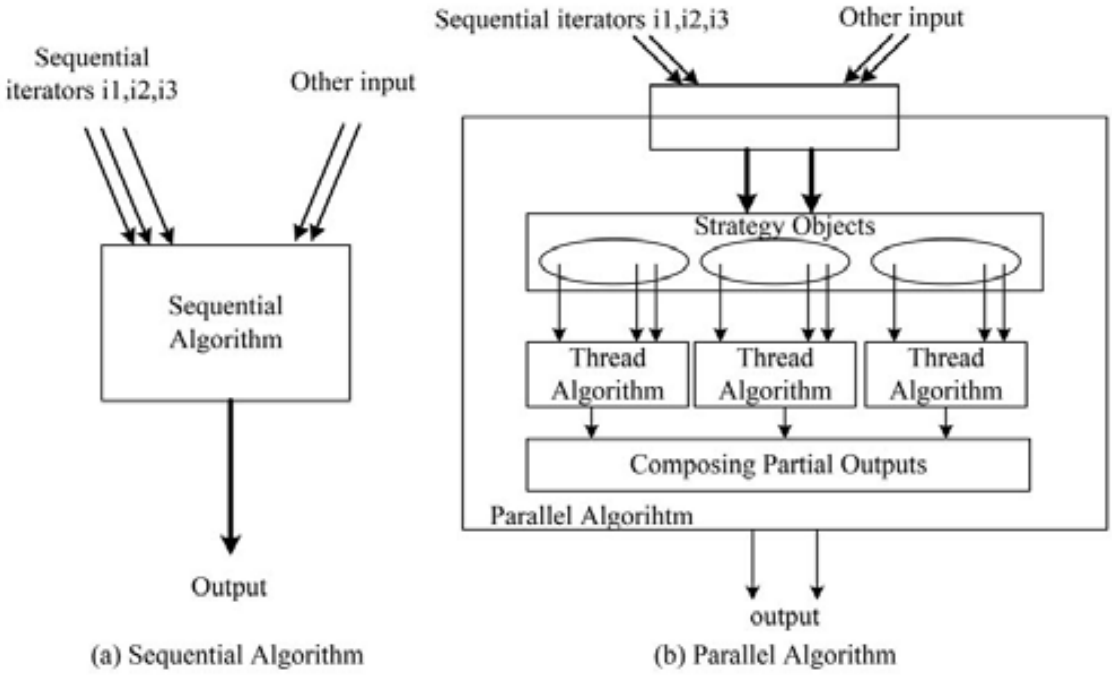


Figure 2.9: Sequential algorithm vs parallel algorithm.

The parallel algorithm can execute an application at a time on many different PEs, and then collect the result back together. Designing the parallel algorithm, a given problem may be decomposed in many different ways and orders. Typical forms of sequential algorithms is shown in Fig.2.9-a: The sequential algorithm takes in 3 iterators and other input parameters, traverses the iterators sequentially and produces an output. A typical parallel algorithm is shown in Fig.2.9-b: Strategy objects are used to convert the sequential iterators to per-thread parallel iterators which are used in the partial algorithms by each thread. The results computed by each of the threads in the partial algorithms are composed to produce the final output.

Michael J. Flynn [*Flynn*] created one of the earliest classification systems for parallel algorithms, known as Flynn's taxonomy. Flynn classified the parallel algorithms by whether they were operating by a single set or multiple sets of instructions, whether or not those instructions were using a single or multiple sets of data as shown in Table 2.II.

Table 2.II: Flynn's taxonomy.

Data \ Instruction	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

2.3.5 Parallel Programming Model

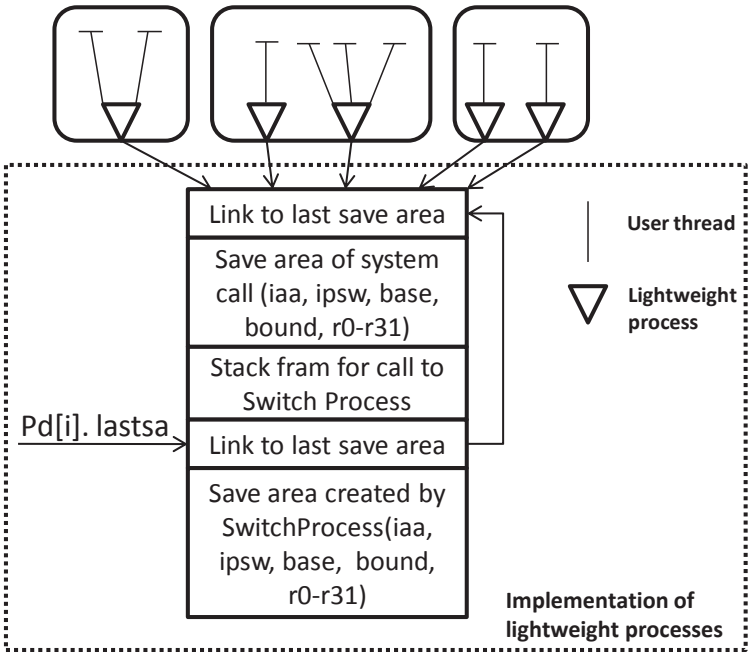


Figure 2.10: Thread, process and OS.

The parallel programming model is used to abstract HW/SW interfaces during the high level specification of the application software. The software is adapted to the existing multiprocessor platforms by a low level hardware layer that implements the programming model [82]. There are three types of parallel programming models:

- Shared address space: communicate via memory, e.g. load, store, atomic swap.
- Message passing: send and receive messages, e.g., send, receive library calls.
- Data Parallel: several agents operate on several data sets simultaneously.

Synchronization among the different subtasks is typically one of the greatest barriers in the parallel programming design. As shown in Fig. 2.10, in two multi-threading design, we separate resource holding and dispatch ability into separate concepts: process and thread. The process has several threads inside a single process, and several threads can be executed in the same code at the same time. OS is responsible for ensuring synchronization of the data that needs to be modified

Table 2.III: Comparison of OPENMP and POSIX.

Application Programming Interface(API)	OPENMP	POSIX(Pthreads)
Hybird System	SMP	SMP
Compiler	Omni-1.6 (omcc -g)	gcc -pthread
Parallelism Language	Openmp	Pthread
Algorithm	Fork-Join Model	Worst-case memory-to-cpu bandwidth

by multiple threads or processes. However, synchronization delay occurs in any application programs, if the OS blocks on one of them, then they are all blocked. This inspire us using parallel programming to analysis the system synchronization delay. Programming model is made up of the languages and libraries. Based on the different Application Programming Interface (API), nowadays, there are two popular multi-threading technologies: OPENMP [*openmp*] and POSIX [*pthread*], the basic characteristics of OPENMP and POSIX technologies as shown in Table 2.III.

There are many other critical issues in the frame of parallel computing research. As shown in Fig.2.11, the programming model is a bridge between a system developer's natural model of an application and an implementation of that application on available hardware. However, based on our GNSS-R applications, we need to answer the following questions in the parallel system design.

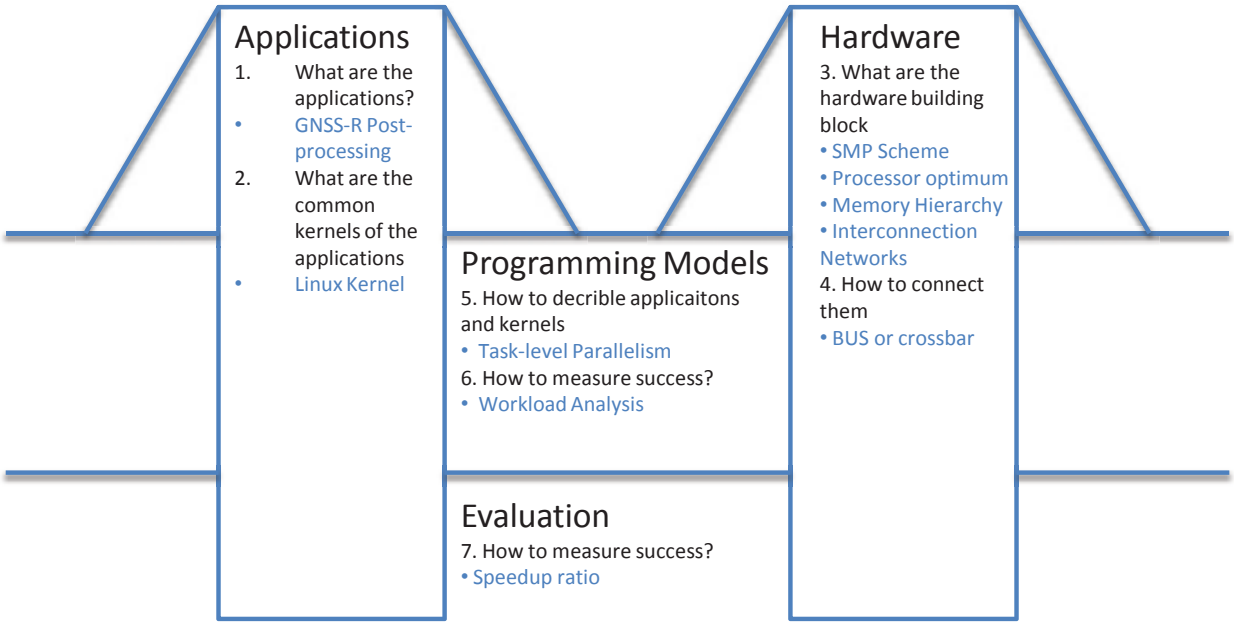


Figure 2.11: A view from Berkeley: seven critical questions for 21st Century parallel computing. (Adapted from Krste Asanovic et al. [88])

What is the workload analysis for parallel computing in the left tower? If the programming models, compilers and OS can help span the gap between applications

and right tower? We will follow our research line to investigate each issues in the following Chapters.

Bibliography

[1] M. Martín-Neria, "A passive reflectometry and interferometry system (paris):application to ocean altimetry," *ESA Journal*, vol. 17, no. 4, pp. 331–355, 1993.

[2] O. Nogués-Correig, E. Cardellach-Galí, J. Sanz-Camderrós, and A. Rius, "A gps-reflections receiver that computes doppler-delay maps in real time," *IEEE Transactions on Geoscience and Remote sensing*, vol. 45, no. 1, pp. 156–174, 2007.

[3] Spilker, Parkinson, Axelrad, and Enge, *Global Positioning System: Theory and Applications*, ser. V-164. Inst. Aeronaut. Astronaut., Washington, DC., 1996, vol. II.

[4] P. Misra and P. Enge, *Global Positioning System: Signals, Measurements, and Performance*. Ganga-Jamuna Press, Lincoln, Massachussets, 2006.

[5] E. Cardellach, F. Fabra, O. Nogués-Correig, S. Oliveras, S. Ribó, and A. Rius, "Gnss-r ground-based and airborne campaigns for ocean, land, ice and snow techniques: application to the gold-rtr datasets," *RADIO SCIENCE*, 2011.

[6] M. Martín-Neira, M. Caparrini, J. Font-Rosselló, S. Lannelongue, and C. S. Vallmitjana, "The paris concept: an experimental demonstrationof sea surface altimetry using gps reflected signals," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 39, no. 1, pp. 142–149, 2001.

[7] S. T. Lowe, C. Zuffada, Y. Chao, P. Kroger, L. E. Young, and J. L. LaBrecque, "5-cm-precision aircraft ocean altimetry using gps reflections," *GEOPHYSICAL RESEARCH LETTERS*, vol. 29, no. 10, May 2002.

[8] G. Ruffini, F. Soulat, M. Caparrini, O. Germain, and M. Martín-Neira, "The eddy experiment: Accurate gnss-r ocean altimetry from low altitude aircraft," *GEOPHYSICAL RESEARCH LETTERS*, vol. 31, no. p.L12306, 2004.

[9] G. A. Hajj and C. Zuffada, "Theoretical description of a bistatic system for ocean altimetry using the gps signal," *Radio Science*, vol. 38, no. 5, p. 1089, 2002.

[10] A. Rius, E. Cardellach, and M. Martin-Neira, "Altimetric analysis of the sea-surface gps-reflected signals," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 48, no. 4, pp. 2119 – 2127, 2010.

[11] E. Cardellach, C. O. Ao, M. de la Torre Juárez, and G. A. Hajj, "Carrier phase delay altimetry with gps-reflection/occultation interferometry from low earth orbiters," *GEOPHYSICAL RESEARCH LETTERS*, vol. 31, no. 4, p. L10402, 2004.

[12] A. Helm, G. Beyerle, and M. Nitschke, "Detection of coherent reflections with gps bipath interferometry," *Canadian Journal of Remote Sensing (2004)*, no. 2000, p. 11, 2004.

[13] R. N. Treuhaft, S. T. Lowe, C. Zuffada, and Y. Chao, "2-cm gps altimetry over crater lake," *GEOPHYSICAL RESEARCH LETTERS*, vol. 28, no. 23, pp. 4343 – 4346, 2001.

[14] F. Fabra, E. Cardellach, A. Rius, S. Ribó, S. Oliveras, O. Nogués-Correig, M. Belmonte, M. Semmling, and S. D'Addio, "Phase altimetry with dual polarization gnss-r over sea ice," *IEEE Trans. Geosc. Remote Sensing (submitted)*, 2011.

[15] A. M. Semmling, G. Beyerle, R. Stosius, G. Dick, J. Wickert, F. Fabra, E. Cardellach, S. Ribó, A. Rius, A. Helm, S. B. Yudanov, and S. d'Addio, "Detection of arctic ocean tides using interferometric gnss-r signals," *GEOPHYSICAL RESEARCH LETTERS*, vol. 38, no. 4, p. L04103, 2011.

[16] J. Garrison, A. Komjathy, V. Zavorotny, and S. Katzberg, "Wind speed measurement using forward scattered gps signals," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 40, no. 1, pp. 50 – 65, 2002.

[17] E. Cardellach, G. Ruffini, D. Pino, A. Rius, and A. Komjathy, "Mediterranean balloon experiment: ocean wind speed sensing from the stratosphere, using gps reflections," *Remote Sensing of Environment*, vol. 88, no. 3, pp. 351 – 362, 2003.

[18] A. Komjathy, M. Armatys, D. Masters, P. Axelrad, V.U.Zavorotny, and S. Katzberg, "Retrieval of ocean surface wind speed and wind direction using reflected GPS signals," *Journal of Atmospheric and Oceanic Technology*, vol. 21, no. 3, pp. 515 – 526, 2004.

[19] G. Ruffini, F. Soulat, M. Caparrini, O. Germain, and M. Martin-Neira, "The gnss-r eddy experiment i: Altimetry from low altitude aircraft," *GEOPHYSICAL RESEARCH LETTERS*, vol. 31, no. 4, p. L12306, 2004.

[20] V. Zavorotny and A. Voronovich, "Scattering of gps signals from the ocean with wind remote sensing application," *IEEE Transactions on Geoscience and remote sensing*, vol. 38, no. 2, pp. 951 – 964, 2000.

[21] T. Elfouhaily, D. Thompson, and L. Linstrom, "Delay-doppler analysis of bistatically reflected signals from the ocean surface: theory and application," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 40, no. 3, pp. 560 – 573, 2002.

[22] J. Marchan-Hernandez, N. Rodriguez-Alvarez, A. Camps, X. Bosch-Lluis, I. Ramos-Perez, and E. Valencia, "Correction of the sea state impact in the l-band brightness temperature by means of delay-doppler maps of global navigation satellite signals reflected over the sea surface," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 46, no. 10, pp. 2914 – 2923, 2008.

[23] E. Valencia, J. Marchan-Hernandez, A. Camps, N. Rodriguez-Alvarez, J. Tarongi, M. Piles, I. Ramos-Perez, X. Bosch-Lluis, M. Vall-llossera, and P.Ferre, "Experimental relationship between the sea brightness temperature changes and the gnss-r delay-doppler maps: Preliminary results of the albatross field experiments," *IEEE Proc. IGARSS*, pp. 741–744, 2009.

[24] E. Cardellach and A. Rius, "A new technique to sense non-gaussian features of the sea surface from l-band bi-static gnss reflections," *Journal of Remote Sensing of Environment*, vol. 112, no. 6, pp. 2927–2937, June 2008.

[25] F. Soulat, M. Caparrini, O. Germain, P. Lopez-Dekker, M. Taani, and G. Ruffini, "Sea state monitoring using coastal gnss-r," *GEOPHYSICAL RESEARCH LETTERS*, vol. 31, no. 4, p. L21303, 2004.

[26] E. Valencia, A. Camps, J. Marchan-Hernandez, N. Rodriguez-Alvarez, I. Ramos-Perez, and X. Bosch-Lluis, "Experimental determination of the sea correlation time using gnss-r coherent data," *IEEE Geoscience and Remote Sensing Letters*, vol. 7, no. 4, pp. 675 – 679, 2010.

[27] E. Cardellach, S. Ribó, and A. Rius, "Technical note on polarimetric phase interferometry (popi)," *physics/0606099*, 2006.

[28] D. Masters, P. Axelrad, and S. Katzberg, "Initial results of land-reflected gps bistatic radar measurements in smex02," *Journal of Remote Sensing of Environment*, vol. 92, no. 4, pp. 507–520, 2004.

[29] D. Manandhar, R. Shibasaki, and H. Torimoto, "Prototype software-based receiver for remote sensing using reflected gps signals," in *Proc. ION GNSS 19th International Technical Meeting of the Satellite Division*, pp. 643 – 652.

[30] S. J. Katzberg, O. Torres, M. S. Grant, and D. Masters, "Utilizing calibrated gps reflected signals to estimate soil reflectivity and dielectric constant: Results from smex02," *Journal of Remote Sensing of Environment*, vol. 100, no. 1, pp. 17 – 28, 2006.

[31] E. Cardellach, F. Fabra, O. Nogués-Correig, S. Oliveras, S.Ribó, and A. Rius, "From greenland to antarctica: Csic/ieec results on sea-ice, dry-snow, soil-moisture and ocean gnss reflections," in *Proceedings of 2nd International Colloquium - Scientific and Fundamental Aspects of the Galileo Programme*.

[32] V. Zavorotny and A. Voronovich, "Bistatic GPS signal reflections at various polarizations from rough land surface with moisture content," in *Proc. IEEE IGARSS*, vol. 7, pp. 2852–2854.

[33] V. Zavorotny, D. Masters, A. Gasiewski, B. Bartram, S. Katzberg, P. Axelrad, and R. Zamora, "Seasonal polarimetric measurements of soil moisture using tower-based GPS bistatic radar," in *Proceedings of Geoscience and Remote Sensing Symposium 2003*, 2003, pp. 515 – 526.

[34] L. Chung, Jyh-Ching, Ching-Lang, Chia-Chyang, Ping-Ya, and Ching-Liang, "Stream soil moisture estimation by reflected GPS signals with ground truth measurements," *IEEE Transactions on Instrumentation and Measurements*, vol. 58, no. 3, pp. 730 – 737, 2009.

[35] A. Komjathy, J. Maslanik, V. Zavorotny, P. Axelrad, and S. Katzberg, "Sea ice remote sensing using surface reflected gps signals," in *Proceedings of Geoscience and Remote Sensing Symposium*, vol. 7, pp. 2855 – 2857.

[36] M. Rivas, J. Maslanik, and P. Axelrad, "Bistatic scattering of gps signals off arctic sea ice," *IEEE Transaction of Geoscience and Remote Sensing*, vol. 48, no. 3, pp. 1548 – 1553, 2010.

[37] A. Komjathy, M. Armatys, D. Masters, P. Axelrad, V.U.Zavorotny, and S. Katzberg, "A novel technique for characterizing the thickness of first-year sea ice with the gps reflected signal," *JPL TRS 1992*.

[38] M. Wiehl, R. Légrésy, and R. Dietrich, "Potential of reflected GNSS signals for ice sheet remote sensing," *Progress in electromagnetics research*, vol. 40, pp. 177–205.

[39] N. Rodriguez-Alvarez, A. Camps, M. Vall-llossera, X. Bosch-Lluis, A. Monerris, I. Ramos-Perez, E. Valencia, J. Marchan-Hernandez, J. Martinez-Fernandez, G. Baroncini-Turricchia, C. Pérez-Gutiérrez, and N. N. Sánchez, "Land geophysical parameters retrieval using the interference pattern gnss-r technique," *IEEE Trans. Geosc. Remote Sensing*, vol. 49, no. 1, pp. 71 – 84, 2010.

[40] A. Rius, J. M. Aparicio, E. Cardellach, M. Martín-Neira, and B. Chapron, "Sea surface state measured using GPS reflected signals," *Geophys. Res. Lett.*, vol. 29, no. 4, p. 2122, 2002.

[41] S. Gleason, "Remote sensing of ocean, ice and land surfaces using bistatically scattered gnss signals from low earth orbit," Ph.D. thesis, University of Surrey.

[42] S. Delwart, C. Bouzinac, P. Wursteisen, M. Berger, M. Drinkwater, M. Martín-Neira, and Y. Kerr, "SMOS validation and the CoSMOS campaigns," *IEEE Trans. Geosc. Remote Sensing*, vol. 46, no. 3, pp. 695–704, 2008.

[43] S. Blanch and A. Aguasca, "Sea water dielectric permittivity model from measurements at l band," *IEEE*, 2004.

[44] J. Wu, S. Wu, G. Hajj, W. Bertiger, and S. Lichten, "Effects of antenna orientation on GPS carrier phase," *Manuscripta Geodaetica*, vol. 18, no. 2, pp. 91–98, 1993.

[45] H. Shen and F. Petrot, "Novel task migration framework on configurable heterogeneous mpsoc platforms," in *Proceedings of the Asia and South Pacific Design Automation Conference*, Pacifico Yokohama, Yokohama, Japan, Jan. 2009, pp. 733–738.

[46] S. E. Lee, J. H. Bahn, Y. S. Yang, and N. Bagherzadeh, "A generic network interface architecture for a networked processor array (nepa)," *ARCS 2008*, vol. 4934, pp. 247 – 260, 2008.

[47] A. Kumar, A. Hansson, J. Huisken, and H. Corporaal, "An fpga design flow for reconfigurable network-based multi-processor systems on chip," in *Design, Automation Test in Europe Conference (DATE07)*, Nice, France, 2007, pp. 1 – 6.

[48] P. Zhou, B. Zhao, Y. Du, Y. Xu, Y. Zhang, J. Yang, and L. Zhao, "Frequent value compression in packet-based noc architectures," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC 2009)*, Yolohama, Japan, Jan. 2009, pp. 13 – 18.

[49] S. Mu, X. Zhang, N. Zhang, J. Lu, Y. S. Deng, and S. Zhang, "Ip routing processing with graphic processors," in *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE10)*. Dresden, Germany: IEEE, 2010, pp. 93–98.

[50] J. Goodacre and A. Sloss, "Parallelism and the arm instruction set architecture," *IEEE Journal of Computer*, vol. 38, no. 7, pp. 42 – 50, July 2005.

[51] J. Glossner, D. Iancu, J. Lu, E. Hokenek, and M. Moudgill, "A software-defined communications baseband design," *IEEE Journal of Communications Magazine*, vol. 41, no. 1, pp. 120 – 129, Jan. 2003.

[52] J. Glossner, M. Moudgill, D. Iancu, G. Nacer, S. Jinturkar, S. Stanley, M. Samori, T. Raja, and M. Schulte, "The sandbridge convergence platform," 2005.

[53] M. Kistler, M. Perrone, and F. Petrini, "Cell multiprocessor communication network: Built for speed," *IEEE Journal of Micro*, vol. 26, no. 3, pp. 10 – 24, June 2006.

[54] L. Hammond, B. A. Nayfeh, and K. Olukotun, "A single-chip multiprocessor," *IEEE Journal of Computer*, vol. 30, no. 9, pp. 79 – 85, Aug. 2002.

[55] S. Gochman, A. Mendelson, A. Naveh, and E. Rotem, "Introduction to intel core duo processor architecture," *Intel Technol.J.*, vol. 10, no. 2, pp. 89 – 97, May 2006.

[56] P. Marwedel, "The mimola design system: Tools for the design of digital processors," in *Proceedings of Design Automation, 1984*, 1984, pp. 587 – 594.

[57] S. Kobayashi, K. Mita, Y. Takeuchi, and M. Imai, "Rapid prototyping of jpeg encoder using the asip development system: Peas-iii," in *Proceedings of Multimedia and Expo 2003*, 2003, pp. 149–152.

[58] A. Hoffmann, T. Kogel, A. Nohl, G. Braun, O. Schliebusch, O. Wahlen, A. Wieferink, and H. Meyr, "A novel methodology for the design of application-specific instruction-set processors (asips) using a machine description language," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 11, pp. 1338 – 1354, Nov. 2001.

[59] D. Goodwin and D. Petkov, "Automatic generation of application specific processors," in *Proceedings of compilers, architecture and synthesis for embedded systems (CASES03)*, 2003, pp. 137 – 147.

[60] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of Design Automation Conference*, 2001, pp. 684 – 690.

[61] G. D. Micheli and L. Benini, *Networks on chips: technology and tools*, San Francisco, CA: Morgan Kaufmann, July 20 2006.

[62] D. Wingard, "Micronetwork-based integration for socs," in *Proceedings of Design Automation Conference*, 2001, pp. 673 – 678.

[63] A. Fanet, "Noc: The arch key of ip integration methofology," in *Proceedings of MPSoC Symp*, 2005.

[64] B. Bailey, G. Martin, and A. Piziali, "Esl design and verification: A prescription for electronic system level methodology," San Francisco, CA: Morgan Kaufmann, 2007.

[65] C. C. Sun and S. S. Jan, "Gnss signal acquisition and tracking using a parallel approach," in *Proceedings of Position, Location and Navigation Symposium, 2008 IEEE ION*. Monterey, CA, USA: IEEE, 2008, pp. 1332–1340.

[66] R. Ernst, J. Henkel, and T. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Transactions on Design & Test of Computers*, vol. 10, no. 4, pp. 64 – 75, Dec. 1993.

[67] R. Gupta and G. D. Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Transactions on Design & Test of Computers*, vol. 10, no. 3, pp. 29 – 41, Sept. 2002.

[68] A. Kalavade and E. Lee, "The extended partitioning problem: hardware/software mapping and implementation-bin selection," in *Proceedings of Rapid System Prototyping*, June 1995, pp. 12 – 18.

[69] T. C. Srimat and R. Anand, "Best-effort computing: Re-thinking parallel software and hardware," in *Proceedings of Design Automation Conference (DAC2010)*, Anaheim, CA, USA, 2010, pp. 865 – 870.

[70] W. Wolf and M. Kandemir, "Memory system optimization of embedded software," *Proceedings of IEEE*, vol. 91, no. 1, pp. 165 – 182, Jan. 2003.

[71] E. D. Greef, F. Catthoor, and H. D. Man, "Memory organization for video algorithms on programmable signal processors," in *Proceedings of Computer Design: VLSI in Computers and Processors (ICCD)*, Oct. 1995, pp. 552 – 558.

[72] F. Franssen, I. Nachtergaele, H. Samsom, F. Catthoor, and H. D. Man, "Control flow optimization for fast system simulation and storage minimization," in *Proceedings of European Design and Test Conference*, 1994, pp. 20 – 24.

[73] K. Masselos, F. Catthoor, C. Goutis, and H. D. Man, "A performance oriented use methodology of power optimizing code transformations for multimedia applicatins realized on programmable multimedia processors," in *Proceedings of Signal Processing Systems*, Aug. 1999, pp. 261 – 271.

[74] P. Panda, N. Dutt, and A. Nicolau, "Memory organization for improved data cache performance in embedded processors," in *Proceedings of System Synthesis*, Nov. 1996, pp. 90 – 96.

[75] M. Kandemir, J. Ramanujam, and A. Choudhary, "Improving cache locality by a combination of loop and data transformations," *IEEE Transactions on Computers*, vol. 48, no. 2, pp. 159 – 167, Feb. 1999.

[76] J. Shin, B. Petrick, M. Singh, and A. Leon, "Design and implementation of an embedded 512kb level-2 cache subsystem," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 9, pp. 1815 – 1820, 2005.

[77] A. Asaduzzaman, F. N. Sibai, and M. Rani, "Impact of level-2 cache sharing on the performance and power requirements of homogeneous multicore embedded systems," *Journal of Microprocessors & Microsystems*, vol. 33, no. 5, pp. 388 – 397, 2009.

[78] E. Speight, H. Shafi, L. Zhang, and R. Rajamony, "Adaptive mechanisms and policies for managing cache hierarchies in chip multiprocessors," in *International Symposium on Computer Architecture (ISCA '05)*, Madison, Wisconsin, USA, 2005, pp. 346 – 357.

[79] B. Abali, X. Shen, H. Franke, D. Poff, and T. Smith, "Hardware compressed main memory: operating system support and performance evaluation," *IEEE Transactions on Computers*, vol. 50, no. 11, pp. 1219 – 1226, 2001.

[80] P.R.Panda, N.D.Dutt, and A. Nicolau, "On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems," *ACM Transactions on Design Automatic Embedded System*, vol. 5, no. 3, pp. 682 – 704, July 2000.

[81] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal, "Run-time management of a mpsoc containing fpga fabric tiles," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 1, pp. 24 – 33, 2008.

[82] K. Lobna, B. Aimen, G. Marius, F. Anne-Marie, P. Fréderic, and J. Ahmed-Amine, "Parallel programming of multi-processor soc: a hw-sw interface perspective," *International Journal of Parallel Programming*, vol. 36, no. 1, pp. 68 – 92, 2008.

[83] Y. Chenjie and P. Peter, "Off-chip memory bandwidth minimization through cache partitioning for multi-core platforms," in *Proceedings of Design Automation Conference (DAC2010)*, Anaheim, CA, USA, 2010, pp. 132 – 137.

[84] S. Pasricha and N. D. Dutt, "A framework for cosynthesis of memory and communication architectures for mpsoc," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 3, pp. 408 – 420, 2007.

[85] H. Jingtong, X. C. Jason, T. Wei-Che, H. Yi, Q. Meikang, and H. M. S. Edwin, "Reducing write activities on non-volatile memories in embedded cmps via data migration and recomputation," in *Proceedings of Design Automation Conference (DAC2010)*, Anaheim, CA, USA, 2010, pp. 350 – 355.

[86] Y. Liang and T. Mitra, "Instruction cache locking using temporal reuse profile," in *Proceedings of Design Automation Conference (DAC2010)*, Anaheim, CA, USA, 2010, pp. 344 – 349.

[87] F. Song, S. Moore, and J. Dongarra, "L2 cache modeling for scientific applications on chip multi-processors," in *Proceedings of Parallel Processing (ICPP 2007)*, XiAn, China, 2007, pp. 51 – 59.

[88] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2006-183, December 18 2006.

Web Reference

[*GOLD-RTR*] http://www.ice.csic.es/research/gold_rtr_mining/gold_rtr.php

[*Intel IXP2855 Network Processor*] http://www.intel.com, Intel Corp.,Santa Clara, CA, 2005

[*Flynn*] http://en.wikipedia.org/wiki/Flynn's_taxonomy

[*openmp*] https://computing.llnl.gov/tutorials/openMP/

[*pthread*] https://computing.llnl.gov/tutorials/pthreads/

Part III

Parallel System Design Based on SMLOL

Parallel System Design Based on SMLOL

3.1 Introduction

In this Chapter, we provides a framework for applying the post-processing algorithms and testing the timing performance over a realistic scenario. In parallel system design, it is clear that the SMP scheme is our first choice, regarding the time to market issue. In order to evaluate the post-processing algorithm of GNSS-R application, we create a task-level parallelism system based on SMP scheme, named Symmetric Multi LEON3 On Linux (SMLOL). The work is mainly focus on the SW/OS design, instead of HW design. Meanwhile, a set of the simulations are conducted by MPARM in order to tackle on the system bottleneck in SMLOL. With the analysis of the simulation result, we can design a novel platform named HTPCP in the next Chapter, in order to get the optimized result of GNSS-R application. The following Sections are organized as follows:

- Overview the SMLOL platform, such as the potential of the GR-CPCI-XC4V development board; setup the demonstration platform; the architecture of the SMLOL platform.
- Description to the hardware design in lower layer of SMLOL platform.
- Introduction to the software and OS design in higher layer of the SMLOL platform.
- Illustrative numerical simulation results for the virtual platform by MPARM.
- Main conclusions.

3.2 SMLOL Platform Overview

As it mentioned before, an important effort has been put on the task-level parallel system design, in particular, based on multi-core system mounted with embedded OS.

In this Section, we provide a framework for applying the post-processing algorithms and testing the performance over a realistic scenario.

Before the system design, we need to understand the potential of the design board; how to setup the SMLOL demonstration platform; and what is the the blueprints of the SMLOL architecture.

3.2.1 Board Review

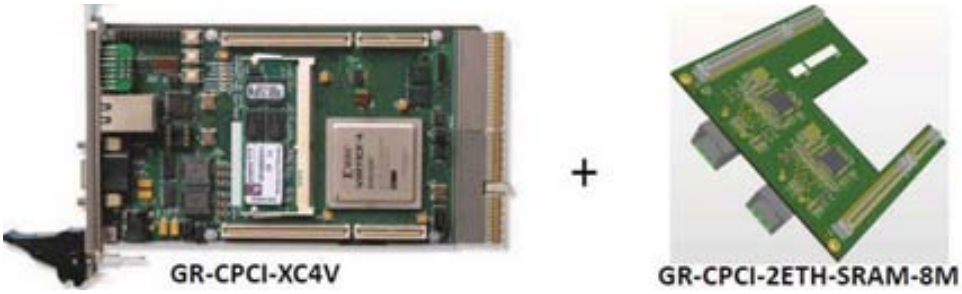


Figure 3.1: GR-CPCI-2ETH-SRAM-8M board installed on GR-CPCI-XC4V board.

The GR-CPCI-XC4V board [*GR-CPCI-XC4V LEON Compact-PCI development board*] has been developed as a co-operation between Aeroflex Gaisler and Pender electronic design as shown in Fig.3.1. In order to support the early development and fast prototyping of LEON3 designs, the board incorporates with the following components: the Xilinx Virtex-4 FPGA (XC4VLX2001513); on-board configuration proms (6 x XC18V04); 16 MB FLASH prom (4MB x 32); 1 standard SO-DIMM socket for up to 256 MB SDRAM; ethernet PHY 10/100 Mbit transeiver; 33/66 MHz 32/64-bit CPCI interface (3V); standard RS-232 UART port for DSU or slave; 120-pins memory and custom I/O expansion connectors (AMP-177-984-5; JTAG and slave-serial FPGA programming capability; CPCI system controller (clock distribution and PCI arbitration); 1 Mbyte static ram (256K x 32) adapter etc.. It supports LEON3 core frequencies up to 100 MHz.

The GR-CPCI-2ETH-SRAM-8M is a custom mezzanine board which is developed for the GR-CPCI-XC4V FPGA development board as shown in Fig.3.1. This mezzanine board provides two Ethernet PHY with RJ45 connectors and 8MB of 10ns SRAM memory. The board implements 2 banks of SRAM memory, each bank is 4MB (512kword x 32 bits) in size, giving a total of 8MB of SRAM memory. The SRAM devices are Cypress CY7C1061 types with 10ns access time. The memory is in fact 40 bits wide in order allow Error Detection and Correction (EDAC) checks bits to be used for the SRAM memory. The Ethernet PHY interface chips on the mezzanine board are National Semiconductor DP83848 devices which are capable of being used either with a Media Independent Interface (MII) configuration or with an Reduced Media Independent Interface (RMII).

Table 3.I lists the features of three types of FPGA: XC5VLX110T, XC4VLX200, XC2VP30. From the comparison of LUTs, we find out that XC4VLX200 is 2.5 times

Table 3.I: Three types of FPGAs comparison.

Development Board	Maxima Freq.	FPGA Features	Slices	4 Input LUTs	IOB	DSP	BRAM	DCM	BSCANs
Virtex-5	100MHz	XC5VLX110T	17280	69120	680	64 DSP48e	5328 Kb	12	2
GR-CPCI-XC4V	100MHz	XC4VLX200	89088	178176	960	96 DSP48s	6048Kb	12	4
Virtex-II Pro	100MHz	XC2VP30	13696	27392	644	136 MULT18 × 18s	2448Kb	8	1

bigger than XC5VLX110T, and 6.5 times bigger than XC2VP30. Thus we can infer that the XC4VLX200 FPGA is adapted up to 12 LEON3 cores.

3.2.2 Setup Demonstration Platform

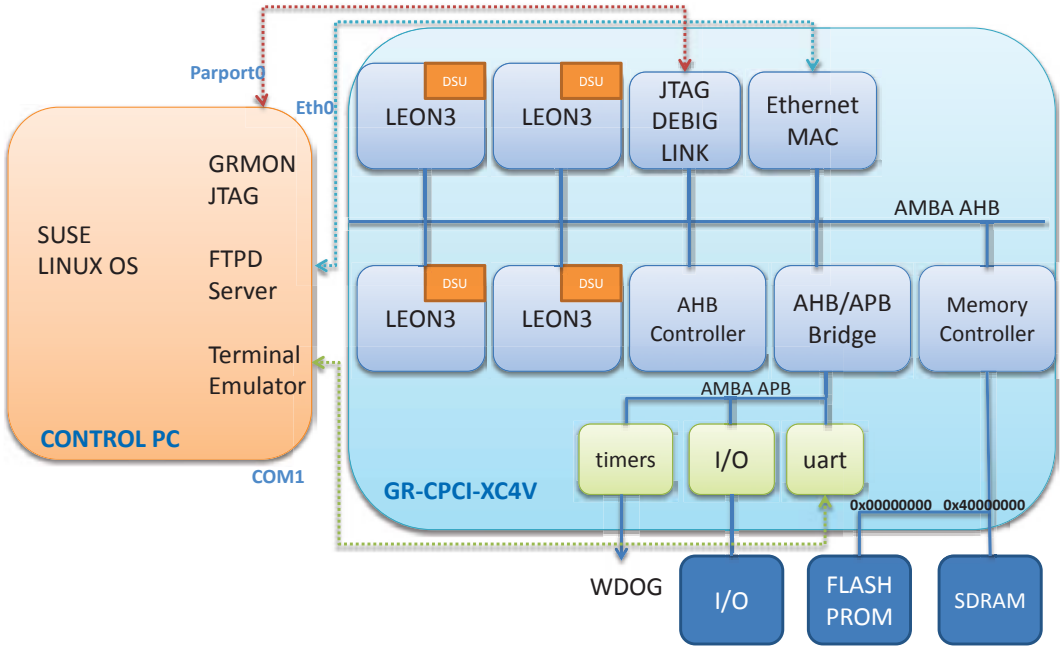


Figure 3.2: SMLOL work schematic.

The aim of this demonstration is to transmit and execute the input waveforms from a Control PC to the Xilinx GR-CPCI-XC4V board, then send the result back to the Control PC by ethernet as shown in Fig.3.2. The Control PC emulates the function of GOLD-RTR. The GR-CPCI-XC4V board setups the SMLOL platform. The demonstration platform includes 8MB prom @0x00000000 which contains the bitstream programming file of LEON3MP (leon3mp.bit), and 512MB external SDRAM @0x400000000 which contains the RAM image file (image.dsu) of Linux kernel. The image file (image.dsu) is downloaded at the address *0x40000000* by GRMON. Thus LEON3 processors will start operation from this address, and the Linux boot sequence is displayed on the Hyper terminal. There are five steps to realize this demonstration platform:

- Configure the GRLIB and LEON3 system.
- Generate the bitstream file for the Hardware layer and download it on board.
- Create the FTP client in the Linux image, and install the ftp server on the Control PC to store the input data.
- Configure the custom application from the busy box.
- Build and download the Linux kernel image and load it into the SDRAM on the board.

3.2.3 SMLOL Architecture

The SMLOL architecture is restricted to homogeneous processors with a global shared memory. The SMLOL architecture includes two parts of design: HW design in the lower layer; SW and OS design in the upper layer. The HW design is including the choosing of processors, endianess design, memory hierarchy design, AMBA bus and Ethernet interface design. The SW and OS design is including the Linux kernel design, file system, C library, compiler and processes of each application.

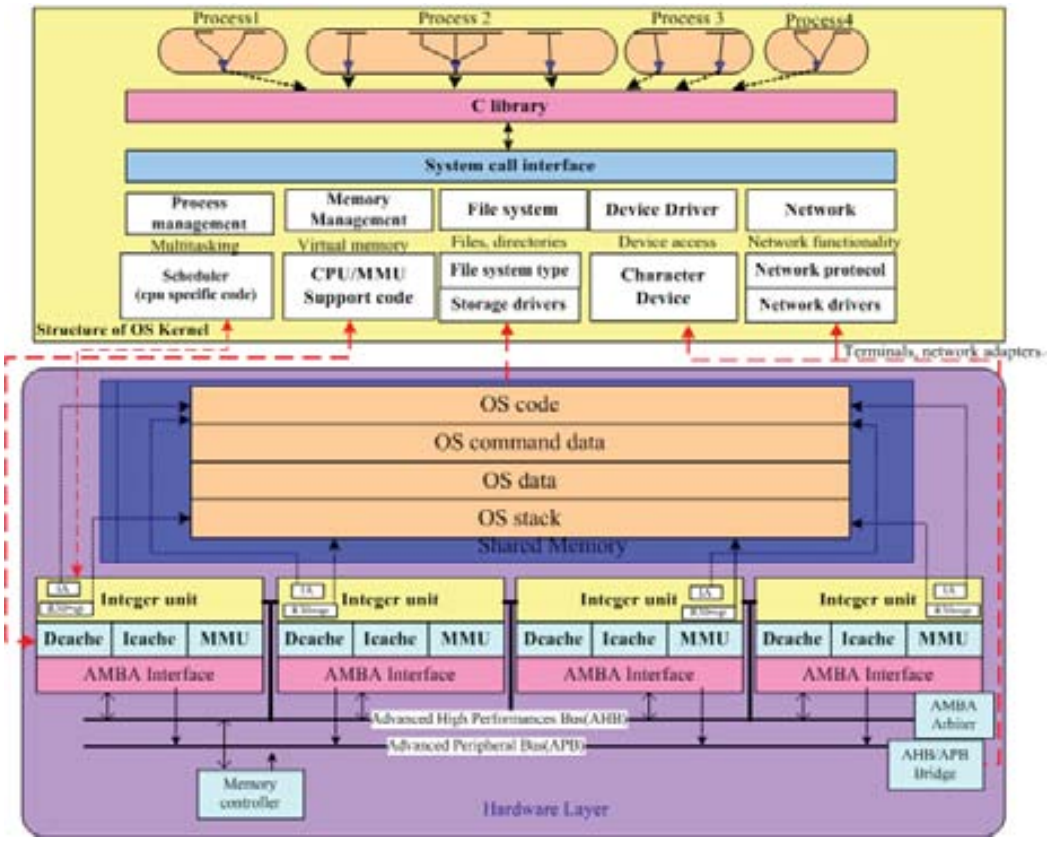


Figure 3.3: SMLOL architecture.

As shown in Fig.3.3, the system is integrated with multi-LEON3 processors with their own caches and MMU, but shared the main memory. Taking into account time-to-market consideration and rapid prototype development, a shared memory system appears as the first option to explore, since a general OS can run on the entire system by a uniform memory. Note that a common bus-based SMP organization has the scalability limitation due to the bus congestion and the cache coherency issues, therefore, the bus-based SMP platform limits the number of processors to 8 or 16 cores.

It is easy to build up the higher OS level and SW programming. The system/user applications are compiled by the cross-compiler (sparc-linux-g++) and installed in the romfs of SNAPGEAR 2.6 kernel. Task level parallel computing is supported in SMLOL, the OS scheduler leverage the processes to each processor, and access to the independent data file on the shared memory. SMLOL has the potential to avoid the data dependency and supply workload fairness principle on each processor. The SNAPGEAR embedded Linux OS [*Linux for LEON processors*] can take advantage of increased number of processors in a SMP system, where each processor can do the inter-processor communication in the shared memory.

3.3 Hardware Design in Lower Layer

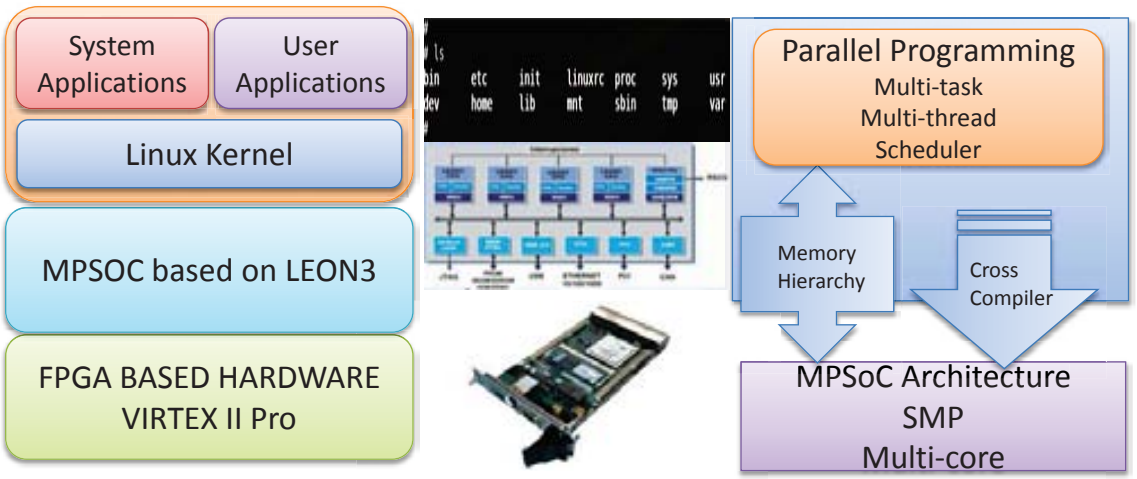


Figure 3.4: Linux on SMP design flow.

The aim of this work is to build-up a system platform which can obtain the real-time geophysical parameters by processing the consecutive 1 ms CC-WAV of GOLD-RTR instrument, and also adapt the in-flight campaign environment over the space. In order to adapt the parallel processing design, our first exploration is the investigation of the platform as shown in Fig.3.4. It is like building a Personal Computer (PC), it retains the processors, memory, bus and other peripherals. However, conduct as a FPGA, we need to design all these components from the system level to the layout level by synthesis and place & route processes. Thereby, we

can achieve the bitstream file and load it into the PROM of the FPGA, make it work whenever the FPGA is power on.

The following sections are focused on the hardware design on SMLOL platform. The first deals with the the configurable processors comparison and their development tools. Then it describes the endianess design. Finally, it describes the memory hierarchy design.

3.3.1 Configurable Processors and Development Tools

In this Section, we will conduct an evaluation of three reconfigurable FPGA based processors - two soft cores - MicroBlaze (MB) and non-fault-tolerant LEON3, and one hard core (PowerPC 405). These three processors are the most popular architectures in the FPGA design. They are all 32-bit processors and support for radiation effects testing and mitigation. To enable the flexibility of these reconfigurable processors, we need to incorporate the development tools to build the processors on FPGA, and evaluate their performance metrics by benchmarks. A standard performance benchmark, Dhrystone, is developed for the fixed-point operation processors.

The following subsections will compare the configurations of each processor and briefly introduce the development tools of each processors. Finally combined with the benchmark results, it illustrates the performance metrics of each processor.

3.3.1.1 Processors Comparison

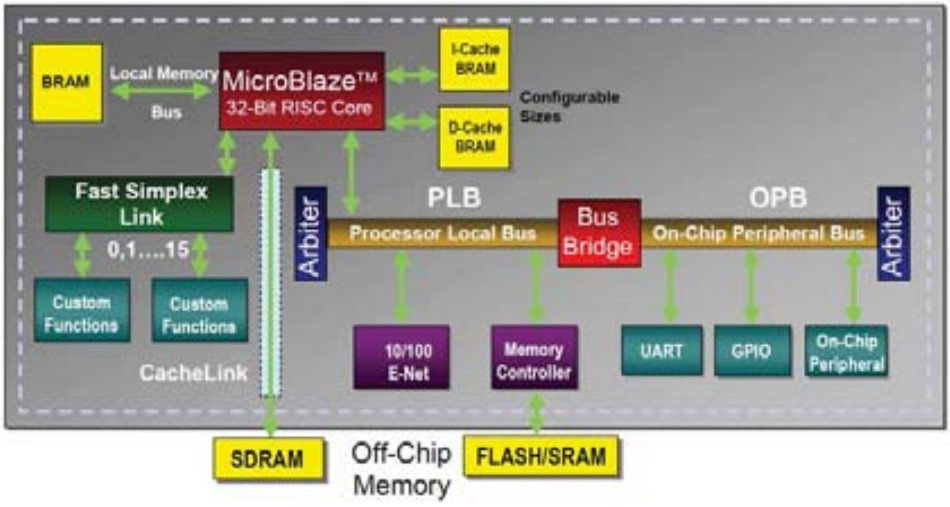


Figure 3.5: Microblaze architecture.

As shown in Fig.3.5, the Microblaze processor [*MicroBlaze processor*] is a 32-bit Harvard Reduced Instruction Set Computer (RISC) architecture optimized for implementation in Xilinx FPGAs with separate 32-bit instruction and data buses. It can runs at a full speed to execute programs and access data from both on-chip

and external memory at the same time. The architecture of Microblaze is a single-issue, 3-stage pipeline with 32 general-purpose registers, an Arithmetic Logic Unit (ALU), a shift unit, and two levels of interrupt. The design can then be configured with more advanced features to tailor to the exact needs of the target embedded application such as: barrel shifter, divider, multiplier, single precision floating-point unit (FPU), instruction and data caches, exception handling, debug logic and others. It works with 1.19 Dhrystone MIPS (DMIPS)/MHz performance. The processor has up to four interfaces for memory accesses: Local Memory Bus (LMB), IBM's on-chip Peripheral Bus (OPB) or Processor Local Bus (PLB), and Xilinx CacheLink (XCL). The LMB provides single-cycle access to on-chip dual-port block RAM (BRAM). The OPB/PLB interface provides a connection to both on-chip and off-chip peripherals and memory. The difference between PLB and OPB is that the PLB connects with high-bandwidth master and slave device, and the OPB decouples lower bandwidth devices from the PLB. The XCL interface is intended for use with specialized external memory controllers. Microblaze also supports up to 8 Fast Simplex Link (FSL) ports, each with one master and one slave FSL interface. The FSL is a simple point-to-point interface that connects user developed custom hardware accelerators to the Microblaze processor pipeline to accelerate time-critical algorithms.

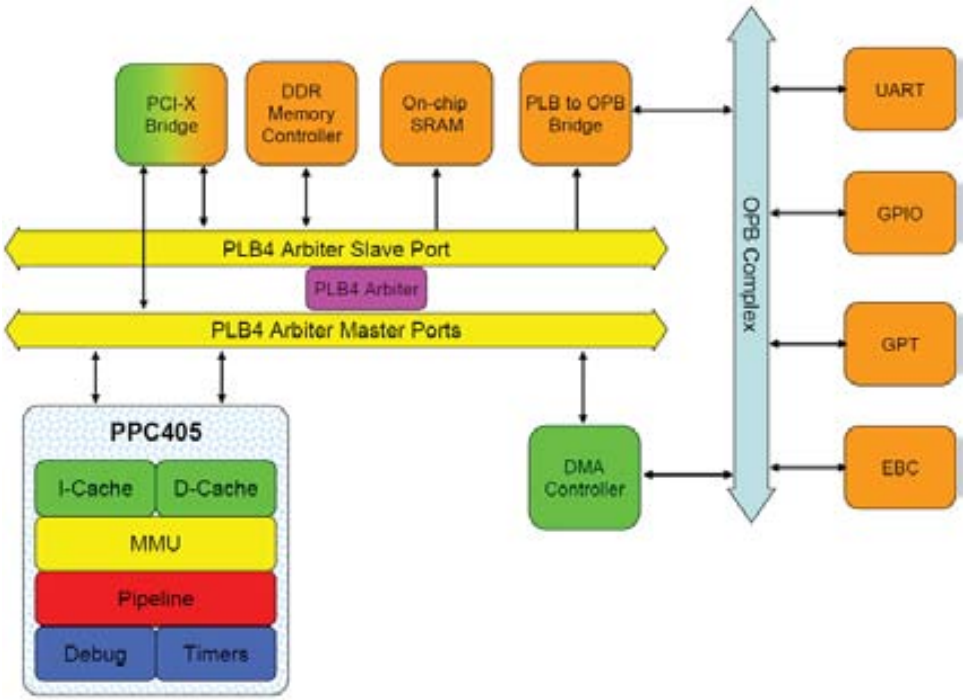


Figure 3.6: PowerPC 405 architecture.

As shown in Fig.3.6, the IBM PowerPC 405 core [*IBM PowerPC 405 core*] is a 32-bit RISC processor and a key element of IBM's power architecture licensing portfolio. This licensable embedded core integrates a scalar 5-stage pipeline, separate instruction and data caches, a JTAG port, trace FIFO, multiple timers and a MMU. It works with 1.52 DMIPS/MHz performance. The core is available as a hard macro in the IBM premium process technologies including 130nm, and also as a fully synthesizable

core that can be fabricated at multiple foundries. The 405 core can be integrated with peripheral and application-specific macro cores using the CoreConnect bus architecture to develop system-on-a-chip solutions. A typical SOC implementation based on the PowerPC 405 CPU and CoreConnect, uses a three level bus structure for system level communication, configuration and control functions. High bandwidth memory and system interfaces are tied to the 405 Core via the PLB. Less demanding peripherals share the OPB and communicate to the PLB through the OPB bridge. On-chip configuration and control registers implemented in various IP cores and at chip-top in an SOC are connected to the CPU by the Device Control Register (DCR) bus. This three level bus architecture provides common interfaces for the IP cores and enables quick turnaround custom solutions for high volume applications.

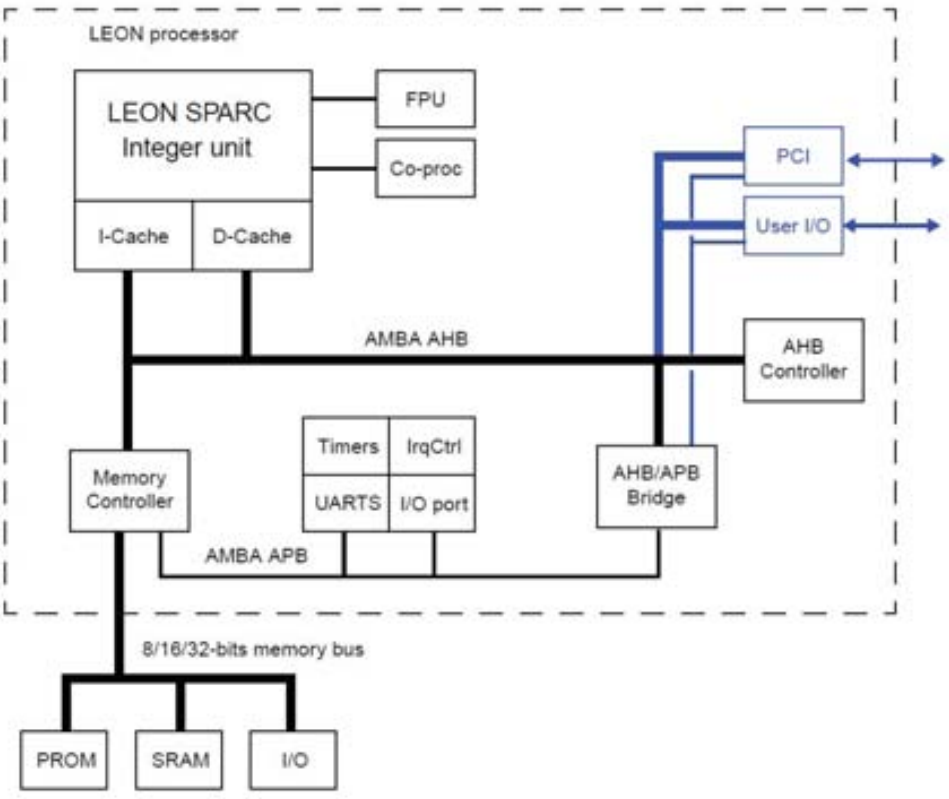


Figure 3.7: LEON3 architecture.

The LEON3 is open-source VHDL models of a 32-bit processing core that is fully compliant with the standard IEEE-1754 SPARC V8 architecture. It was originally developed by Jiri Gaisler at ESA for critical space applications. The core comes with SPARC V8 compliant integer units complete with hardware multiply, divide and MAC units. LEON3 has a pipeline depth of 7 stages. It features a Harvard memory architecture, and a configurable set-associative cache sub-system. The number of registers in their register files is configurable within the SPARC V8 standard (2 to 32 registers). LEON3 also provides an interface to one of several available FPU cores as well as custom co-processors. They also include support for an optional debug unit, timers, watchdogs, UARTs and interrupt controllers. The processor is

fully synthesizable and up to 16 cores can be implemented in ASymmetric Multi-Processing (ASMP) or Symmetric Multi-Processing (SMP) configurations. A typical configuration with four processors is capable of delivering up to 1.7 DMIPS/MHz of performance. The LEON3 multiprocessor core is available in full source code under the GNU GPL license for evaluation, research and educational purposes. A low cost license is available for commercial applications. As shown in Fig.3.7, there are two on-chip buses are provided: AMBA AHB and APB. The APB bus is used to access on chip registers in the peripheral functions, while the AHB bus is used for high-speed data transfers. The full AHB/APB standard is implemented and the AHB/APB bus controllers can be customized through the TARGET package.

3.3.1.2 Development Tools

Both PPC405 and MB processor-based systems are defined by the Xilinx Platform Studio (XPS) tools that are included in the Xilinx Embedded Development Kit (EDK). XPS contains a “base system builder“ wizard that can be used to define a basic system. If the system that contains the hardware and software definitions differs from the basic configuration, it will require the manual editing of the Microprocessor Hardware Specification (MHS) file and Microprocessor Software Specification (MSS) file. The Xilinx EDK is used exclusively to synthesize the PPC405 and MB processor system. The XPS graphical user interface is used to generate the HDL files and libraries that define the PPC405 and MB systems. XPS will put together all of the IP based upon connections described in the MHS file. The Synplify FPGA synthesis tool as the third-party tool is supported on all the platforms. The software development tool Xilinx EDK is exclusively for the software development of the PPC405 and MB processors. It is based on GNU compiler tools and two different debugging environments. An Eclipse-based Software Development Kit (SDK) is a Xilinx microprocessor debugger interface.

The LEON3 processor is configured by a script-based graphical tool “xconfig“. This tool allows the user to customize all configurable aspects of the LEON3 processor. The main design steps of LEON3/GRLIB can refer to the GRLIB User’s Manual [*grlib*]. For the basic configuration of LEON3, we can manually editing the config.vhd file under the GRLIB package. There are three important files in the directory of the design:

- *local makefile*: used to generate tool-specific project scripts, e.g. *.qsf
- *config.vhd*: Configurable VHDL file for the LEON3 design
- *leon3mp.vhd*: top module VHDL file for the LEON3 design, including instantiation of cores connected to AHB/APB, and available ports for the FPGA pin assignment.

The GRLIB package of LEON3 and IPs can be downloaded from the Gaisler‘s website. The Synplify FPGA synthesis tool and Xilinx ISE as the third-party tool is supported on all the platforms. The top level design needs to be synthesized by

Synplify v 8.6.2., thus generate the netlist file *leon3.edf*. Then the netlist file needs to be placed & routed by Xilinx ISE v 10.1i, thus generate the bitstream file *leon3mp.bit*, thereby the design can be download into the GR-CPCI-XC4V board by Xilinx Manage Configuration Project (iMPACT). The FPGA (XC4VLX2001513) can fit up to twelve LEON3 processors.

The software development tool for LEON3 is accomplished by an Eclipse IDE plug-in (GRTools) or the command line tools. The cross-compiler for LEON3 processors is Bare C Compiler (BCC), which is based upon the GNU compiler tools and the Newlib standalone C-library. The debug monitor is GRMON [*GRMON User's Manual*], which supports two operating models: command-line mode and GNU debugger (GDB) mode. These two operation models allow GRMON to accept commands manually through a terminal window or act as a GDB gateway. The GRMON debugging interface allows easy control for operations such as setting breakpoints, inspecting the stack etc.. With the "info sys" command, we can check the detailed information of each attached core in real-time. Also we can debug the *elf* application with Debug Support Unit (DSU) + GRMON tool.

3.3.1.3 Performance Metrics

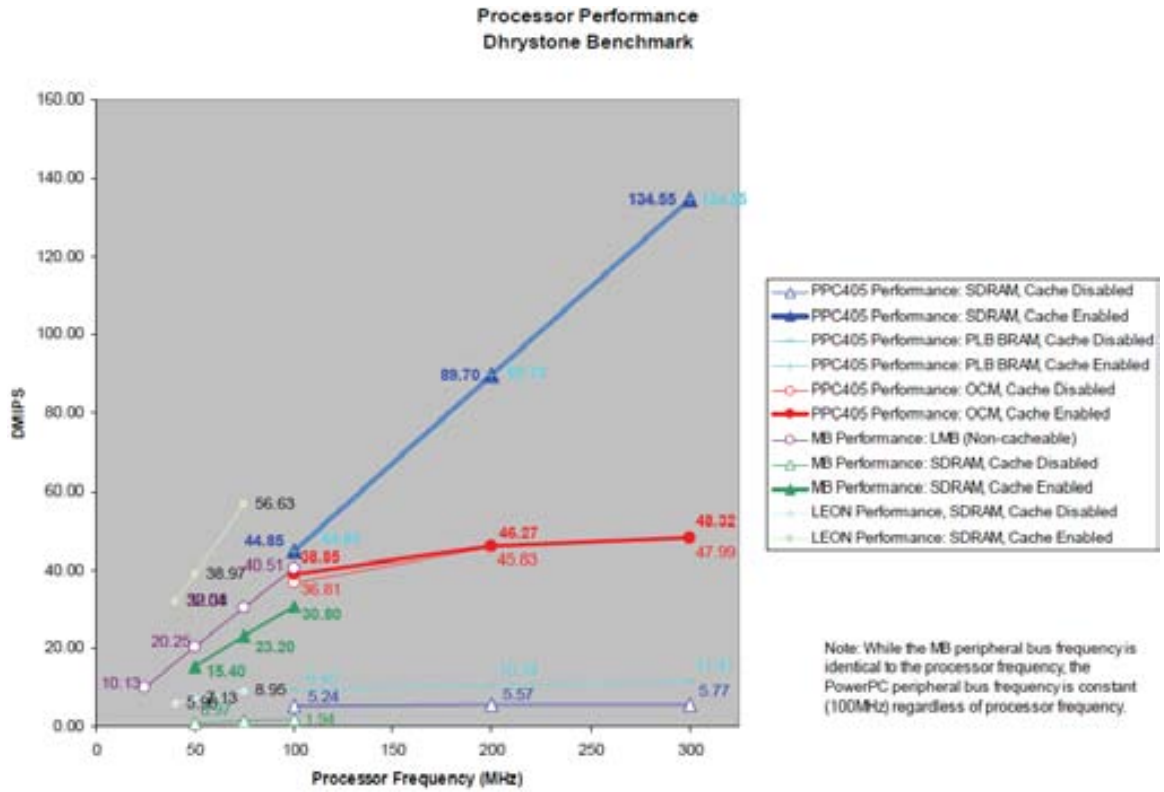


Figure 3.8: Dhrystone benchmark results on three types of processors. (Adapted from [*SANDIA REPORT*])

Dhrystone v2.1 is developed for three types of processor (PPC405, MicroBlaze

v6.00, LEON3) and executed on the GR-CPCI-XC4V development board with Virtex-4 FPGA. The main objective of this Dhrystone benchmark is to compare the timing performance of the processors, and analysis how the designs of processor impact on the FPGA resource utilization. As shown in Fig.3.8, the benchmark results of the processor show that the LEON3 is more efficient than either the MB or PPC405, since it cannot operate at the higher frequencies, it is not suitable for computationally intensive algorithms. However, the PowerPC and MB have wider operating ranges and better suited for more computationally intensive applications. Moreover, the fault-tolerant version (LEON3FT) has been designed for operation in the harsh space environment, and includes functionality to detect and correct Single-Event Upset (SEU) errors in all on-chip RAM memories.

Therefore, in this work, we choose LEON3 to handle the post-processing task. Regarding to the dedicated FIFO-style connection of MB, called FSL which is supporting for the coprocessors design in register level, we propose the MB to sustain the data distribution task.

3.3.2 Endianess Design

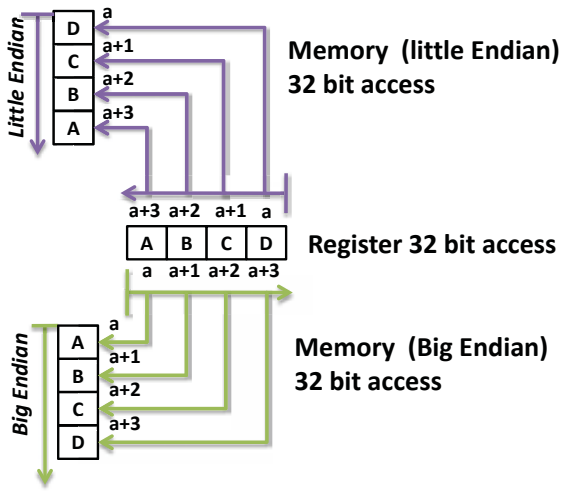


Figure 3.9: Little endian v.s. big endian.

Endianness is a problem when a binary file is created on a big-endian computer (FPGA with LEON3 processor) that can not be read on the little-endian computer (PC with Intel processor). Integers are usually significance with increasing memory addresses known as little-endian; it‘s opposite, the most-significant byte first that is called big-endian as shown in Fig.3.9. LEON3 (SPARC) is conforms to the big-endian byte ordering. This means that the most significant byte of a word has lowest address. Since our PC with Intel processor is little-endian format as well as the GOLD-RTR instrument, the input waveform from PC or GOLD-RTR instrument needs to change the byte order from little-endian to big-endian, in order to adapt the order format of LEON3 processor on our design board.

In this work, we need to swap the bytes by an ad-hoc code. Note that the required byte swap depends on the length of the variables stored in the file (Appendix A), therefore a general utility to convert endian in binary files does not exist. The conversion should follow the input waveform format, firstly get the same size of the input and result variables, and then byte rotate each input variable and assign it to the result variable as shown in Fig.3.10. It is the same in both directions (little to big and big to little) .

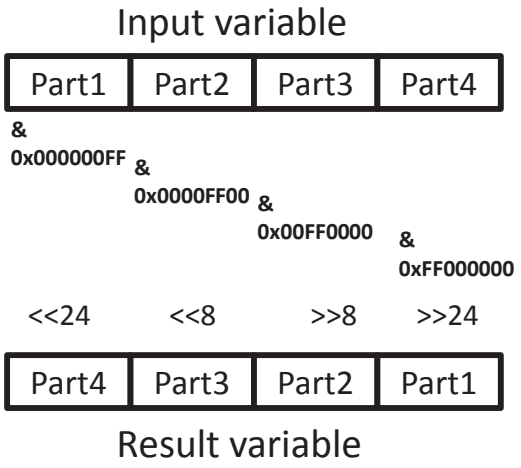


Figure 3.10: Byte rotate each input variable and assign to the result variable.

3.3.3 Memory Hierarchy Design

The memory hierarchy design of SMLOL is shown in Fig.3.11. The most critical components (Cache, MMU and S/DRAM) determine the success of an SMP architecture. Caches are made of much faster on-chip memory and integrated with the CPU additionally improves system throughput. However, the private caches may lead to cache-coherence and cache-migration problems. Since we choose the linux-2.6.21.1 kernel as the OS mounted on LEON3 platform, it requires the MMU inside the system. The MMU for LEON3 is compliant to the SPARC Reference MMU (SRMMU), it has 32-bit virtual address and 36-bit physical address, the most important character is its fixed 4KB page size. That means if the size of Dcache is bigger than 4KB, it will induce a big migration cost. The main memory is made of external off-chip SDRAM memory, it may lead to the scalability issues since all cores share the same memory and increased the memory latency.

There are two ways to optimize the memory performance of SMP: (1) constructing a suitable memory hierarchy (2) optimizing the size of it. Since the memory hierarchy will dominate the performance of SMP, we firstly pay attention to optimizing the size of it. Unlike a general-purpose processor, the memory hierarchy of LEON3 can be configured by the scripts.

In the synthesis report of Synplify Pro, we found out that the critical path of SMLOL

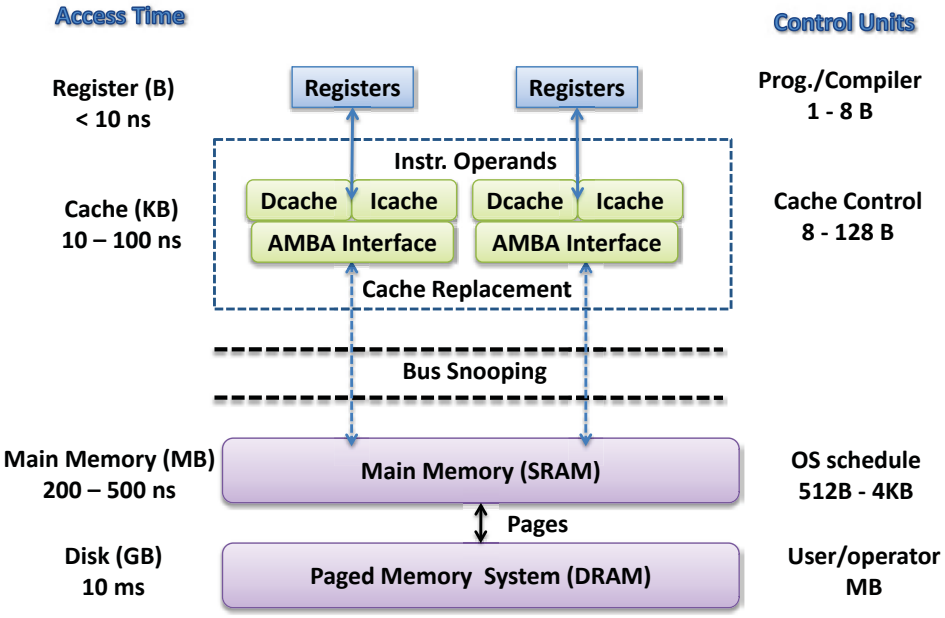


Figure 3.11: Memory hierachy design.

system is 26 ns with the slack component Dcache. This is caused by the cache coherency issues. As the number of processors increases, the performance benefit from caches is partially offseted by the long latencies incurred when one processor references data owned by another processor's cache [1]. The pipeline will be stalled because the required data is not yet available due to memory latency. In order to avoid this issue, we optimized our system with the parameters as shown in Table 3.II.

Table 3.II: Hardware parameters in SMLOL design.

System freq (MHz)	100
No. of LEON3	4
Size of Icache (KB)	64
Size of Dcache (KB)	32
Cache Replacement	LRU
Write Policy	Write Through
Size of MMU (KB)	4
Size of PROM (MB)	8
Size of SDRAM (MB)	512

3.4 Software and OS Design in Higher Layer

After building the hardware layer design of SMLOL platform, we turn into the software and OS designs. If we want to mount the embedded Linux successfully on the SMP-based hardware platform, we need to pay attention to the workload analysis and

the Linux kernel configuration. Combined with the workload analysis and the post-processing algorithm, we can evaluate the feasibility of the SMLOL platform. The SMLOL design is quite generic in the sense that they should not be restricted to this specific case of study. This research is divided into four parts:

- Analysis the parallel workload of the GNSS-R postprocessing application.
- Description to the post-processing code - Coherent/Incoherent.
- Generation to the embedded Linux OS for SMLOL architecture. The design is focus on the configuration of the compiler, Linux kernel analysis, CPU configuration, and ethernet configuration.
- Evaluation to the timing performance of the multi-task applications in SMLOL platform. There are three important parameters that to be considered: speedup ratio, system throughput and standard deviation of execution time.

3.4.1 Parallel Workload Analysis and Mathematical Modeling

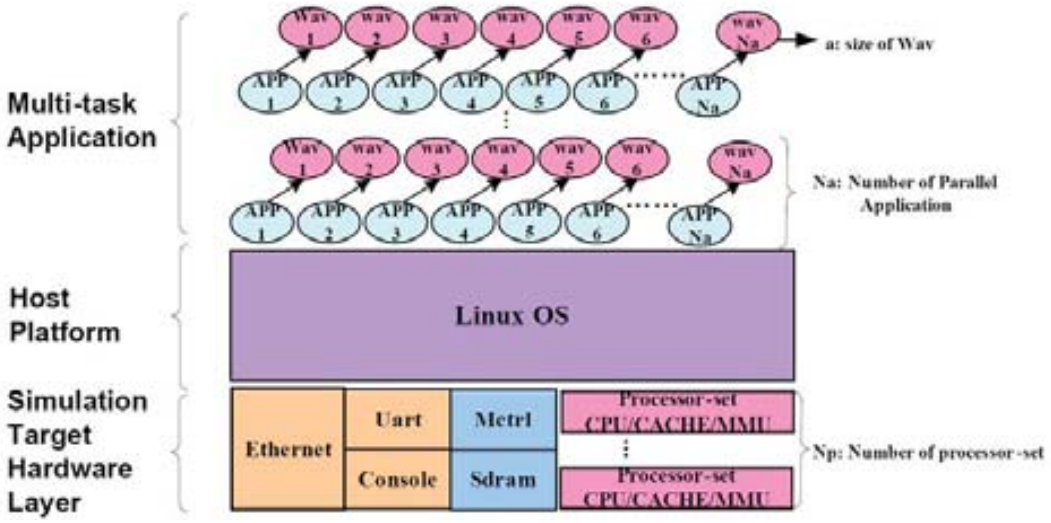


Figure 3.12: Multi-task application.

The parallel workload concentrates on modeling and implementing GNSS-R post-processing algorithm on SMLOL. This application is implemented in the task-level parallelism. The parallel workload is presented by a mathematical model. The design space is resort to full system OS to test the computing and communication reliability. The challenge in modeling workloads is the gap between the software level and hardware level. To simplify the timing affection and maximize the system efficiency, we denote on the mathematical model of the parallel workload based on the characteristic of GNSS-R applications. The reasons of implementing on the SMP platform are shown below:

- More than one program can run at the same time on an SMP system, it can get better throughput than a uni-processor, since different programs can run on different cores simultaneously.
- SMP architecture supports multiprocessing, in order to avoid the processors remaining idle.
- The OS scheduler can leverage the processes on each core. Therefore, it has the potential to avoid the data dependency and supply workload fairness principle on each core. And the processor utilization could reach its maximum potential.

Table 3.III: Timing affections in parallel system.

Layers	Components	Time affection
Application Layer	Task-level parallelism	Parallel Workload Rate(Na/Np)
	Prog. /Compiler	Scheduler
Memory Hierarchy Layer	Cache /Memory	Memory Access Time
	Memory/ Bus	Bus Busy Ratio

As shown in Fig.3.12, the multi-task parallel application is composed by the user program (App), Number of parallel applications (Na), objective stream data (Wav), Number of processors (Np) and some system programs i.e. the kernel command, drivers etc.. As shown in Table 3.III, we classify two major layers in the parallel system: Application Layer and Memory Hierarchy Layer. At the Application Layer, the parallel workload rate and the scheduler determine the execution time, since each task can be moved easily with the support of OS. At the Memory Hierarchy Layer, memory access time and bus busy ratio will be the key point to avoid the data dependency and supply workload fairness principle on each processor, since they determine the communication between processor and memory, also the communication between the memory and bus.

Let's recall the system throughput that described in Chapter II. The GOLD-RTR schedules the coherent integration time of 1 ms over ten correlation channels, with 64 lags each, work simultaneously and continuously. The input raw data is 10 000 waveforms per second, each waveform being 64 lags long. The size of waveform is fix to 160 B during each time slot (1 ms). Each waveform contains the parameters of Coordinated Universal Time (UTC), therefore real time parallel processing will become the reality. The calculation of aggregate system throughput would be 12.8Mbps:

$$T_{in} = \frac{10\text{channel} \times 160\text{B}}{1\text{millisecond}} = 1.6KB/m\,\text{sec} = 12.8Mbps. \tag{3.1}$$

Now we propose a mathematical model to analysis the timing performance of parallel system. This method can help us effectively evaluate the system throughput and balance the parallel workload. The target architecture is composed by several timing models:

1. Inherently sequential computations: $\sigma(n_p)$,
2. Potentially parallel computations: $\varphi(n_p)$,
3. Communication operations: $\kappa(n_p, p)$,
4. Size of parallel workload: $\rho(n_p)$.

The speedup expression following the Amdahl's law is shown below:

$$\psi(n_p, p) \leq \frac{\sigma(n_p) + \varphi(n_p)}{\sigma(n_p) + \varphi(n_p)/p + \kappa(n_p, p)}, \tag{3.2}$$

Efficiency is a fraction of speedup ratio presents as the following:

$$\varepsilon(n_p, p) \leq \frac{\sigma(n_p) + \varphi(n_p)}{p\sigma(n_p) + \varphi(n_p) + p\kappa(n_p, p)}, \tag{3.3}$$

Follow the analysis, we could minimize the $\kappa(n_p, p)$ in the Memory Hierarchy Layer by leverage the $\rho(n_p)$ into small units, and convert the $\sigma(n_p)$ into $\varphi(n_p)$ in the Application Layer. It reveals that there is a parallel workload rate p to express the speedup ratio and efficiency. Based on the characteristic of GNSS-R post-processing application, we could get the relation between the size of parallel workload $\rho(n_p)$, parallel workload rate p and the number of processors n_p. Assume that the $\rho(n_p)$ is a constant value, the parallel system will take $\varphi(n_p)$ time to process the size of the parallel workload $\rho(n_p)$, therefore, the total processing time T is expressed below:

$$T = \rho(n_p) \times p \times \varphi(n_p), \tag{3.4}$$

3.4.2 The Post-Processing Code - Coherent/Incoherent

The main focus of this Section is a brief explanation of the algorithm in the post-processing code. The post-processing algorithm - Coherent/Incoherent is implemented by C language and compiled by O2 optimization level. In Chapter V, we will detail on the post-processing algorithm and its physical meaning.

Firstly we need to understand the coherent and incoherent process in the GNSS-R application. It is possible to integrate coherently over longer or shorter periods of time and then accumulate the correlation powers in a similar manner. Varying the coherent integration interval to arrive at an optimal value will be investigated as part of our ongoing research.

As the GOLD-RTR schedules consecutive time slots of 1ms, and generates ten correlation channels' CC-WAV. Fig.3.13 shows the collected complex values of one correlation channel during 1s. In order to exhibit interference of one identical

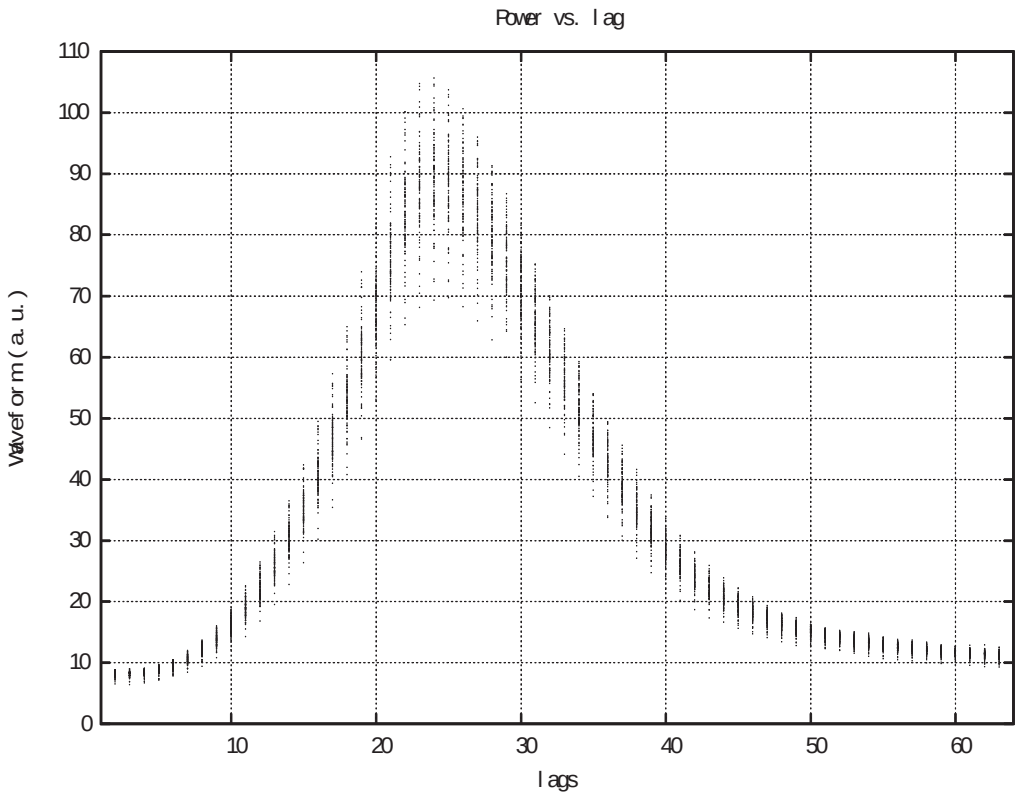


Figure 3.13: Coherent process input: Every 1s, get 1000 waveforms, 64 lag each, for each correlation channel, total 64000 complex values for each each correlation channel.

waveform (reflected and directly), our approach is to integrate the reflected CC-WAV for each channel by adding coherently the different time slot (64 lags) for 1 s, and compares the integration result with the corresponding direct one. It could dramatically reduce the load of the link between satellite and ground. The idea of this algorithm for each correlation channel has two main steps:

1) By integrating coherently the complex value $C = I + iQ$, we can achieve the variable C_{coh} during the intervals dt_{coh}:

$$C_{coh}(lag_k, Channel_j, t_{coh}) = \sum_{\tau} C(lag_k, Channel_j, \tau), \tag{3.5}$$

where the sum applies to all τ within the interval $[t_{coh} - dt_{coh}/2\ ,\ t_{coh} + dt_{coh}/2]$, if this interval do not contain a possible navigation bit transition. In our application dt_{coh} will take one of the values (1 ms, 2 ms, 4 ms, 10 ms, 20 ms).

$$C_{coh}(lag_k, Channel_j, 1ms) = \sum_{\tau=0.5ms}^{1.5ms} C(lag_k, Channel_j, \tau), \tag{3.6}$$

$$C_{coh}(lag_k, Channel_j, 2ms) = \sum_{\tau=1ms}^{3ms} C(lag_k, Channel_j, \tau), \tag{3.7}$$

$$C_{coh}(lag_k, Channel_j, 4ms) = \sum_{\tau=2ms}^{6ms} C(lag_k, Channel_j, \tau), \tag{3.8}$$

$$C_{coh}(lag_k, Channel_j, 10ms) = \sum_{\tau=5ms}^{15ms} C(lag_k, Channel_j, \tau), \tag{3.9}$$

$$C_{coh}(lag_k, Channel_j, 20ms) = \sum_{\tau=10ms}^{30ms} C(lag_k, Channel_j, \tau). \tag{3.10}$$

2) By integrating incoherently the complex value C_{coh}, we can achieve the variable C_{incoh} during intervals dt_{incoh} :

$$C_{incoh}(lag_k, Channel_j, t_{incoh}) = \sum_{\tau=0.5s}^{1.5s} |C_{coh}(lag_k, Channel_j, \tau)|^2, \tag{3.11}$$

where the sum applies to all τ within the interval $[t_{incoh} - dt_{incoh}/2$, $t_{incoh} + dt_{incoh}/2]$. In our application, dt_{incoh}=1s.

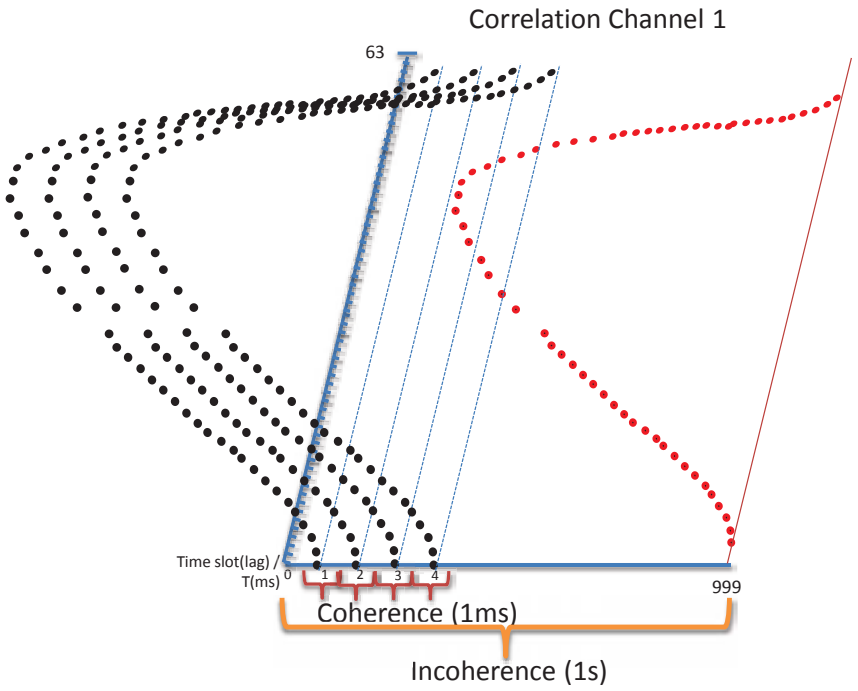


Figure 3.14: Incoherent process output: Every 1s, get 64 integrated complex values for each correlation channel.

The role of coherent process is to integrate the complex value for each time slot and each channel. The role of incoherent process is mainly to achieve the amplitude

of each integration value. For instance, if the coherent integration time is 1 ms, therefore, every 1 ms, we can always receive one waveform for each channel, which has a header of 32 bytes and 128 bytes of complex-valued samples. The 64 complex signal pairs $[I\sum(k), Q\sum(k)]$ are presented for the complex value of each time slot, they accumulated over 1 s for each channel. And every 1s, the integrated complex value go through the incoherent process, we can get 64 amplitude of the integrated complex value for each time slot, they calculated separately for each channel as shown in Fig.3.14. The incoherent function for each channel is shown in the following:

$$C = I + iQ, \tag{3.12}$$

$$C_{coh}(k,j) = \sum_{j=0}^{N_w-1} I_k(j) + i \sum_{j=0}^{N_w-1} Q_k(j), \tag{3.13}$$

$$|C_{coh}(k,j)| = \sum_{j=0}^{N_w-1} \sqrt{I_k(j)^2 + Q_k(j)^2}, \tag{3.14}$$

$$|C_{coh}(k,j)|^2 = \sum_{j=0}^{N_w-1} [I_k(j)^2 + Q_k(j)^2], \tag{3.15}$$

$$C_{incoh}(k,j) = \sum_{j=0}^{N_w-1} [I_k(j)^2 + Q_k(j)^2] \Big/ N_w, \tag{3.16}$$

where k represents the lag index (0 to 63), j represents a waveform index (0 to 999), and N_w represents number of waveform for each lag (N_w=1000). The $C_{incoh}(k,j)$ correspond to the amplitude of the complex signal pairs for each time slot.

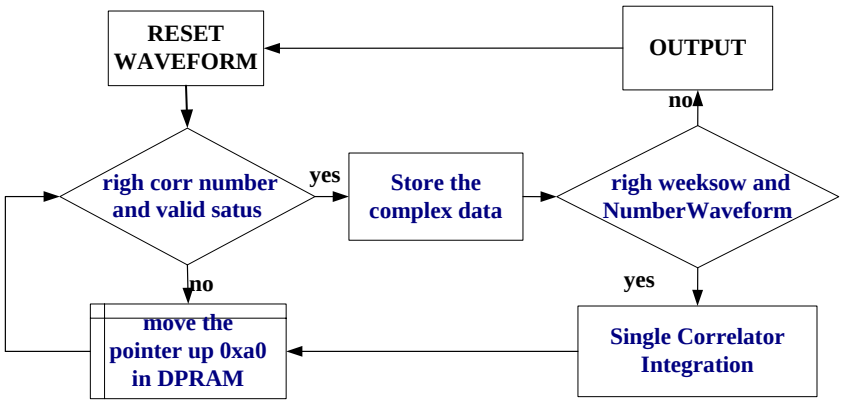


Figure 3.15: Design flow of Post-processing code.

Fig.3.15 depicts the design flow of the post-processing code. After resetting the system and initialing the structure of the output waveform, the first waveform is read at the entry address pointer. Then, the system decides if the received waveform has the right correlator number and the valid status. Otherwise, the system moves the pointer up to the stack of DPRAM and checks the next waveform. If the correlator

number and status are valid, the system stores the waveform (complex data), and then decides if this waveform has the right second of week (SOW) and a valid waveform number. In this case, the stored complex data is included in the integration function and the pointer is moved back to the stack of DPRAM in order to decide the next waveform. If the waveform doesn't have the right SOW or the valid waveform number, the system generates the output result and resets the waveform structure to restart. The equation of the post-processing function is shown below, eventually, we can always get 64 values of WaveformOutput.data[i] every 1 second. Each value respects to the integrated output of each time slot.

$$\text{WaveformOutput.data[i]} = \sum_{\text{j=0}}^{\text{NumberWaveform}} \sqrt{\text{I}_\text{i}(j)^2 + Q_i(j)^2)} \Big/ NumberWaveform \qquad (3.17)$$

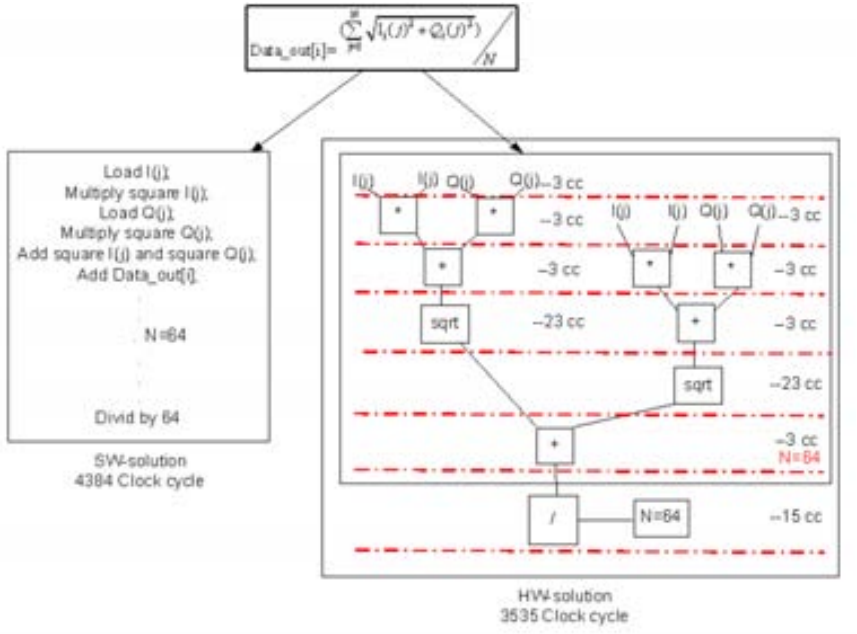


Figure 3.16: Software design versus hardware design.

In general, this application can be realized either as the software algorithm (e.g., C or C++) or as the structural hardware (e.g., VHDL). As shown in Fig.3.16, based on the lists of the cycles per instruction of LEON3 processor in the GRLIB IP Cores Manual [*grip*], we estimate the software routine needs 4383 clock cycles to calculate this simple post-processing application. However, based on the simulation of the structural hardware (VHDL) for the same algorithm, the hardware routine only takes 3535 clock cycles to compute the same result. It is more efficient to realize the pipeline computation in the function level by the hardware method. However, in order to realize the flexibility of the algorithm, we choose the software method to realize the post-processing algorithm. Dealing with this strict timing-driven application, we need to strengthen the real-time control by the hardware-assisted.

3.4.3 Linux Embedded OS Analysis and Design

From the OS perspective, there are two major classes of OS architectures for the multiprocessor system, SMP kernels and master-slave kernels [2]. SMP kernels are the most widely used OS architecture. All processors run a single copy of SMP kernel that exists on shared memory. Since all processors share the code and data of the SMP kernel, **cache migration** and **synchronization** are the key issues for the SMP Linux kernel design. In our case, we choose the open source of SNAPGEAR Linux kernel (Linux 2.6.21.1) as the Linux embedded OS on the board, since it supports for the LEON3 processor and the SMP architecture. The main design steps of SNAPGEAR Linux kernel can refer to the Snapgear for LEON manual [*Snapgear for LEON manual*].

The process of booting the Linux OS is quite similar as the BIOS process: 1) compile our Linux kernel by a proper compiler (sparc-linux-gcc); 2) built the Linux kernel with the system/user applications and file system; 3) generate the ROM image (image.flashbz) or the RAM image (image.dsu) and load it into the FPGA. The Linux kernel will initialize the components on the board as soon as we load the image file into the FPGA. Then we can control and operate all the devices on the board, also execute the applications in the file system.

3.4.3.1 Compiler Requirement

There are three cross-compilers suitable for LEON3 processor with the SPARC V8 instruction architecture: Bare-C Cross-Compiler (BCC); RTEMS Cross-Compiler (RCC); GLibC-Linux Cross-Compiler (Glibc-Linux).

Firstly we use BCC (*sparc-elf-gcc*) to get the *elf* executable file for LEON3 by the instruction below. By this way, we can trace and profile the application on board. The instruction of CPU0 begins with *0x40000000* and CPU1 begins with *0x40000800* that is shown in the disassembly window of GRMON. However, it does not support for the multi-threaded application, and can not be used in the OS.

```
sparc-elf-gcc -mv8 -msoft-float -g -O2 WavIntegration.c -o WavIntegraion.o -lm
```

Secondly we use RCC (*sparc-rtems-gcc*) to get the *elf* executable file which includes Board-Support Package (BSP) for LEON2, LEON3 and ERC32, and target it into the RTEMS on the hardware by the instruction below. RCC realizes the program on board without porting to the embedded Linux OS. The RTEMS executables file is in *elf* format and has three main segments: text, data and bss. The text segment is default at address *0x40000000* for LEON2/3, which is followed immediately by the data and bss segments. The stack starts at top-of-ram and extends downwards. BSPs provide interface between RTEMS and target hardware through initialization code, which are specific to target processor and a number of drivers. Console and timer drivers are supported for all three types of processor. However it is not support multi-threaded application (e.g. POSIX).

sparc-rtems-gcc -mv8 -msoft-float -g -O2 WavIntegration.c -o WavIntegraion.o -lm

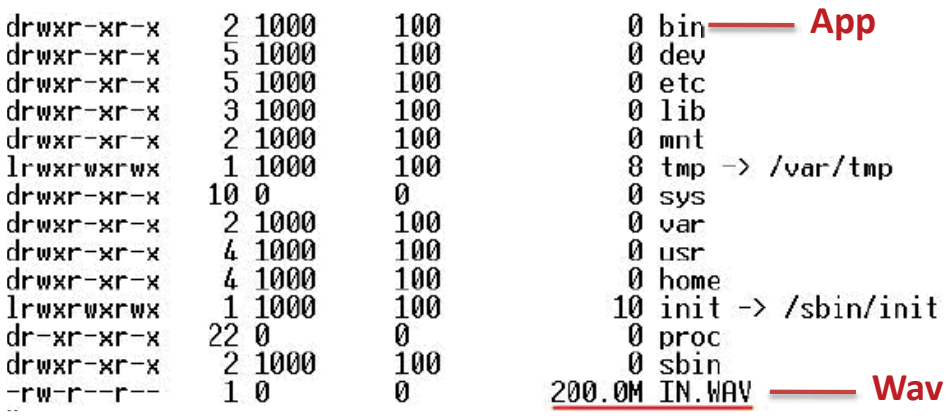


Figure 3.17: The file system of the SNAPGEAR Linux

Finally, we decide to use Glibc-Linux cross-compiler (*sparc-linux-gcc*) to get the *elf* executable file, and port it into the file system of the SNAPGEAR Linux Kernel (image.dsu) as shown in Fig.3.17. The waveforms is transmitted into the file system by ethernet. And the instruction of the compilation is shown below. The reason is that it supports for the multi-threaded application (POSIX). Also we can use the file system to process the waveforms.

sparc-linux-gcc -mv8 -msoft-float -g -O2 WavIntegration.c -o WavIntegraion.o -lm

Table 3.IV: The impact on compiler optimization levels.

Compiler Optimization	Executable Time(s)	Size of Executable file(B)
O1	11.921	106777
O2	11.378	105998
O3	11.346	197618

However, the different optimize flags may induce the different performances as shown in Table 3.IV. For instance, the same application which is compiled by the same compiler with the different optimization flags may induce the different executable time and size of executable file. Compared to the results, the level 2 optimization generates the best tradeoff between timing and area.

3.4.3.2 Linux Kernel Analysis

The Linux kernel design concentrates on implementing the parallel application on the embedded OS. In order to avoid the unnecessary scheduling effort, we choose the multi-task application which is composed by a user executable program, a binary input data and system drivers i.e. the kernel command, drivers etc.. With proper

OS support, the host platform can easily move tasks between processors, in order to balance the workload efficiency.

The SNAPGEAR embedded Linux OS utilizes the 2.6 kernel which supports the multithreading applications for SMP systems. Every 200 milliseconds, the scheduler performs the load balancing in order to redistribute the task and maintain a balance across the processors. To understand how SMP is initialized for a given architecture, check out the smp.c or smpboot.c files within the kernel at ./linux/arch/<arch>/kernel/.

The kernel maintains a pair of run queues for each processor (the expired one and the active one). Each run queue supports 140 priorities, with the top 100 used for real-time tasks, and the bottom 40 for user tasks. Tasks are given time slices for execution, they move from the active run queue to the expired run queue. This provides fair access for all tasks to each core, and lock only on one task per core. There are two issues should be noticed during the Linux kernel design, synchronization issue and cache migration issue.

- Synchronization Issue

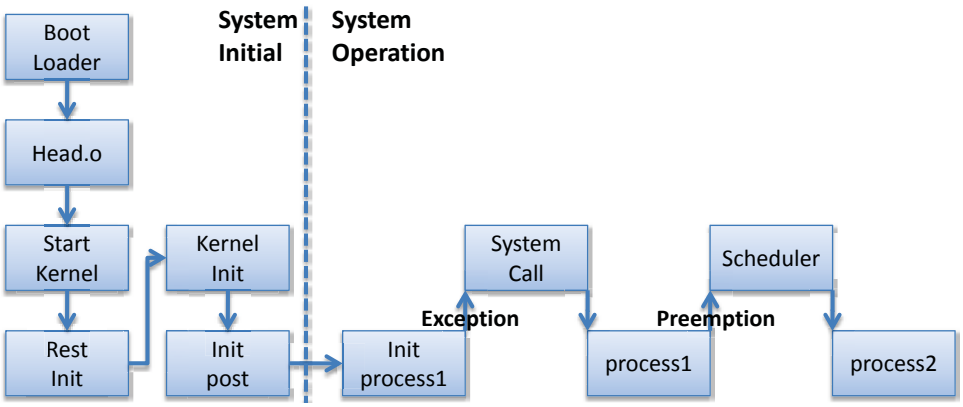


Figure 3.18: Linux kernel initializes the process and incurs the latency

SMP kernel is affected by the need for synchronization and locking of the resources. As shown in Fig. 3.18, while a process or a thread is waiting to obtain a lock, some of its cache lines maybe replaced, thus the thread is scheduled again, it may experience higher memory latency. In paper [1], the authors presented the problem that as the number of CPUs increase, the performance benefit from caches is partially offsets by the long latencies incurred when one CPU references data owned by another CPU‘s cache.

To keep the dcache synchronized with an external memory, cache snooping should be enabled. The dcache monitors write accesses on the bus to cacheable locations. If another master writes to a cacheable location which is currently cached in the dcache, the corresponding cache line is marked as invalid. In the multi-set configuration, the i/dcache controllers can be configured with a lock bit in the cache tag. Setting the lock bit prevents the cache line to be replaced

by the replacement algorithm.

- Cache Migration Issue

Table 3.V: The impact on the cache migration cost.

Bogo MIPS	Icache Size (KB)	Dcache Size (KB)	Migration Cost μs
31.84	2*8	2*8	40000
31.84	2*8	2*4	10000
64.81	2*4	2*4	10000

The migration cost is a parameter related with the cache configuration. The numbers specified in Table 3.V are measured in μs. It sets up an intra-core migration cost of 10 ms, 20 ms and 30 ms etc.. The BOGO MIPS is an unscientific measurement of CPU speed. It is made by the Linux kernel when it boots, in order to calibrate an internal busy loop. From the testing result (Table 3.V), we find out that the size of dcache determines the migration cost, and the size of icache determines the BOGO MIPS. In order to minimize the performance penalty due to the cache line migration, we need to decrease the size of dcache and icache to $2Set * 4KB$.

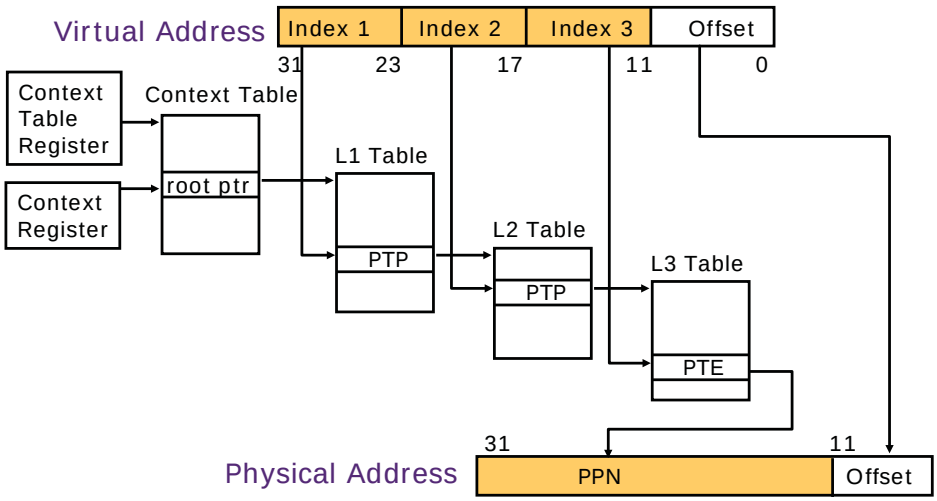


Figure 3.19: SRMMU virtual address and physical address mapping.

The reason of the large migration cost is that the SRMMU has 32-bit virtual address and 36-bit physical address and the fixed size of the page (4KB) as we mentioned before, also data snoopy protocols are the most appropriate solution for small to medium-scale shared bus multiprocessors. If MMU is disabled, the dcache is operated as normal with physical address mapping. If MMU and dcache snooping are enabled, the dcache tags store the virtual address, however, the physical tags store in a separate RAM that is used for snooping, which could lead to cache-migration issues when boots the Linux kernel on board as shown in Fig.3.19. Therefore, the size of the dcache has to be smaller or equal to 4 KB per set, refer to MMU page size.

3.4.3.3 CPU Configuration

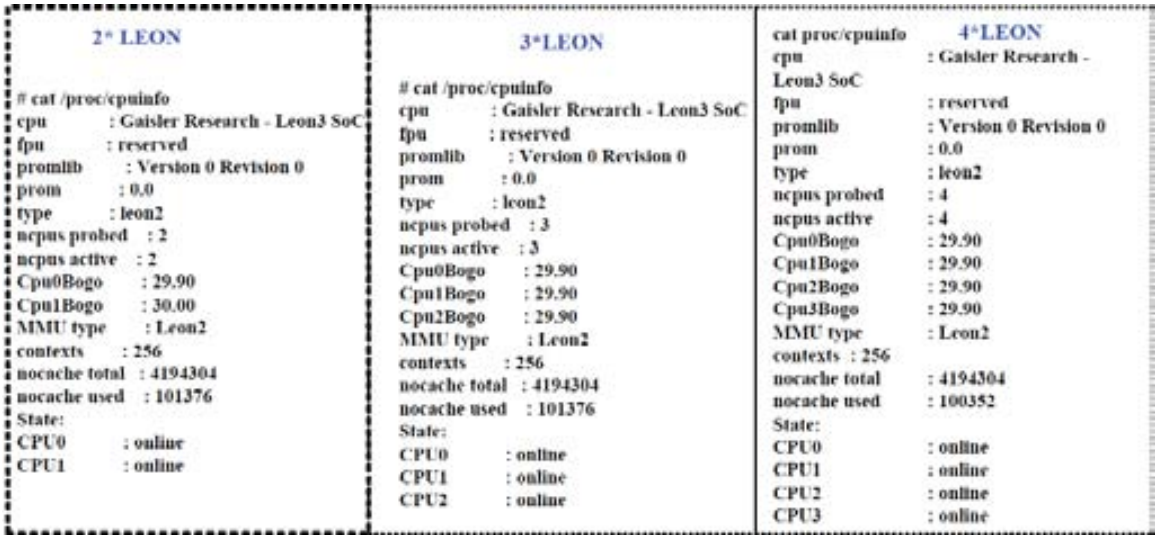


Figure 3.20: "CPU info" description on SMLOL platform with 2/3/4 cores.

To implement the embedded Linux kernel on the SMLOL hardware, the kernel must be properly configured. The *CONFIG_SMP* option must be enabled during the kernel configuration to make the kernel SMP aware. We can identify the number and the type of processors by the *proc* filesystem in the Linux kernel. Thus, we can retrieve the *cpuinfo* under the directory */proc* of Linux file system by the command *cat* below.

```
cat /proc/cpuinfo
```

Fig.3.20 presents the content of the *cpuinfo* file with 2/3/4 cores platform. We prove that the LINUX 2.6 kernel has been successfully installed on the SMLOL hardware, and the BOGO MIPS of each core is the same.

3.4.3.4 Ethernet Configuration

As shown in Fig.3.21, the demonstration platform needs an Ethernet cable to transfer waveform data from the CONTROL PC to the GR-CPCI-XC4V board. The IP address of the FTP server (installed on Control PC) is (*192.168.0.1*), netmask (*255.0.0.0*) and route (*127.0.0.1*). And the IP address of the FTP client (installed on GR-CPCI-XC4V board) is (*192.168.0.80*), netmask (*255.0.0.0*) and route (*127.0.0.1*). After configuring the CPU and designing the user application, the next step is to configure the Ethernet in the Linux kernel.

The first step is to create the FTP client in the Linux kernel, and install the ftp server on the Control PC to store the input waveform. To create the FTP client in the Linux kernel, we need to configure the Linux kernel by the networking service

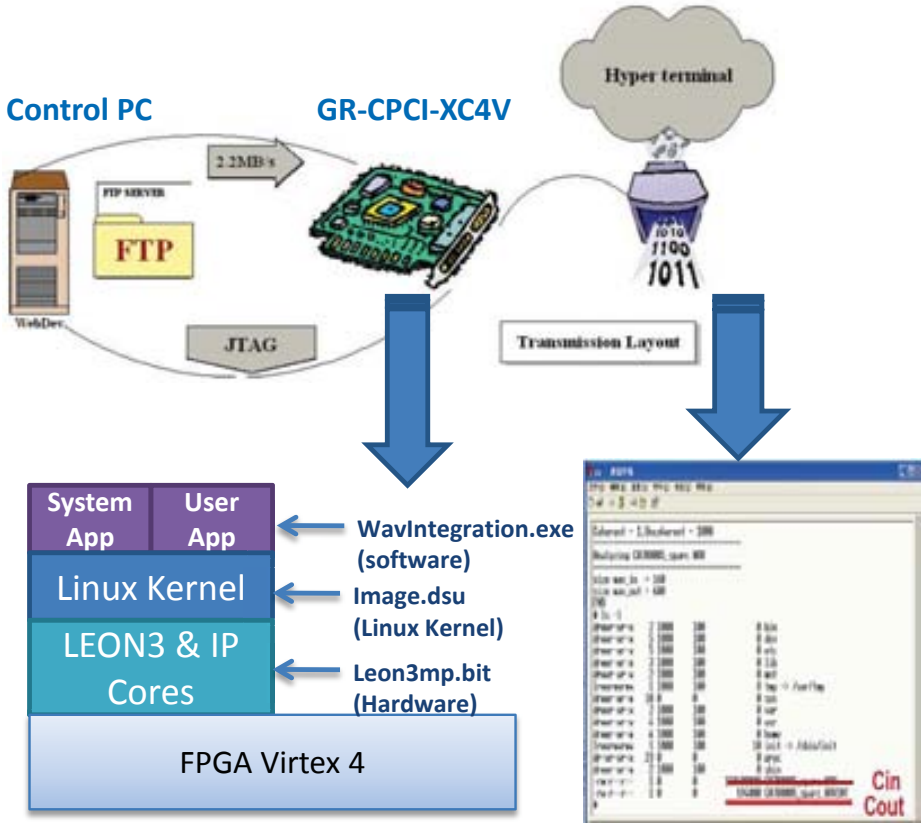


Figure 3.21: Ethernet configuration.

(TCP/IP), which is enabled for the FTP application. Then we configure the driver of the ethernet device (LAN91c111), set the base address of the ethernet MAC to *0x20000300* to avoid conflicts with other cores, and set the MAC address with the original MAC address of the chip. Since the ethernet chip is connected to the GPIO port of the LEON3 system, the IRQ signal of GPIO is set to *0x5*, in order to determine the priority of the interrupt. For the details of the configuration, you can refer to the Snapgear for LEON manual.

The SMLOL platform is composed by the Snapgear embedded Linux kernel, external SDRAM and GRETH, which is installed on GR-CPCI-XC4V board. GRETH provides an interface between AMBA/AHB bus and Ethernet Media Access (MAC) interface. It is supported by IEEE standard 802.3 Local Area Network (LAN) protocols with 10/100Mbit speed in both full-duplex and half-duplex. Two data rates are currently defined for the operation over the twisted-pair cables. The MAC controller with AMBA AHB host interface receive and transmit the data autonomously by DMA transfer, without the CPU involvement.

As shown in Fig.3.22, File Transfer Protocol (FTP) provides a method for copying files over a network from one computer to another. User Datagram Protocol (UDP) is connectionless protocol, since you don't know if the transmitted data or message will get the destination or get lost on the way. There may be corruption while transferring

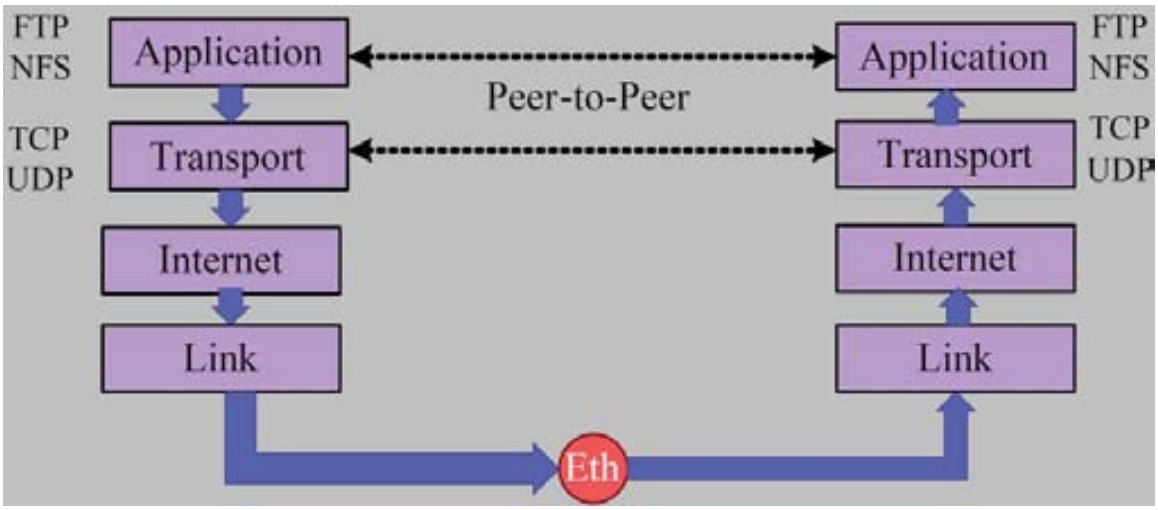
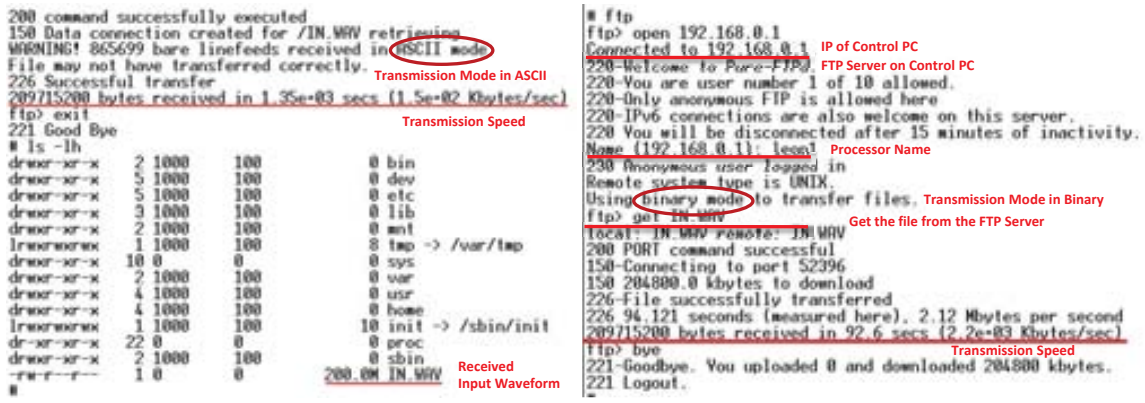


Figure 3.22: FTP and UDP protocol.

a message. FTP uses Transmission Control Protocol (TCP) protocol because the file transfer has to be correct. UDP is good for the high speed, but not everything will get to the destination. The main point is to guarantee the communication path that remove its receive data faster than they arrive in the buffer. Therefore, in our case, we choose FTP as the transmission protocol.



a) ASCII Transmission

b) Binary Transmission

Figure 3.23: FTP transmission time.

After saving all the configuration of the Linux Kernel, we can build the image and load it into the SDRAM on the board. The image file (image.dsu) is downloaded at the address *0x40000000* by GRMON. Then the LEON3 processor will start operation from this address, we can check the boot sequence on the Hyper terminal. We can enter the commands of the Linux by the Hyper terminal after displaying the Linux booting sequence; connect and login to the FTP server; and get the file (*IN.WAV*) from the FTP server (Control PC) to the FTP client (GR-CPCI-XC4V board). As shown in Fig.3.23, the input waveform file (IN.WAV, 209.7152MB) transmits from the ftp server

(Control PC) to ftp client (GR-CPCI-XC4V board). The speed can reach up to 150KB/s by ASCII transmission and 2.2MB/s by binary transmission. It is quicker to transmit the waveforms in binary mode.

3.4.4 Multi-task Application and Timing Performance

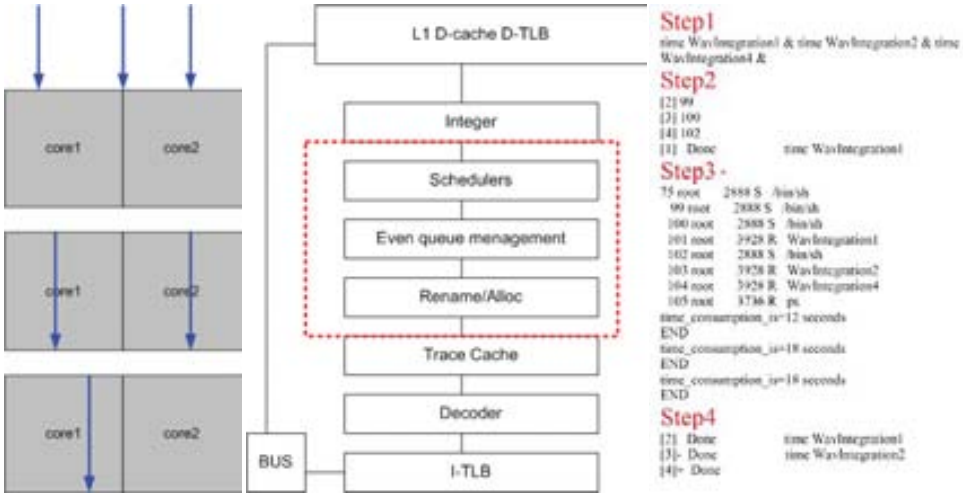


Figure 3.24: The scheduling of three parallel tasks execute in SMLOL with 2 cores.

In the task-level parallelism design, if two applications run in parallel on a two cores platform, it can enhance the execution time as a factor of 2. As shown in Fig.3.24, If we got three tasks work in parallel on a two cores platform, only two tasks could work at the same time, which takes 18 s. Another one will be executed in sequence, which takes 12 s. The speedup ratio turns to be 1.8. It illustrates that two cores working together could produce the cache locking issue as we demonstrated in the Subsection 3.4.3.2. This process is managed by the scheduler in Linux kernel, therefore, we can not identify which core is running and which one is idle.

In order to get the timing performance of this multi-task application, we need to understand how to implement this post-processing algorithm on the Linux kernel of SMLOL platform. In the Linux kernel design, we need to compile the post-processing applications (WavIntegration1, WavIntegration2 etc.) into the Linux image file, and configure the custom application from the busy box. Busy box is a single binary but a collection of many standard UNIX tools. It is very suitable for the embedded Linux application. In BusyBox, the *init (inittab)* option is selected, as the first script to be executed by the Linux kernel. The initial scripts of Linux kernel on SMLOL system is shown below:

```
1
2 mount -t sysfs none /sys                    -- mount the file system
3
4 mount -t devpts devpts /dev/pts             -- mount the device
5
6 hostname leon3                              -- define the host name
```

```
7
8                                                  -- configure ethernet
9
10 /sbin/ifconfig lo up 127.0.0.1 netmask 255.0.0.0
11
12 /sbin/ifconfig eth0 up 192.168.0.80
13
14 route add 127.0.0.1 dev lo
15
16 route add default dev eth0
17
18 ls -l                                           -- assign the input data
19
20 cp C07B0005_sparc.WAV 1.wav
21
22 cp C07B0005_sparc.WAV 2.wav
23
24
25 ls -l                                    -- execute the user application
26
27 (echo 1)>foo1.txt
28
29 (time WavIntegration1)2>>foo1.txt
30
31 echo 1
32
33 (echo 2)>>foo1.txt
34
35 (time WavIntegration1 & time WavIntegration2)2>>foo1.txt
36
37 echo 2
```

There are three issues that need to be declared: 1) The IP address (192.168.0.80) defined in the Ethernet Configuration is the IP of the GR-CPCI-XC4V board, which is different with the one of Control PC. 2) The input data is simply created by duplicating the waveform file (C07B0005_sparc.WAV). In reality, we get the input data by the FTP transmission from the Control PC to GR-CPCI-XC4V board (details refer to the next subsection). 3) We execute the post-processing applications (WavIntegration1, WavIntegration2 etc.) in parallel by "&" command, and "printf" the execution time into the *foo1.txt* file.

No.	Number of Parallel Application																	
LEON3	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	11.264	22.46	33.954	45.221	56.425	67.544	78.363	90.083	100.972	112.735	123.607	134.438	145.84	156.846	168.04	178.233	189.428	200.4
2	11.224	11.092	19.468	23.908	30.159	36.074	44.643	48.213	55.266	59.959	73.971	74.946	78.919	82.472	94.744	97.551	105.213	107.4
3	11.333	11.98	13.959	21.117	28.818	31.003	36.222	43.097	44.819	49.006	50.979	55.25	63.864	62.43	65.862	73.46	74.051	78.3
4	11.372	11.907	13.037	14.871	27.745	24.966	25.768	31.395	42.428	38.686	41.368	46.321	48.872	60.907	64.122	62.049	72.23	69.6
5	11.421	12.027	13.265	14.975	17.338	23.255	21.049	31.895	32.308	36.568	51.609	54.636	50.05	50.344	52.7	56.125	59.772	63.5
6	11.47	12.081	13.402	15.151	17.55	20.639	27.756	28.571	45.366	50.896	47.066	43.221	50.936	51.976	56.843	68.34	63.534	62.4

Figure 3.25: Execution time (s) of parallel application on SMLOL @ 60MHz.

The parallel applications (N_a) run with different numbers of cores (N_p) on SMLOL. The execution times of each parallel application is shown in Fig.3.25. Note that the captured execution time has three types of parameters as shown in Table.3.VI, in our

Table 3.VI: Three types of execution time (s).

real	Elapsed real (wall clock) time used by the process.
user	Total number of CPU-seconds that the process used directly in user mode.
sys	Total number of CPU-seconds used by the system on behalf of the process in kernel mode.

case, we choose the real time as the measurement results. From the above results, we can extract three important parameters: speedup ratio, system throughput and the standard deviation of execution time.

- Speedup Ratio

  Amdahl's law is used in parallel computing to predict the maximum speedup by the multiple processors platform. The speedup of a program is limited by the time of the sequential fraction of the program. Amdahl's law states that the potential program speedup is defined by the parallel fraction which can be parallelized:

$$Speedup = \frac{1}{1-P}, \tag{3.18}$$

  Therefore, if a small portion of the program which cannot be parallelized, it will limit the overall performance. However any large mathematical or engineering problem will typically consist of several parallelizable parts and several non-parallelizable parts. This relationship can be given by the formula:

$$Speedup = \frac{1}{\frac{P}{N}+S}, \tag{3.19}$$

  where:

  P is defined by the parallel fraction of code which can be parallelized.

  N is defined by the number of processors.

  S is defined by the serial fraction of code which can not be parallelized.

Table 3.VII: The speedup parameter for multi-processors.

N	1	2	3	4	5	6
$speedupratio$	1	1.78x	2.23x	2.58x	2.71x	2.7x
P	0	0.876	0.826	0.817	0.789	0.756
S	1	0.124	0.174	0.183	0.211	0.244

The *Speedup ratio* is calculated by the ratio of the execution time on multi-core platform and the execution time on the single-core platform. Table 3.VII shows the extracted average *speedup ratio,* which is tested on the different number of processors *N* in the platform. The 2 cores platform attains 1.78x mean speedup

ratio over a single core baseline. However the 6 core platform can only attains 2.7x mean speedup ratio over a single core baseline. Through the Equation 3.19, we can calculate the parallelizable parts *P* and the non-parallelizable parts *S* of the programs in the multi-task application.

- Maximum System Throughput

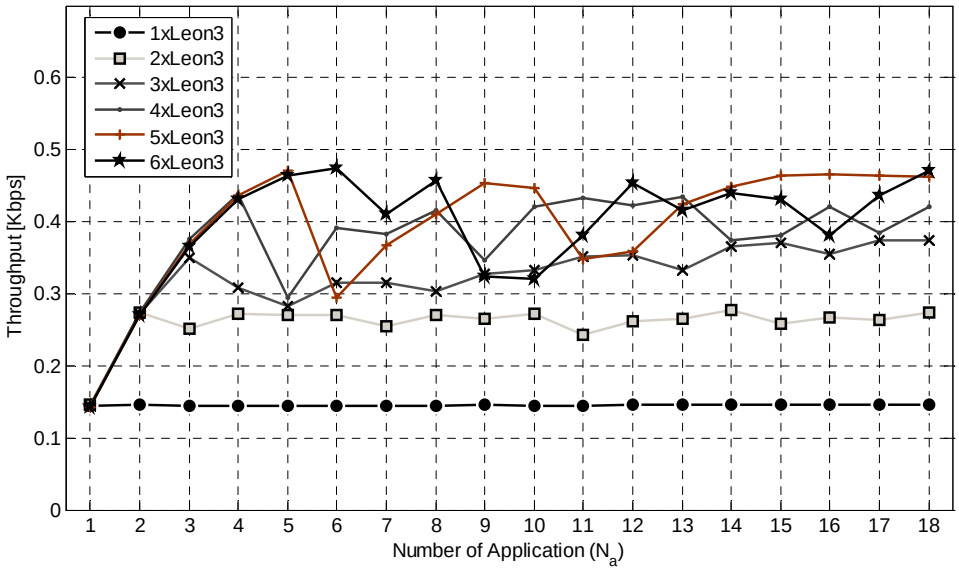


Figure 3.26: System throughput v.s. parallel applications at multi-core platforms.

In the multi-task application, the number of parallel applications N_a and the number of processors N_p will determine the maximal system throughput. As shown in Fig. 3.26, with the increasing of the parallel applications, the system throughput varies in the different multi-core platforms. It clearly shows that the maximum system throughput always appears at the point that when N_a is equal or a multiple to N_p. It means that the processors have the maximum utilization ratio at these points. When the N_a is less than N_p, the system throughput has an upward trend, when the N_a is greater than N_p, the lines began to fluctuate, and this volatility is increasing with the value of N_p. That means as the number of cores increase, the performance benefit from additions cores is offsets by the other components. Therefore, adding more cores to an SMP system does not increase throughput since workloads cannot always take advantage efficiently of multi-core.

- The Standard Deviation of Execution Time

As shown in Fig.3.27, with the increasing number of the parallel applications, the execution time is increasing linearly. With the increasing of the number of cores, the real value of the execution time gets more and more difference than the ideal one. That means the six core platform should get the speedup ratio as 6 under ideal circumstances. However, in reality, it is far away from we expected.

Fig.3.28 shows that there is an evidently lag of execution time as the increasing of N_p. We use σ to represent the standard deviation of the execution time, it

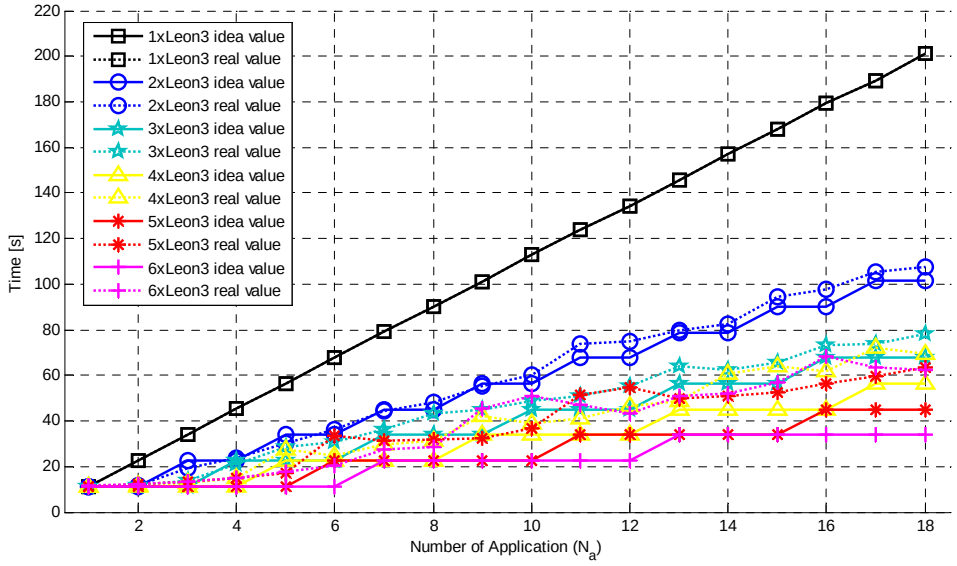


Figure 3.27: Execution time v.s. parallel applications at multi-core platforms.

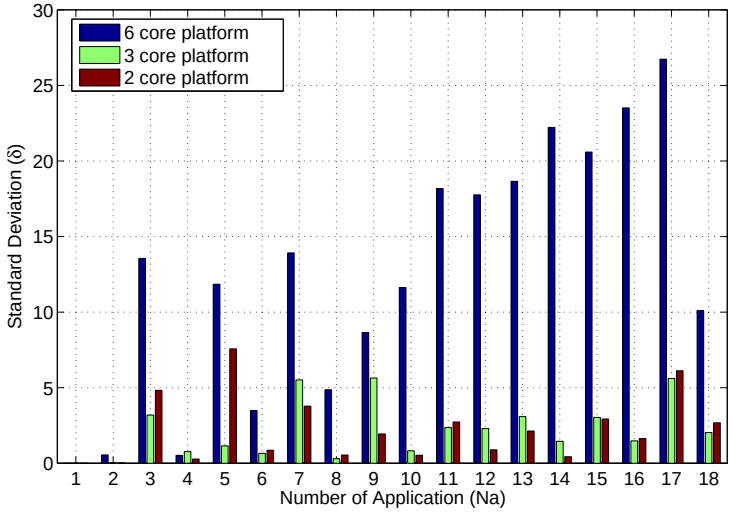


Figure 3.28: Standard deviation of the execution time with 2 core, 3 core and 6 core platforms.

presents how much variation of the execution time from the average time. The equation is shown below:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{3.20}$$

where the standard deviation σ is the squre root of the average value of $(x-\mu)^2$, x is the execution time measured on the multi-core platform of a finite data set x_1, x_2, ..., x_n. μ is the mean value of the execution time x.

A low standard deviation (<1) indicates that the data points tend to be very close to the mean, whereas high standard deviation indicates that the data are spread out over a large range of values. The value of the standard deviation is 7.802s for 2 core platform, and 9.025s for 3 core platforms, even 19.105s for 6 core platforms compared to the non-parallel platform. The increasing of the lag is dramatic.

3.5 MPARM Simulation

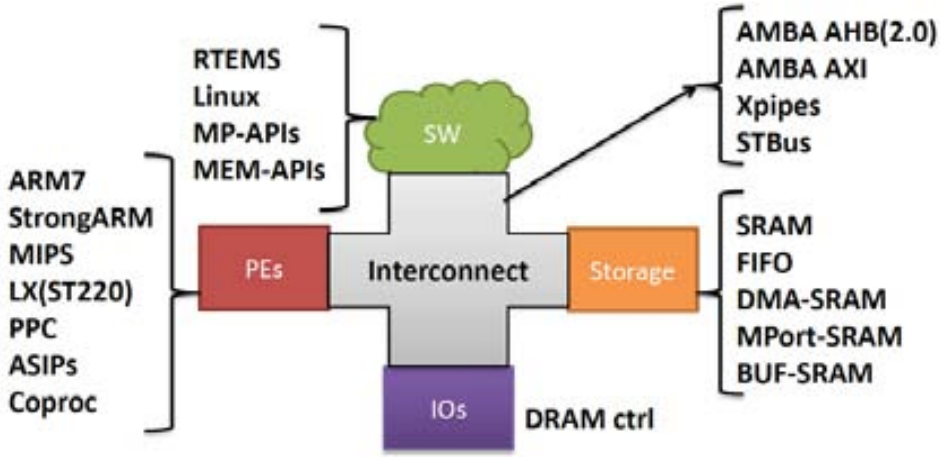


Figure 3.29: MPARM scheme.

In the Multi-task Application design, we found out that the standard deviation of the execution time and speedup ratio can't reach our timing requirement as a factor of 3. In this Section, we provide an emulation platform, MPARM. It allows the simulation of innovative memory hierarchy architectures, also help us optimize the system performance (i.e. cache-memory gap, bus busy ratio, memory access time) and tackle on the bottleneck over the SMLOL.

We implement a series of simulations based on the MPARM and uClinux simulator [*MPARM*]. MPARM is a multi-processor cycle-accurate architectural simulator, used to analysis the system-level design tradeoffs in the usage of different processors, interconnects, memory hierarchies and other devices. The output of MPARM includes accurate profiling of system performance, execution traces, signal waveforms, and power estimation. The MPARM platform includes HW/SW components as shown in Fig.3.29. Currently, a port of the RTEMS OS runs on MPARM, and a uClinux port is well underway. Based on uClinux OS (2.0 non-mmu) and different memory hierarchies, we can get the optimized timing parameters through the simulations.

3.5.1 Tackle on the Bottleneck of SMLOL

As we known, the SMLOL platform is a bus-based SMP architecture. It integrates multi-LEON3 processors with their own caches and MMUs, but shared the main memory. It is easy to build up the higher OS level and SW programming. The processors are connected through a shared AMBA bus. The bus guarantees that a memory operation on the L2 cache or the main memory must be completed before the next bus transaction taking place. There is a private SRMMU for each LEON3 processor [*Linux for LEON processors*], which maintains the processor access to the shared memory. We put the post-processing applications into the uClinux and simulate with different type of architectures at 200MHz system frequency. Through simulation, we can optimize the memory hierarchy and get the HW parameters, i.e. bus busy ratio, bus access time, cache hit and miss ratio, computing clock cycle, memory latency etc.. Also we can accurately measure the overall execution time of the post-processing application. The simulations are carried out under the following assumptions:

- Cache memory gap: the gap between the cycle time of processors and data exchange time of cache-memory has continuously increased in SMLOL, due to the overhead involved in the shared resource.

- Memory bus gap: the gap between the low memory operation time on the external memory and the time consumption of the bus transaction. According to the parallel workload analysis [8][9], applying the L2 memory in the memory hierarchy can get rid of the bus busy ratio and memory access time.

3.5.2 Simulation Results

Table 3.VIII: Simulation results by MPARM.

Number of Platform	P1	P2	P3	P4	P5	P6	P7	P8	P9
Number of cores	1	1	1	4	1	1	1	1	1
D-cache	4k*4	4k*4	4k*4	4k*4	4k*4	4k*4	4k*4	4k*4	4k*4
I-cache	8k*2	8k*2	8k*2	8k*2	8k*2	8k*2	8k*1	8k*2	8k*4
D-cache miss (%)	1.10	2.42	1.10	1.10	2.29	2.29	2.29	1.74	2.29
I-cache miss (%)	1.92	0.91	1.92	1.92	0.95	0.95	1.49	0.91	0.95
L2 memory	2MB	NON	2MB	2MB	NON	NON	NON	NON	NON
L3 memory	1MB	1MB	1MB	1MB	1MB	1MB	1MB	1MB	1MB
Overall time(s)	0.407	0.133	0.207	0.24	0.066	0.134	0.143	0.133	0.134
MP ratio	13	13	13	13	2	13	13	13	13
bus access	76128	4482493	38064	159377	4490505	4490562	4645920	4454921	4490562
bus busy(%)	4.34	61.8	2.17	8.50	22.5	61.9	63.9	61.4	61.9

- Simulation 1: L2 cache effects bus busy ratio

  As shown in Table.3.VIII, Platform One (P1) and Platform Two (P2) are based on 1 core MPARM with uClinux OS at 200 MHz system frequency. Both of them

have 4-set Dcache (4KB/set) and 2-set Icache (8KB/set), and 1MB L3 private memory, and the ratio between the memory delay and the processor delay are 13. Based on uClinux OS, both of the platforms are processing 80KB input data by the same post-processing application. The difference is that P1 has 2MB L2 memory, but P2 does not have.

The objective of this simulation is to analyze how L2 cache effects bus busy ratio. In this regard, P1 with L2 memory can imply a lower bus utilization but with a higher execution time. However, P2 is experienced the contrary effect. As we described before, the L3 memory is a private memory, the gap between the memory-bus is partially offset by the L2 cache, therefore, the L2 memory can get rid of the burden of bus transmission dramatically. However, adding L2 memory in the platform can effect the processing speed of L1 Icache slightly, due to the longer routine.

- Simulation 2: 1 core v.s. 4 core effects bus access time

  As shown in Table.3.VIII, Platform Three (P3) and Platform Four (P4) are based on 1 core MPARM and 4 core MPARM separately. Both platforms are operating with uClinux OS at 200 MHz system frequency. Both of them have 4-set Dcache (4KB/set) and 2-set Icache (8KB/set), and 1MB L3 private memory. The ratio between the memory delay and the processor delay are both 13. The difference is that P3 is processing 40KB input data, but P4 is processing 160KB input data by the same post-processing application based on uClinux OS. The idea is that each core will process the same quantity of the input data.

  The objective of this simulation is to compare how the additional cores affect the bus access time. In this regard, the P4 with 4 cores will experience more than four times the bus access time with respect to P3. However, the overall execution time is almost the same. Therefore, the multi-core solution can be an efficient solution to enhance the timing features of the system. We only need to consider the capacity of the bus utilization.

- Simulation 3: The ratio of *memory delay/processor delay* effects the bus busy ratio

  As shown in Table.3.VIII, Platform Five (P5) and Platform Six (P6) are based on 1 core MPARM with uClinux OS at 200 MHz system frequency. Both of them have 4-set Dcache (4KB/set) and 2-set Icache (8KB/set), and 1MB L3 private memory. Based on uClinux OS, both of the platforms are processing 80KB input data by the same post-processing application. The difference is that the ratio between the memory delay and the processor delay (M/P) in P5 is 2, but the P6 is 13. The idea is that P5 will use the fast L3 memory than P6.

  The objective of this simulation is to compare how the memory access time affects the bus bus ratio. In this regard, the bus of P6 will be four times busier than P5 since it has a slow L3 memory. Therefore, choosing an on-chip L3 memory can reduce the bus busy ratio compare to the off-chip L3 memory.

- Simulation 4: Cache association effects Icache miss rate

As shown in Table.3.VIII, Platform Seven (P7), Platform Eight (P8) and Platform Nine (P9) are based on 1 core MPARM with uClinux OS at 200 MHz system frequency. Both of them have 4-set Dcache (4KB/set) and 1MB L3 private memory. The ratios between the memory delay and the processor delay are 13. Based on uClinux OS, all of the platforms are processing 80KB input data by the same post-processing application. The difference is that the Icache associations (8KB/set) of P7, P8, P9 are 1-set (direct map), 2-set, 4-set separately.

The objective of this simulation is to compare how the Icache association affects the Icache miss rate. With the increasing sets of the Icache from P7-P9, the P8 can attain the minimal Icache miss. Indeed, Icache miss ratio is an important factor in the overall execution time. Therefore, 2-set Icache association can attain higher timing performance.

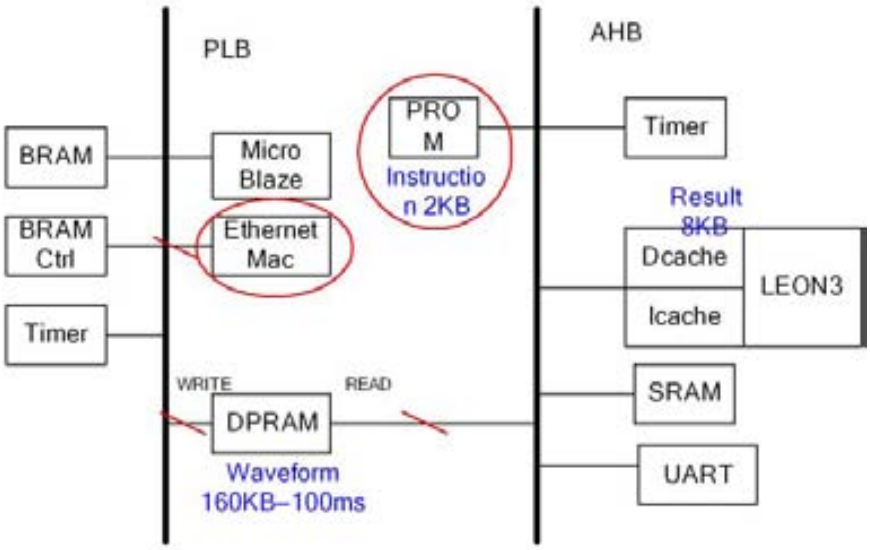


Figure 3.30: MB and LEON3 dual-core architecture.

After a series simulations of MPARM and uClinux, we can conclude that the bottleneck of the timing requirement in SMLOL is the bus busy ratio and the external memory access time. It also exist the risk of the confliction between transmission and processing. Therefore, memory allocation issues will also need to be considered. Fig.3.30 gives the preliminary idea about how to use two buses and L2 memory to solve the memory allocation issue, in order to get rid of the confliction between transmissions and processing, memory access time and the bus busy ratio.

3.6 Summary

In this Chapter, we have proposed the hardware design, software and OS design of the SMLOL platform. We have introduced the SMLOL architecture the hardware parameters. Then combined with the workload analysis of the GNSS-R post-processing application, we present the embedded Linux OS and SW design. Two

issues had been discussed: synchronization issue and cache migration issue. A multi-task application is implemented on SMLOL, in order to get the transmission time and the computing time. Finally, we introduced a virtual platform, MPARM, aims at optimizing the system performance and tackle on the bottleneck of the SMLOL platform.

This Chapter proposed a preliminary idea to speculate the software/hardware efficiency. And afford on the simulation of different levels design (processor, memory, bus) reveal that how the pipeline stall affect the system throughput in parallel system. Comparison to the affection of different components, memory hierarchy design could be the hotpot for solving the timing degrading in parallel system.

In parallel system, speedup ratio refers to how much a parallel algorithm is faster than a corresponding sequential algorithm. Ignore the relative cost of the system, we compared the execution time with the increasing parallel application based on multi-task application. We found out that the standard deviation and speedup ratio can't reach our timing requirement. In order to solve the synchronization issue and cache migration issue in the system level, we need to speculate it in the hardware level by changing the memory hierarchy and communication system. Therefore, in the next Chapter, we will introduce a novel architecture, HTPCP, in order to realize the post-processing algorithm in real-time.

Bibliography

[1] J. Black, D. Wright, and E. Salgueiro, “Improving the performance of oltp workloads on smp computer systems by limiting modified cache lines,” in *Proceedings of IEEE International Workshop on Workload Characterization (WWC’03).* IEEE, 2003, pp. 21–29.

[2] J. Kim and M. Ryu, “Aprix: A master-slave operating system architecture for multiprocessor embedded systems,” in *Proceedings of 12th IEEE International Workshop on Future Trends of Distributed Computing Systems*. Kunming, China: IEEE, 2008, pp. 233–237.

Web Reference

[*Linux for LEON processors*] http://www.gaisler.com

[*grlib*] https://www.gaisler.com/products/grlib/grlib.pdf

[*grip*] http://www.gaisler.com/products/grlib/grip.pdf

[*sourceForge*] http://sourceforge.net/

[*GRMON User's Manual*] https://www.gaisler.com/doc/grmon.pdf

[*Snapgear Linux for LEON*] https://www.gaisler.com/products/linux.html

[*GR-CPCI-XC4V LEON Compact-PCI development board*] http://www.gaisler.com

[*MPARM*] http://www-micrel.deis.unibo.it/sitonew/research/mparm.html

[*MicroBlaze processor*] http://www.xilinx.com/tools/microblaze.htm

[*IBM PowerPC 405 core*] www.xilinx.com/support/documentation/user_guides/ug018.pdf

[*Snapgear for LEON manual*] ftp://gaisler.com/gaisler.com/linux/linux-2.6/snapgear

[*SANDIA REPORT*] http://prod.sandia.gov/techlib/access-control.cgi/2008/086015.pdf

Part IV

Parallel System Design Based on HTPCP

Parallel System Design Based on HTPCP

4.1 Introduction

In this Chapter, we focus on the design of the novel parallel platform, Heterogeneous Transmission and Parallel Computing Platform (HTPCP). This platform is presented to realize the post-processing algorithm for GNSS-R application. Moreover, two problems are proposed and solved, 1) Parallelize the inherent serial output of the GOLD-RTR instrument by Transmission Elements (TEs) and 2) Post-processing the multi-channel (I and Q) correlators in parallel by Processing Elements (PEs). In order to guarantee the real-time characters and minimize the memory requirement in space level, we operate the same post-processing algorithm as mentioned in Chapter III.

This Chapter is organized as follow:

- Description of the HTPCP architecture.
- Survey on hardware design of HTPCP, including the TEs/PEs design and the design flow of HTPCP.
- Introduction to the seven software routines, and verify the correctness of IP core designs in HTPCP.
- Main conclusions.

4.2 HTPCP Architecture

In this Section, we discuss the functional block diagram of HTPCP. HTPCP includes two types of elements: Transmission Elements (TEs) and Processing Elements (PEs) presented in Fig.4.1. The TE is applied to parallelize the inherent serial output of GOLD-RTR. The PE is used to realize the post-processing algorithm for each correlation channel, and hence, to reduce the amount of waveforms prior to downlink through the satellite. The real-time geophysical parameters transmit from the GOLD-RTR to HTPCP by ethernet interface, and distribute to 2 TEs and 4 PEs in HTPCP.

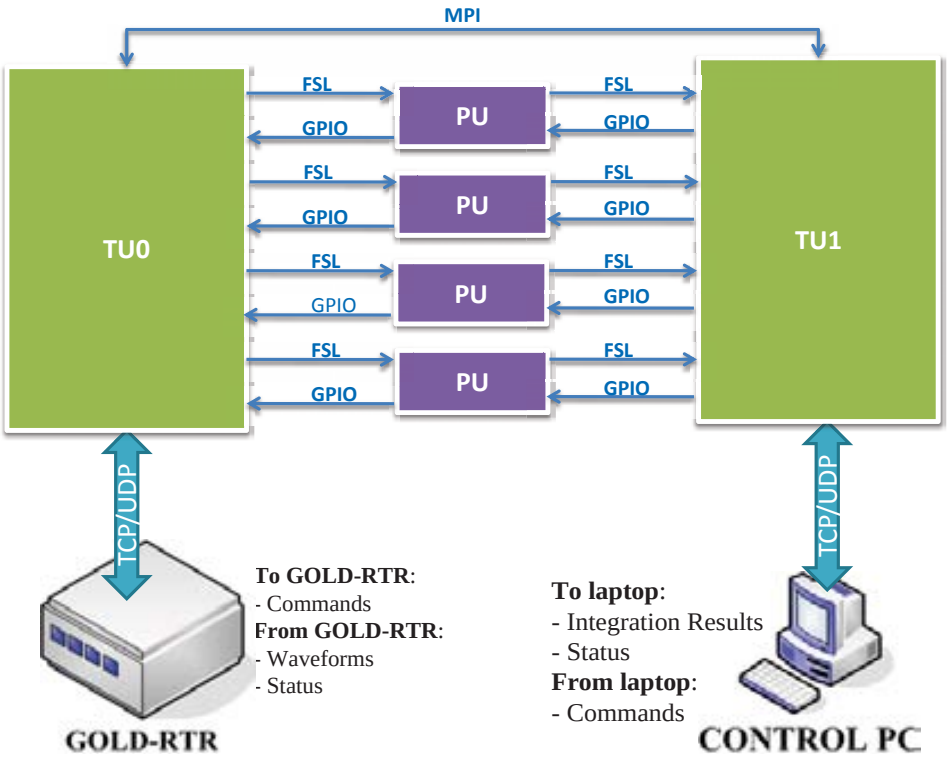


Figure 4.1: HTPCP block diagram.

Each PE processes the consecutive 1 ms waveforms for each channel. Meanwhile each TE accomplishes two tasks: data storage and data distribution. Considered about the different function, each element features its private memory and bus, which are accessible from the local processor. The memory hierarchy of TEs and PEs are distinctly, the size of private memory is variant with the size of bandwidth, and the software framework can be applied to different stacking approach.

The interconnections between TE and PE are done via Fast Simplex Link (FSL), which is composed by a non-latency register-to-memory single direction communication path. The interconnections between TE and TE are done via a Massage Passing Interface (MPI) to realize the transmission between GOLD-RTR and Control PC. The MPI is capable of receiving commands coming from remote processor and passing them to the other remote processor transparently, in order to synchronize the package transmissions with the local traffic. The block diagram of the HTPCP is shown in Appendix H.

All these features make it possible to leverage the workload between the communication and computing, and flexibly assign the other applications in the PE in future, for instance, to calculate the sea surface mean-square slopes, ice roughness and thickness, soil moisture and biomass etc..

4.3 HTPCP Hardware Design

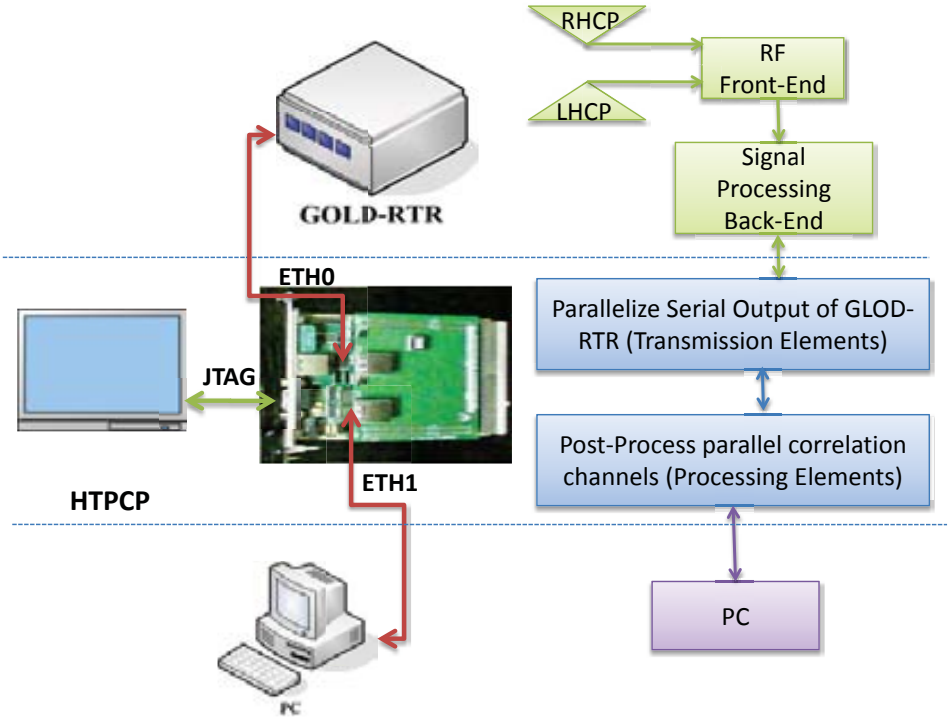


Figure 4.2: Work schematic of GNSS-R application.

The parallel system becomes increasingly important in high performance computing, but the success of this paradigm heavily relies on the efficiency and widespread diffusion of parallel software. However the unbalanced workload in the hardware design can not be fundamentally resolved through software methodologies. This reminds us to design an optimized memory subsystem in the hardware domain, in order to balance the transmission and computing workload.

Fig.4.2 depicts the work schematic of HTPCP. The GOLD-RTR instrument is composed by two physical devices: a rack that contains the front- and back-end electronics of GOLD-RTR and a Control PC which provides control and monitoring functions of the rack electronics as well as disk storage to record waveforms. As an intermediate, HTPCP board communicates with GOLD-RTR and Control PC through two full-duplex ethernet links at 10/100 Mbps. HTPCP is composed by **GR-CPCI-XC4V** LEON compact-PCI development board plus its extension board **GR-CPCI-2ETH-SRAM-8M**. The Xilinx Virtex-4 LX200 FPGA on this board is well suited and configurable for LEON3 core implementations.

The hardware design of HTPCP is composed by two elements: the design of Transmission Elements (TEs) and the design of Processing Elements (PEs). As two key components, the first one is based on the MicroBlaze (MB) system, mainly responsible for the data transmission; while the second one is based on the LEON3 system, mainly in charge of the parallel processing task. In order to guarantee real-time parallel

processing performance, we focus on the integration of two processor systems (LEON3 and Microblaze) in Subsection 4.3.3.

4.3.1 Transmission Elements

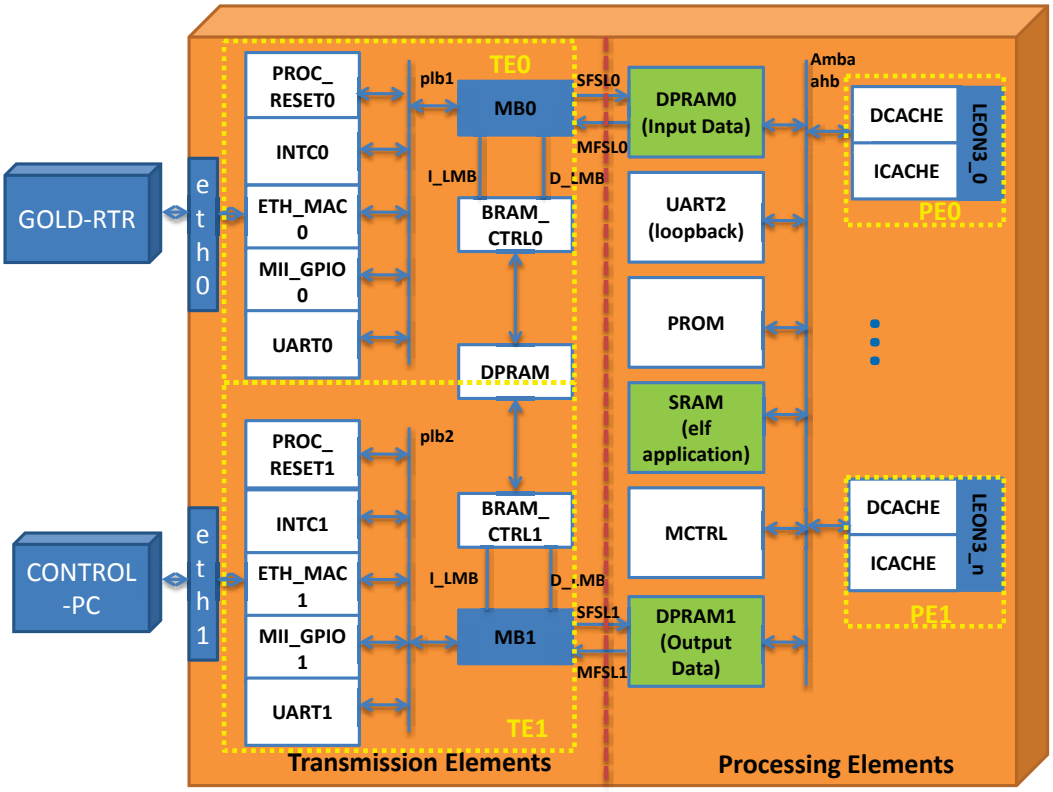


Figure 4.3: Transmission diagram.

By taking into account of the characteristic of the parallel system, the time spent on actual data processing is mostly predictable but the communication and data transfer times are rather dynamic by the nature. The key roles of TEs are to solve the following two functions:

- Forward transparently the control commands and monitoring packets between the Control PC and the GOLD-RTR rack back and forth.
- Distribute the received waveforms to each PE based on the number of corrector; collect the results from each PE and send them to the control PC.

The transmission diagram is shown in Fig.4.3, where two blocks of TEs (TE0 and TE1) are considered based on two Microblaze systems. On one side, they are connecting with two instruments as the transmission controller; on the other side, they are connecting with many PEs as the data distribution controller. To better understand their work principle, we focus on the three issues as follows:

- The operation flow of TE0 and TE1 to illustrate the time sequence of the commands and waveforms transmission.
- Description to the UDP transmission protocol and the package frame.
- The TE/TE interface design - Massage Passing Interface (MPI).

4.3.1.1 Operation Flow of TEs

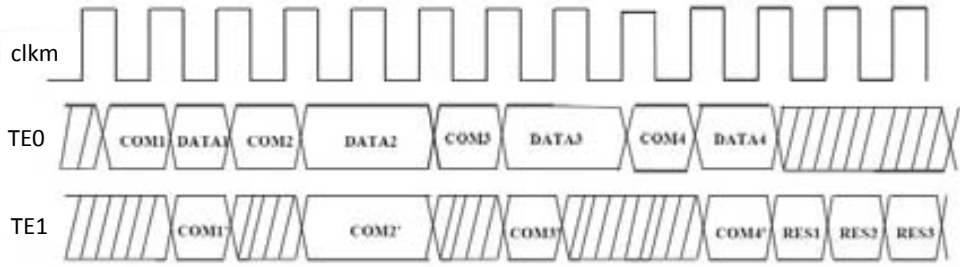


Figure 4.4: Operation flow of transmission elements (TE0 and TE1).

Fig.4.4 provides a graphical interpretation to the operation flow of TE0 and TE1 as an mediator to transmit the commands and waveforms between GOLD-RTR and Control PC.

- GOLD-RTR sends command signal (COM1) to TE0;
- TE0 transmits the COM1 to TE1, and TE1 forwards it to the Control PC transparently;
- Until the Control PC receives COM1, it will reply to TE1 a response command signal (COM1');
- TE1 transmits the COM1' to TE0, and TE0 forwards it to the GOLD-RTR transparently;
- Then, the GOLD-RTR sends the waveform packets to TE0 continuously (5 waveforms per packet).
- TE0 distributes these waveforms to each PE corresponding to the channel number;
- PEs process the waveforms and generate the results.
- TE1 collects the results of PEs and send them to the control PC every 1 s.

The throughput of TE0 degrades from 12.8 Mbps to 1.28 Mbps after distributing the waveforms. The function of post-processing algorithm is trigged by the entry address pointer, and generate the result by the end address pointer. The throughput of TE1 is 5.248 Kbps.

4.3.1.2 Transmission Protocol and Frame

UDP for data transfer is used for bulk transfer, it only provides application multiplexing (via port numbers) and integrity verification (via checksum) of the header and payload. The transmission protocol fulfils by TCP/UDP protocol. Depending on the type of information, the length of the payload can be changed 1 B (minimum) to 1500 B (maximum) as shown in Fig.4.5. The first byte of the payload indicates the type of information contained in the packet. Appendix F shows the possible values of the first Byte, and their meanings transmitted from control PC to HTPCP. Appendix G shows the possible values of the first Byte, and their meaning transmitted from GOLD-RTR to HTPCP.

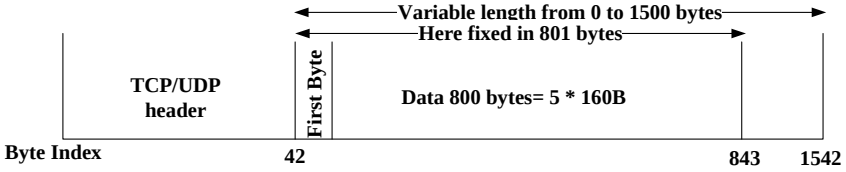


Figure 4.5: TCP/UDP package frame.

Each TCP/UDP packet contains 5 waveforms (as payload), and the transmission takes around 45 us. Each waveform contains the number of correlation channels, which could be selected by TE0 to distribute to each PE. We can observe the transmission time and the package frame by Wireshark software as shown in Fig.4.6. In this case, we receive a UDP packet sent from the GOLD-RTR to the control PC. The IP address of the GOLD-RTR is 10.0.0.2 with MAC address (06:05:04:03:02:01), and the IP address of the Control PC is 10.0.0.51 with MAC address (00:1f:16:20:d4:ca). The frame size is fixed with 1070B. From the first byte of the payload (‘O‘ alphabet), we learn that this command packet is sent by GOLD-RTR to check the communication is OK with control PC.

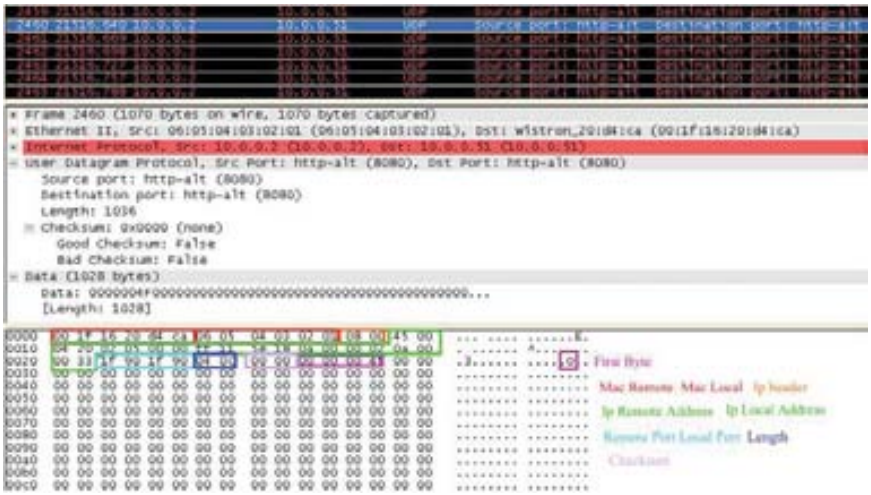


Figure 4.6: Transmission result in the wireshark.

4.3.1.3 TE/TE Interface Design - Massage Passing Interface (MPI)

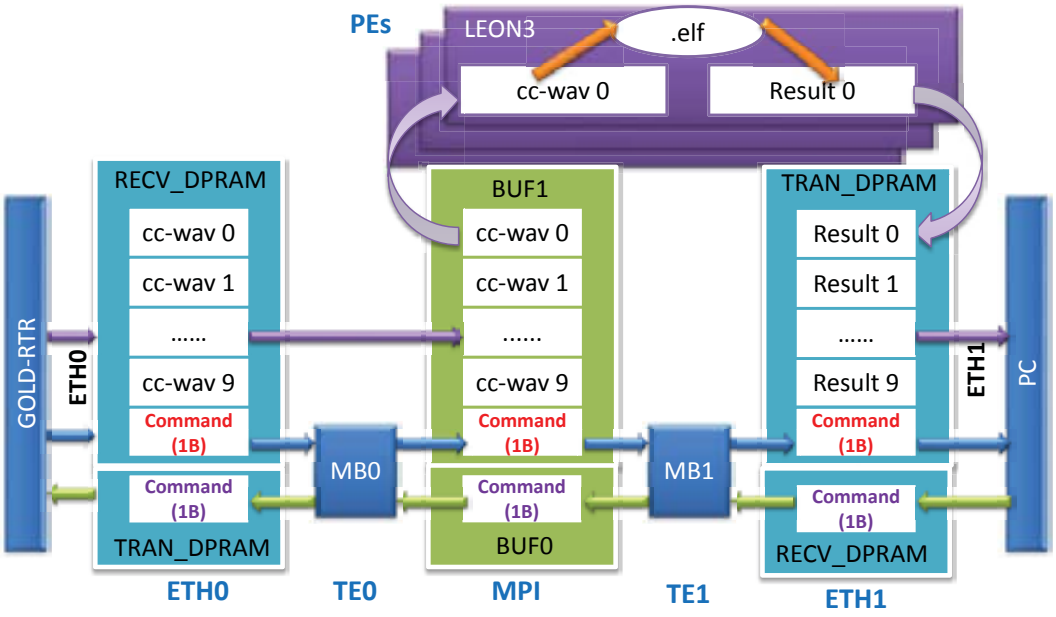


Figure 4.7: MPI transmission in HTPCP.

The interconnections between TE0 and TE1 are done via a MPI, which is capable of receiving and passing commands and waveforms to the remote processor transparently, and bear the conflicts of the local traffic. The MPI is consist by a Dual-Port BRAM (DPRAM) and two BRAM controllers, which are connected and controlled by two MicroBlaze Processors (MB0 and MB1) respectively. The DPRAM includes two buffers (BUF0 and BUF1), and each BRAM controller is capable to send and receive data or commands in these two buffer as shown in Fig.4.7. The key points is aware of the conflicts of the transmission by the sequence control of MPI. As shown in Fig.4.8, the four simple functions of two processors can control the transmission without conflictions.

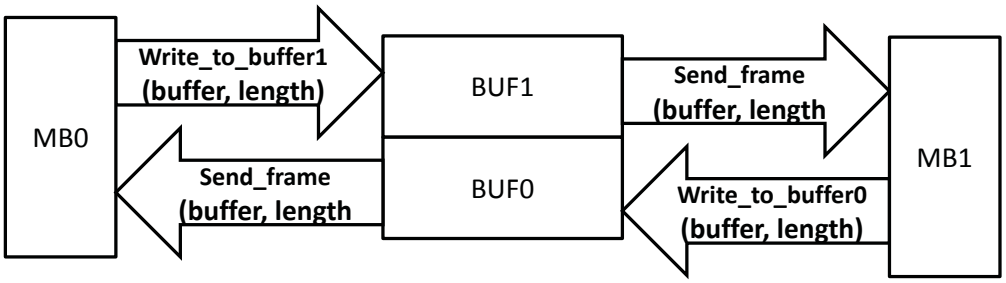


Figure 4.8: MPI sequence control.

4.3.2 Processing Elements

The parallel architectures have the potential to satisfy the timing requirement. However, in most cases, adding more cores does not increase the throughput since workloads cannot always take benefit from multiple cores. Thus we divide the workload into several PEs and TEs, and reallocate the memory resource for each element to exploit the maximize efficiency of parallel computing.

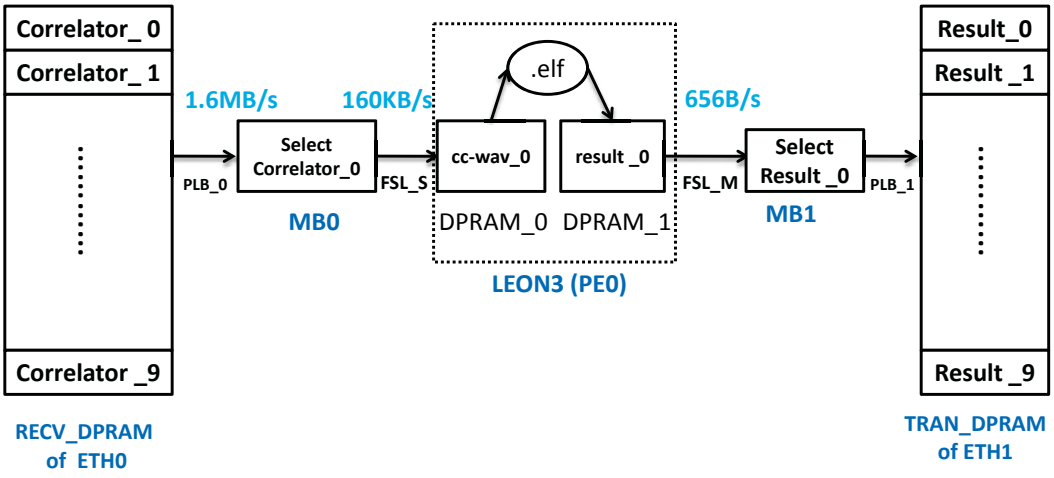


Figure 4.9: The data path in PE design.

The data path in PE is shown in Fig.4.9, where the ten channel waveforms transmit from GOLD-RTR to the RECV_DPRAM of ETH0. The first channel waveform is selected by MB0 and transferred to the LEON3 system by FSL_S link. There are two pointers of the *elf* executable file that point to the physical address of DPRAM_0 and DPRAM_1 respectively. The function of the elf executable file is trigged by the entry address pointer, and generate the result by the end address pointer. The input waveform format $ccwav\_0$ is shown in Appendix A, each waveform has 32 B header and 128 B input data. The output waveform format $result\_0$ is shown in Appendix B, each waveform has 656 B output data. The DPRAM_0 and DPRAM_1 work as the safe threshold to restrict the bandwidth of input and output data. The throughput of input waveform degrades from 1.6MB/s to 160KB/s, after selecting the channel by MB0. The throughput of the output waveform is only 656B/s for each PE. Therefore, the size of DPRAM_0 is set to 16KB to store the CC-WAV for each correlation channel during 1 ms. And the size of DPRAM_1 is set to 16KB to store the result for ten correlation channel during 1 s. To better understand the PE work principle, we focus on the two issues as follows:

- How to solve the memory allocation issue and minimize system latency by the memory hierarchy design.
- How to design the PE and TE interface by FSL & Gpio.

4.3.2.1 Memory Hierarchy Design

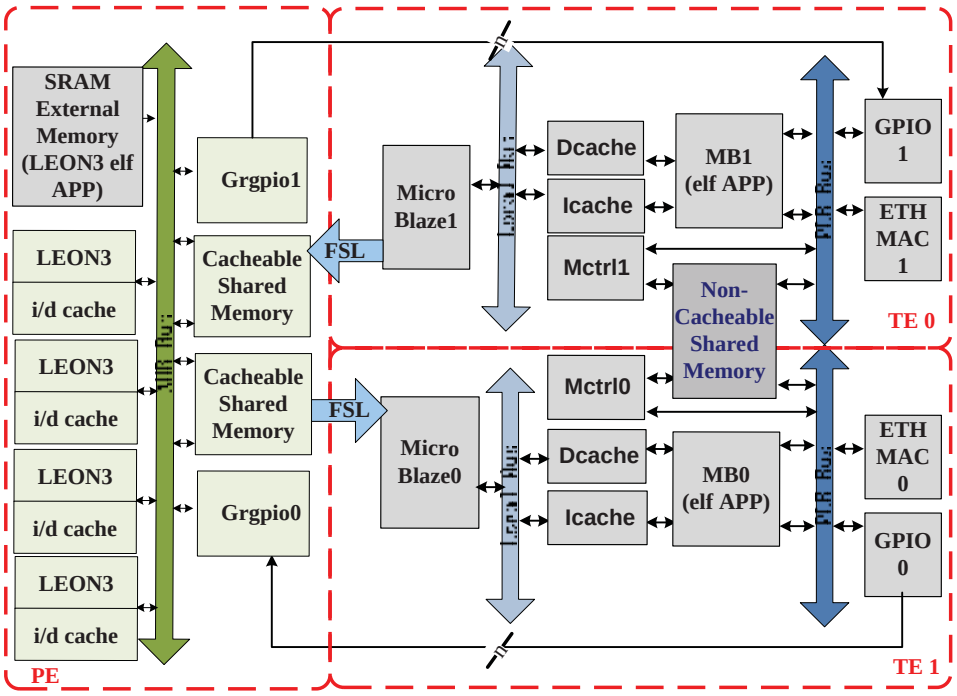


Figure 4.10: Memory hierarchy design in HTPCP.

The previous analysis has an interpretation in terms of the system bandwidth and the timing requirement of GNSS-R application. Since each waveform owns the universal timing parameter (i.e. WeekSow, millisecond) in the header, it is either to process the relevant waveform, or to dump the waveform. In order to process the waveform for each correlation channel, the memory allocation issue and system latency have to be studied. The memory hierarchy design is a design challenge to minimize system latencies and solve the memory allocation issue.

Fig.4.10 shows the memory hierarchy design in HTPCP. It can be observed that four types of memory design.

- I/dcache of LEON3 processor can achieve a fast computation as Level 1 memory design.
- Cacheable shared memory of LEON3 processor is composed by a Dual-Port RAM (DPRAM), the size of DPRAM is depend on the bandwidth of PEs. It employs as the level 2 memory design, the novelty lies in the multi-ports read and write by two types of processor. That means the MB0 processor can write the waveforms into DPRAM, meanwhile, the LEON3 processors can read out the waveforms.
- Another level 2 memory design that was implemented is the Non-cacheable shared memory. It is composed by one DPRAM and two BRAM controllers as it was previously mentioned in MPI design. It aims at non-latency communication

between TE0 and TE1. The size of the DPRAM is variant with the bandwidth of TEs.

- The Level 3 memory represents on-chip Block RAM (BRAM) or off-chip Static RAM (SRAM) of LEON3 processors, which keeps the elf executable file.

As for the GNSS-R application, it can be implemented in a software routine (the elf executable file of LEON3) or a hardware routine (CORR component). Based on the different approaches (software routine or hardware routine) of implementing the application, we propose three solutions for the memory hierarchy design of PE, with the consideration of various factors, i.e. elf allocations, input waveform allocation and output data allocation etc..

- Solution 1: Four LEON3 processors share one software routine.

  In the first solution, four LEON3 processors work together with their private i/dcache, but share with two DPRAMs and the same *ahb* bus as shown in Appendix.C. There is one software routine compiled to elf executable file and allocated into the SRAM0, in order to control the tasks of each LEON3 processors.

  Table 4.I summaries the memory allocation and size employed for this memory hierarchy design. Four LEON3 processors work together with their private Level 1 memory (i/dcache) with the capacity of 2 set* 4 KB/2 set *16 KB. The input data (CC-WAV) fits into the Level 2 memory (DPRAM_0) with the capacity of 1 KB. And the output data (result) fits into another Level 2 memory (DPRAM_1) with the capacity of 1 KB. The elf executable file fits into the L3 memory (SRAM0) with the capacity of 8MB.

  The advantage of this approach is to use the SRAM to fit for the software routine, in order to save the on-chip resources (RAMB16). The disadvantage is that the four LEON3 cores are sharing one elf file, which is executed in sequence. Therefore, it will reduce the processing efficiency by each processor. The processing of each processor is controlled by a multi-processor interruptor, and each processor can only access a section of the Level 2 memory. Meanwhile, we need two GPIO links to control the arrival time of CC-WAV and result in Level 2 memory, in order to calculate the precise processing time.

Table 4.I: Solution1: memory hierarchy design summary.

Memory hierarchy	Component	Size of Memory (set/KB)	Contents of Memory	No.LUT	No.RAMB16	NO.DPRAM
Level 1	dcache/icache	2*4/2*16	data and instruction	4370	36	920
Level 2	DPRAM_0/DPRAM_1	1/1	CC-WAV/results			
Level 3	SRAM	8000	elf executable file			

- Solution 2: Four LEON3 processors with four hardware routines.

  In the second solution, four LEON3 processors work together with their private i/dcache, DPRAM and bus as shown in Appendix.D. There are four hardware

routines (CORR) created by VHDL and connected with each DPRAM and FSL link, in order to fulfill the post-processing application as a co-processor.

Table 4.II summaries the memory allocation and size employed for this memory hierarchy design. Four LEON3 processors work together with their private Level 1 memory (i/dcache) with the capacity of 2 set* 4 KB/2 set *16 KB. The input data (CC-WAV) fits into each Level 2 memory (DPRAM_0) with the capacity of 1 KB, which is sent directly to CORR component and generate the result by CORR. The other end of CORR is connected with FSL link, which can transmit the result to the MB system.

The advantage of this approach is that the post-processing application will be processed in real-time without manual intervention. We can monitor the processing by each clock cycle with a hardware timer. We don't need the L3 memory to fit for the elf application, or another DPRAM to store the results. Therefore, it save a lot of on-chip resources (RAMB16), since it only use one DPRAM and no external memory. Also we don't need the GPIO links to control the processing time. The disadvantage of this approach is that we cannot replace the algorithm flexibly as the software routine.

Table 4.II: Solution2: memory hierarchy design summary.

Memory hierarchy	Component	Size of Memory (KB)	Contents of Memory	No.LUT	No.RAMB16	NO.DPRAM
Level 1	dcache/icache	2*4/2*16	data and instruction	5370	12	0
Level 2	DPRAM	1	CC-WAV			

- Solution 3: Four LEON3 processors execute four software routines in parallel.

  In the third solution, four LEON3 processors work together with their private i/dcache, two private DPRAMs, private bus and a private BRAM as shown in Appendix.E. There are four software routine compiled to elf executable file and allocated into each BRAM, in order to control each LEON3 processors separately.

  Table 4.III summaries the memory allocation and size employed for this memory hierarchy design. Four LEON3 processors work together with their private Level 1 memory (i/dcache) with the capacity of 2 set* 4 KB/2 set *16 KB. The input data (CC-WAV) fits into the Level 2 memory (DPRAM_0) with the capacity of 1 KB. And the output data (result) fits into another Level 2 memory (DPRAM_1) with the capacity of 1 KB. The elf executable file fits into its own on-chip L3 memory (BRAM0) with the capacity of 128KB.

  The advantage of this approach is that it can realize the parallel processing with four different elf executable file on each processor. The disadvantage of this approach is that the elf applications cannot be larger than 128 KB, since it spends a lot of on-chip resources (RAMB16, LUT, Dual-port RAM etc.). In the actual design, the smallest size of elf application can be compiled as 155 KB, which is larger than the 128 KB. Also we need two GPIO links to control the arrival time of CC-WAV and result to the level 2 memory, in order to grantee the precise processing time.

Table 4.III: Solution3: memory hierarchy design summary.

Memory hierarchy	Component	Size of Memory (KB)	Contents of Memory	No.LUT	No.RAMB16	NO.DPRAM
Level 1	d/iCACHE	2*4/2*16	data and instruction	2370	56	0
Level 2	DPRAM_0/DPRAM_1	1/1	CC-WAV/result			
Level 3	Bram_0	128	elf executable file			

Through the analysis of these three solutions, solution one seems to be the most practical approach, and solution two and three can be used to improve the system performance in future. Table.4.IV depicts the final memory hierarchy design of PE (LEON3 system) and TE (MB system). This approach can get rid of the confliction of transmission and processing by dividing the whole system into TEs and PEs. Also it can accurately measure the processing time by each LEON3, and measure the transmission time by the MBs. There are two different interfaces that connect with MB system and LEON3 system: FSL and gpio, which will be described in the next Subsection.

Table 4.IV: The optimized memory hierarchy design summery.

LEON3 System	LEON3 Frequency	100 MHz
	Input Throughput	160 KB/s
	L1 Memory: Icache	2 * 8 KB; LRU
	L1 Memory: Dcache	2 * 8 KB; LRU
	L2 Memory: DPRAM0	1 KB
	L2 Memory: DPRAM1	1 KB
	L3 Memory: SRAM0	8 MB
Microblaze System	Microblaze Frequency	100 MHz
	L1 Memory: Icache	128 KB
	L1 Memory: Dcache	128 KB
	L2 Memory: BRAM	128 KB
	Input Throughput	1.6 MB/ s
	Output Throughput	656 B / s
	Input/Output Ratio	2545 : 1
	Overall execution time (for 1 second input sample)	In the level of us (10^{-6})

4.3.2.2 PE/TE Interface Design - FSL & GPIO

The interface design between PEs and TEs are fulfilled by two FSL links and two gpios links. The novelty of the HTPCP architecture is the introduction of the control mechanisms for integrating the two processor systems. The control mechanisms is able to provide the stream data transmission guarantees the synchronization of the transmission and processing.

The control mechanisms can be designed as a Finite State Machine (FSM), as presented in Fig.4.11. The FSL interface includes two FSL links: FSL_S slave link and FSL_M master link. Each link connects with one DPRAM ($ahbdpram\_0$, $ahbdpram\_1$) separately. The FSM controls the read and write sequences of each FSL link. The functions of three states: $idle$, $read\_input$ and $write\_output$ can be viewed as follows:

- The $idle$ state, the FSM will reset the two DPRAMs and two gpios.

- The $read\_input$ state, the FSM will read the stream data (FSL_S_data) from MB to $ahbdpram\_0$ by the FSL_S link. While the $ahbdpram\_0$ receives the first input data, the elf application will send a signal to GPIO_0 ($gpioo$) and write the calculated results to the $ahbdpram\_1$.

- The $write\_output$ state, the FSM will write the result (FSL_M_data) from $ahbdpram_1$ to the FSL_M master link. While the MB receives the results, it will send a signal to GPIO_1 ($gpioi$) that indicates the FSM to reset the system to the $idle$ state.

The FSM can control the read and write operations of the FSL links without the delay. The execution time of the elf application can be monitored by two gpios as the data arriving of PEs and TEs.

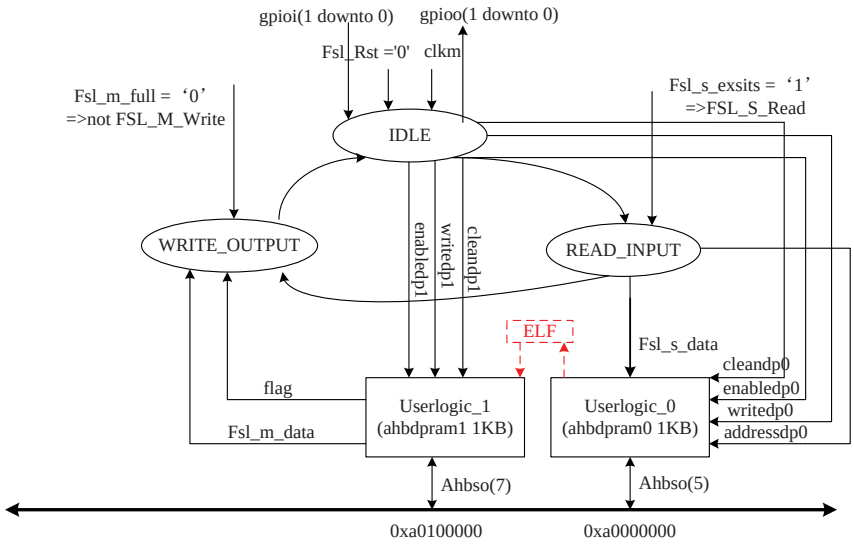


Figure 4.11: The Finite State Machine (FSM) design.

The integration of a customized IP core within the execution unit is very restrictive due to the nature of RISC processor architectures. The restriction is the customized instruction itself. If the critical path of the whole system is through the user IP, the whole soft processor will decrease in performance (processor frequency), since the user IP is included within the soft processor architecture itself. If the RISC architecture doesn't allow the designer to stall the pipeline, the processor can't run at a higher frequency than the critical path would allow. The bigger the customized IP is, the more the designer must be careful not to decrease the whole processor performance.

Fig.4.12 shows the basic idea of connecting the customized user IP with the MB system via the FSL interface. In our application, this IP core refers to a LEON3-based system. Xilinx provides the MB processor up to 16 dedicated 32-bit FSL interfaces, which is a very powerful, easy and flexible way to integrate a customized user IP into a MB-based system. The FSL channels are uni-directional, point-to-point data streaming interfaces. It is possible to provide the customized IP core with many more

inputs/outputs from another processor or external logic, and the big advantage is not necessary to change or extend the MB core or the RISC architecture itself.

During the synthesis of the LEON3 systme design by Synplify Pro v8.6.2, we found out that the critical path does not go through the DPRAM and FSM. It means that these two components will not affect the system processing time or the system frequency. From this point onward, if the PE takes 100 clock cycles to calculate the result by the elf executable file, the FSM will read and write data from DPRAMs continuously by the FSL_S link and the FSL_M link without interruptions. Meanwhile the TEs can execute their own applications, and don't have to wait for the FSL to be available. Therefore, the pipeline of RISC processor cannot be stalled and the processor frequency of TEs and PEs will not be decreased in the HW design.

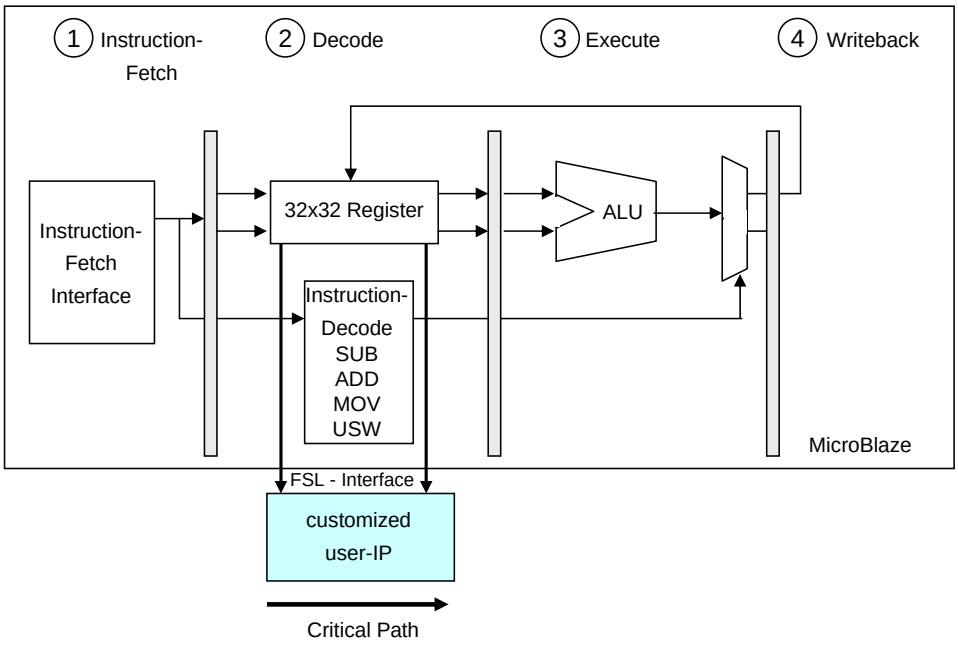


Figure 4.12: Integrated a customized IP into MicroBlaze via the FSL interface.

In the PE and TE interface design, it is easy to verify the data transmission in two DPRAMs by hardware design. However, the biggest challenge is to measure the overall execution time of the elf executable file, since it is a software routine and the value is not constant. Fig.4.13 represents the sequence chart of HTPCP, where the time between two red lines displays the overall execution time of the elf executable file, in addition to this part is controlled by FSM and gpios. One approach to providing the precise the execution time of software routine is to add a timer function in software routine, and ”printf” the time consumption by UART. However, the ”printf” function costs a large percentage of the total execution time. At present, we can only roughly estimate the overall execution time of elf executable file is in the level of “us“ (10^{-6}) apply to one sample waveform (1mec). Appendix I shows the captured the time consumption by UART. In the future, we can connect a hardware timer to the register file of LEON3, thus get more accurate measurement of the elf execution time.

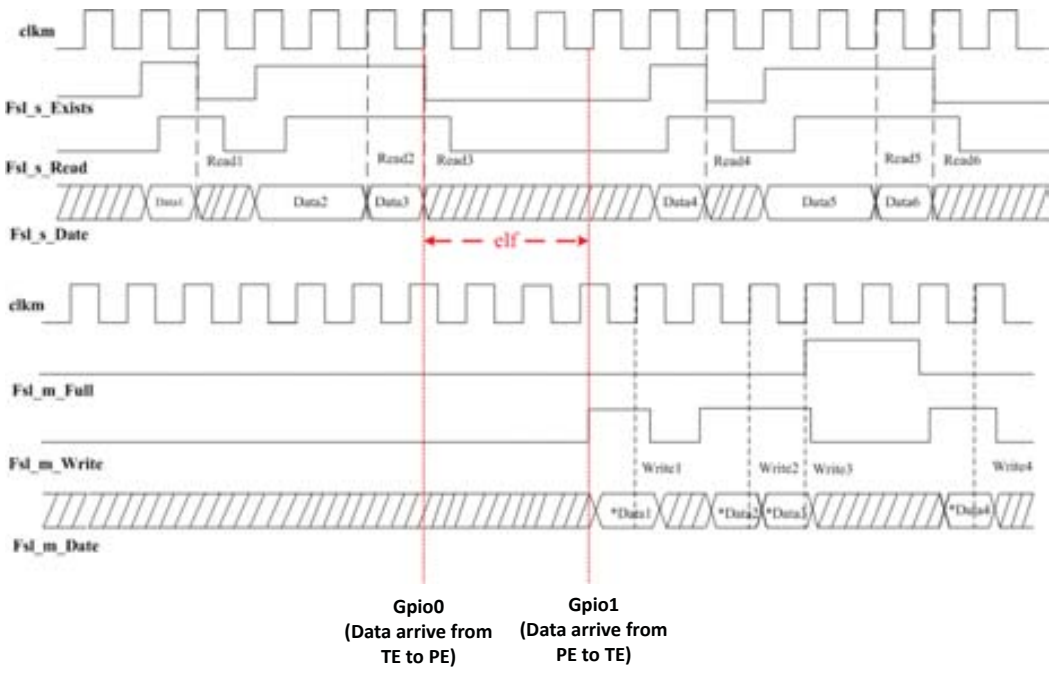


Figure 4.13: The sequence chart of HTPCP.

4.3.3 Design Flow of HTPCP

The main focus of this study is the integration of two processor systems. The focus is on the design method of LEON3 system and Microblaze system, and a methodology to integrate the two system. The design has been created by Xilinx Embedded Development Kit (EDK) tools, and layout for the GR-CPCI-XC4V board.

The block diagram of HTPCP is presented in Fig.4.14. It includes two MB processors with their own buses (mb_plb_0 and mb_plb_1). Several peripherals are connected with the each MB bus (xps_gpio, xps_ethernetlite, xps_timer etc.). The LEON3 system contains four LEON3 processors, which is connected with the same AHB bus, Several peripherals are connected with the AHB bus (i.e. GR-GPIO, DPRAM, MP IRQ Controller, etc.). Note that each MB has one FSL link and one GPIO link connecting with LEON3 system, this will be the key parts to successfully integrate two processor systems.

There are three steps to finalize the configuration of the HTPCP system:

- Configure the LEON3 system, generate the netlist (.edf) of LEON3MP.
- Create the MB system by editing the MHS and MSS files.
- Integrate the LEON3 system as an IP core into the MB system.

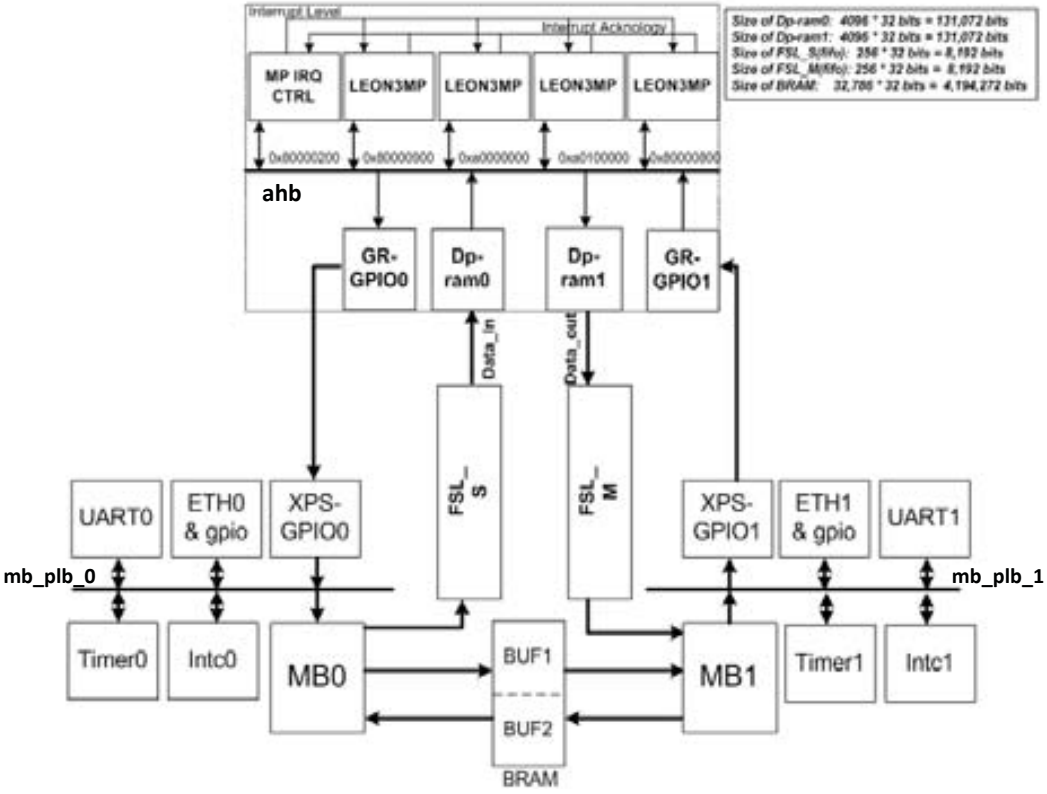


Figure 4.14: HTPCP design flow.

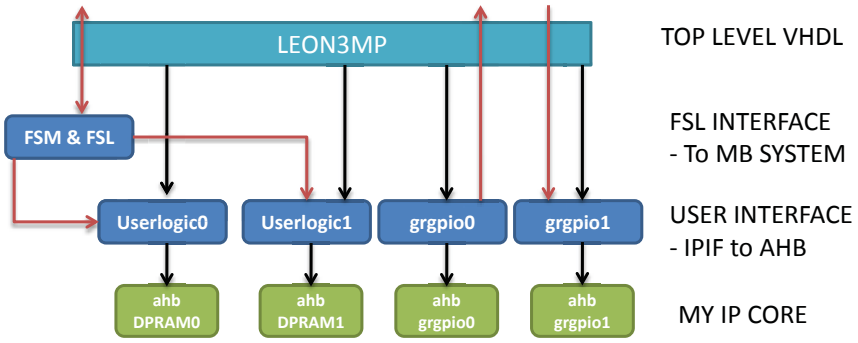


Figure 4.15: IP cores design in LEON3 system.

4.3.3.1 LEON3 System

The IP cores design in LEON3 system is shown in Fig.4.15. It is observed that the LEON3MP block is highlighted as the top-level design in the system. The design focus on the configuration of the top-level VHDL file (leon3mp.vhd, config.vhd). The *leon3mp.vhd* includes the instantiations of IP cores and available I/O ports in LEON3 system. In particular, it is focus on adding the IP cores into the top-level module. In the second level design, we need to add the FSL links into the LEON3MP top-level module, and write a FSM to control the read/write operations of the two FSL

links. This design is to prepare the integration between LEON3 system and Microblaze system. In the following level design, four IP cores and their interfaces have been added into the LEON3MP. Note that the userlogic_1 and userlogic_2 designs are connecting with the two FSL links. And the grgpio0 and grgpio1 designs has to external the input and output pins to connect with MB system.

The methodology of adding the IP cores into the LEON3MP top-level module is described in further detail in the following points:

1. The VHDL code presents in Fig.4.16 shows the two FSL interfaces (master link and slave link) and their I/O ports adding into the *leon3mp.vhd* file. There are two pairs of FSL Slave/Master links has been added into the top-level module, but only one link of each pair (FSL0_S and FSL1_M) has been used to connect with MB0 and MB1 separately. The reason is to avoid the interference between two Microblaze processors, each MB are connecting to a pair of FSL Slave/Master links independently. As it was mentioned before in Section 4.3.2.2, the FSM has been added in the LEON3MP top-level module, where *fsl_S_data_pad* and *fsl_M_data_pad* have been created in the top-level module, that is aim at connecting the two FSL links with the inputs of IP cores (userlogic_1 and userlogic_2) as shown in Fig.4.17.

```
FSL_Rst : in    std_logic;
 -- Slave FSL Signals
FSL0_S_Clk     : out std_logic;
FSL0_S_Read    : out std_logic;
FSL0_S_Data    : in  std_logic_vector(0 to C_DWIDTH-1);
FSL0_S_Control : in  std_logic;
FSL0_S_Exists  : in  std_logic;

FSL1_S_Clk     : out std_logic;
FSL1_S_Read    : out std_logic;
FSL1_S_Data    : in  std_logic_vector(0 to C_DWIDTH-1);
FSL1_S_Control : in  std_logic;
FSL1_S_Exists  : in  std_logic;

-- Master FSL Signals
FSL0_M_Clk     : out std_logic;
FSL0_M_Write   : out std_logic;
FSL0_M_Data    : out std_logic_vector(0 to C_DWIDTH-1);
FSL0_M_Control : out std_logic;
FSL0_M_Full    : in  std_logic;

FSL1_M_Clk     : out std_logic;
FSL1_M_Write   : out std_logic;
FSL1_M_Data    : out std_logic_vector(0 to C_DWIDTH-1);
FSL1_M_Control : out std_logic;
FSL1_M_Full    : in  std_logic;

FSL1_M_Clk_debug     : out std_logic;
FSL1_M_Data_debug    : out std_logic_vector(0 to C_DWIDTH-1);
FSL1_M_Write_debug   : out std_logic
```

Only use this as FSL_Slave Link

Only use this as FSL_Master Link

Figure 4.16: Two FSL links in leon3mp.vhd.

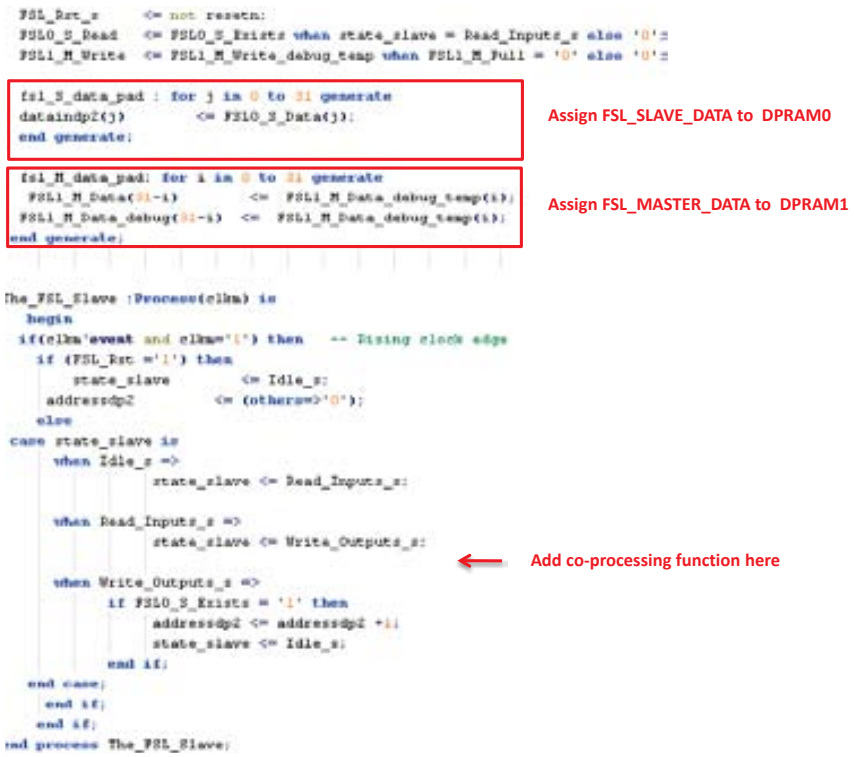


Figure 4.17: FSM design with two FSL links.

2. Fig.4.18 presents the VHDL code of the ahbdpram0 and ahbdpram. These are two different DPRAM, one (ahbdpram) is to connect with FSL0_S link, in order to store the CC-WAV of each correlation channel. The other (ahbdpram0) is to connect with FSL1_M link, in order to store the calculated results of each correlation channels. Their interface designs (IPIF to AHB) in *leon3mp.vhd* file are shown in Fig.4.19, and their I/Os are connecting to the corresponding I/Os pins of the FSL links.

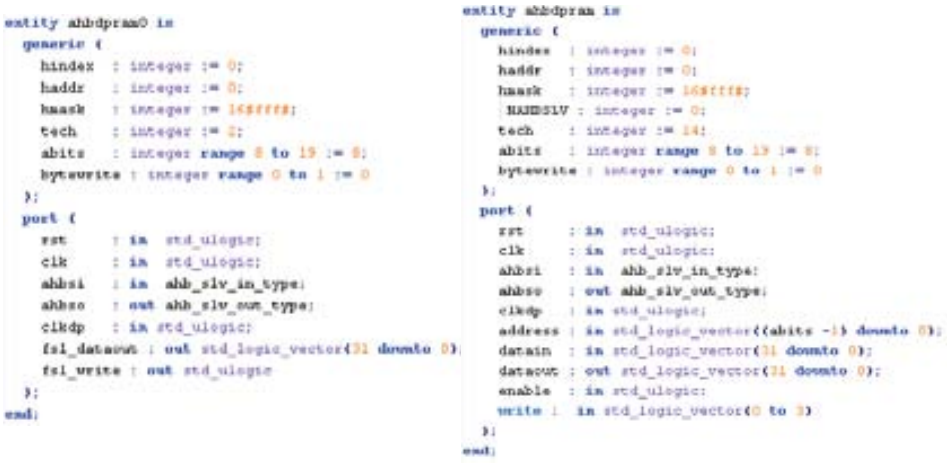


Figure 4.18: IP cores design of Ahbdpram0 and Ahbdpram.

```
userlogic_1: entity work.ahbdpram0
      generic map (hindex=>3, haddr=>16#a01#, hmask=>16#fff#,tech=>virtex4,abits=>abitsdp, bytewrite=>0)
    port map(rstn,clkm,ahbsi,ahbso(3),clkm, FSL1_M_Data_debug_temp, FSL1_M_Write_debug_temp);
   FSL1_M_Write_debug <= FSL1_M_Write_debug_temp;

userlogic_2: entity work.ahbdpram
      generic map (hindex=>5, haddr=>16#a00#, hmask=>16#fff#, NAHBSLV=>4,tech=>virtex4,abits=>abitsdp, bytewrite=>0)
    port map(rstn,clkm,ahbsi,ahbso(5),clkm,addressdp2,dataindp2,dataoutdp2,enabledp2,writedp2);
    enabledp2<='1';  writedp2 <= "1000";
```

Figure 4.19: Ahb ipif designs of Ahbdpram0 and Ahbdpram.

3. Fig.4.20 presents the interface designs (IPIF to AHB) of grgpio0 and grgpio1 in *leon3mp.vhd* file. The external input pin (*gpio1*) are connecting with the input pin (*grgpio1.din*) of grgpio1 component; the external output pin (*gpio0*) are connecting with the output pin (*grgpio0.dout*) of grgpio0 component. We can observe that the external input pin (*gpio1*) and output pin (*gpio0*) pins of the LEON3MP top module are prepared to connect with the xps_gpio0 in MB0 system and xps_gpio1 in MB1 system. As it was mentioned before in Section 4.3.2.2, these two GPIO links is used to guarantee the data arrive between LEON3 system and Microblaze system.

```
gpio_leon3 : if CFG_GRGPIO_ENABLE /= 0 generate
  ----------------------------------------------------------------
  grgpio0: grgpio
    generic map( pindex => 9, paddr => 9, imask => CFG_GRGPIO_IMASK,
  nbits => 32)
    port map( rstn, clkm, apbi, apbo(9), gpioi0, gpioo0);

    pio_0_pads : for i in 0 to 31 generate
    gpioo0.oen(i)   <= '1';
    gpio0(i) <= gpioo0.dout(31-i);
    end generate;
  ----------------------------------------------------------------
  grgpio1: grgpio
    generic map( pindex => 8, paddr => 8, imask => CFG_GRGPIO_IMASK,
  nbits => 32)
    port map( rstn, clkm, apbi, apbo(8), gpioi1, gpioo1);


    pio_1_pads : for j in 0 to 31 generate
        gpioi1.din(j)  <=   gpio1(31-j);
        gpioi1.SIG_EN(j) <= '1';
        gpioo1.oen(j)   <= '1';
     end generate;

end generate;
```

Figure 4.20: IP cores design of grgpio0 and grgpio1.

4. Finally, a compilation of the LEON3 system has been done by Synplify Pro as shown in Fig.4.21, where includes the the top-level files (leon3mp.vhd, config.vhd) and the IP cores (userlogic.vhd, grgpio.vhd etc.). The netlist (*leon3.edf*) file has been generated prepared for the final integration with MB system.

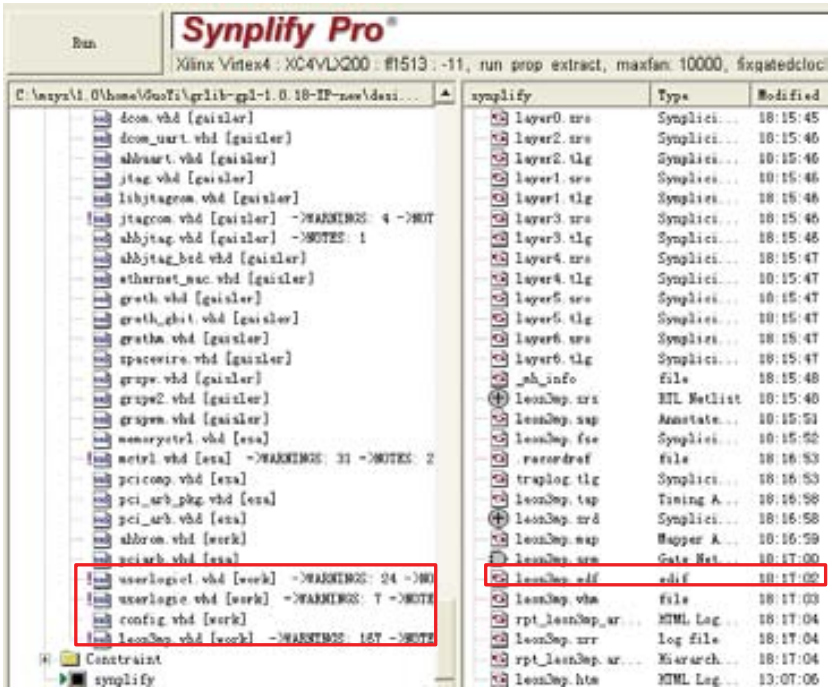


Figure 4.21: Compile LEON3 system by Synplify Pro.

4.3.3.2 Microblaze System

Microblaze has the ability to use its dedicated FSL bus to integrate a customized IP core. However, the integration of two processor systems by FSL is an unprecedented experiment. The focus of the MB systems design is addressed in this Section, where we can understand how to create MB systems of HTPCP by editing the MHS and MSS.

In the Xilinx EDK tools, all the IP cores will be integrated in the Xilinx Microprocessor Project (XMP) with the project file (system.xmp). In our design, we use two Microblaze processors (*microblaze_0* and *microblaze_1*) work as a dual-core platform based on the GR-CPCI-XC4V development board. The"*Target device*" is xc4vlx200ff1513-11 FPGA. The main design steps can refer to the EDK Concepts, Tools, and Techniques Manual [*Xilinx EDK*]. Here we mainly focus on the design considerations of each component in MB system of HTPCP.

Our fancy processors (*microblaze_0* and *microblaze_1*) are not going to be much used unless we connect them to the following IP cores:

- **microblaze v7.10d**- MicroBlaze architecture includes big-endian bit-reversed format, 32-bit general purpose registers, virtual-memory management, cache software support, and FSL interfaces. In our design, the two MicroBlazes (*microblaze_0* and *microblaze_1*) operate as the function of TE0 and TE1.
- **fsl_v20 v2.11a**- The LogiCORE IP FSL V20 bus is a uni-directional point-to-point communication channel bus, used to perform fast communication between

```
BEGIN microblaze
 PARAMETER INSTANCE = microblaze_0
 PARAMETER HW_VER = 7.10.d
 PARAMETER C_FSL_LINKS = 1
 PARAMETER C_FSL_DATA_SIZE = 32
 PARAMETER C_INSTANCE = microblaze_0
 PARAMETER C_FAMILY = virtex4
 BUS_INTERFACE MFSL0 = i_fsl_v20_0
 BUS_INTERFACE DPLB = mb_plb_0
 BUS_INTERFACE IPLB = mb_plb_0
 BUS_INTERFACE DLMB = d_lmb_0
 BUS_INTERFACE ILMB = i_lmb_0
 PORT MB_RESET = mb_reset
END

BEGIN microblaze
 PARAMETER INSTANCE = microblaze_1
 PARAMETER HW_VER = 7.10.d
 PARAMETER C_FSL_LINKS = 2
 PARAMETER C_FSL_DATA_SIZE = 32
 PARAMETER C_INSTANCE = microblaze_1
 PARAMETER C_FAMILY = virtex4
 BUS_INTERFACE SFSL1 = i_fsl_v20_1
 BUS_INTERFACE DPLB = mb_plb_1
 BUS_INTERFACE IPLB = mb_plb_1
 BUS_INTERFACE DLMB = d_lmb_1
 BUS_INTERFACE ILMB = i_lmb_1
 PORT MB_RESET = mb_reset
```

two processor systems (LEON3 system and MBs system). The FSL interface is available on each MB processor, but add manually in the LEON3 system. The interfaces are used to transfer stream data to and from the register file on MB processor to LEON3 system. In our design, the FSL Slave link (i_fsl_v20_0) is connected to the MB0, and the FSL Master link (i_fsl_v20_1) is connected to the MB1.

- **lmb_v10 v1.00a** - The LMB is a fast, local bus for connecting MB instruction and data ports to high-speed peripherals, primarily on-chip block RAM (BRAM). Here we combine two lmb_v10 modules to each MB to play the roles of Dcache and Icache. Therefore, four lmb_v10 modules are created to combine with two MBs.

- **lmb_bram_if_cntrl v2.10a** - The LMB BRAM interface controller connects to an lmb_v10 bus. Version 2.10a of the LMB BRAM controller is required for use with MicroBlaze v6.00a and later, to correctly handle the address mask computation. In our design, four LMB BRAM controllers are created to control each lmb_v10 module.

- **plb_v46 v1.03a** - The Processor Local Bus (PLB) is part of the IBM CoreConnect bus architecture specification, it is the high-speed data interface to the MB core.

```
BEGIN fsl_v20
 PARAMETER INSTANCE = i_fsl_v20_1
 PARAMETER HW_VER = 2.11.a
 PARAMETER C_EXT_RESET_HIGH = 1
 PARAMETER C_FSL_DWIDTH = 32
 PARAMETER C_FSL_DEPTH = 256
 PORT FSL_Clk = sys_clk_s
 PORT SYS_Rst = net_gnd
END

BEGIN fsl_v20
 PARAMETER INSTANCE = i_fsl_v20_0
 PARAMETER HW_VER = 2.11.a
 PARAMETER C_EXT_RESET_HIGH = 1
 PARAMETER C_FSL_DWIDTH = 32
 PARAMETER C_FSL_DEPTH = 256
 PORT FSL_Clk = sys_clk_s
 PORT SYS_Rst = net_gnd
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_bram_if_cntlr_1
 PARAMETER HW_VER = 2.10.a
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0001ffff
 BUS_INTERFACE SLMB = i_lmb_0
 BUS_INTERFACE BRAM_PORT = conn_1
END

BEGIN lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_bram_if_cntlr_0
 PARAMETER HW_VER = 2.10.a
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x0001ffff
 BUS_INTERFACE SLMB = d_lmb_0
 BUS_INTERFACE BRAM_PORT = conn_0
END

BEGIN lmb_v10
 PARAMETER INSTANCE = i_lmb_0
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk_s
 PORT SYS_Rst = sys_bus_reset
END

BEGIN lmb_v10
 PARAMETER INSTANCE = d_lmb_0
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk_s
 PORT SYS_Rst = sys_bus_reset
END
```

All of the peripherals and system memory will communicate with the processor by this bus. In order to reduce the transmission load, each MB posses its own bus and peripherals. *mb_plb_0* is the bus of MB0, and *mb_plb_1* is the bus of MB1.

- **bram_block v.1.00a** - There are three BRAMs in our design. Two BRAMs (bram1 and bram2) are created, with the size of 128KB each. The function of these two BRAM are to store the elf executable file of each MB. Since *bram1* connects with *lmb_bram_if_cntlr_0* and *bram2* connects with *lmb_bram_if_cntlr_1*, they will not require additional xps_bram controllers to control the memory. Moreover, another BRAM (bram_block_0) is created, with the size of 128 KB. The function

```
BEGIN plb_v46
 PARAMETER INSTANCE = mb_plb_0
 PARAMETER HW_VER = 1.03.a
 PORT PLB_Clk = sys_clk_s
 PORT SYS_Rst = sys_bus_reset
END

BEGIN plb_v46
 PARAMETER INSTANCE = mb_plb_1
 PARAMETER HW_VER = 1.03.a
 PORT PLB_Clk = sys_clk_s
 PORT SYS_Rst = sys_bus_reset
END
```

of this BRAM is to fulfil the MPI interconnection between MB0 and MB1.

- **xps_bram_if_cntrl v1.00a**- This module will be the interface to control the BRAM (*bram_block_0*) for MPI design. As it was mentioned before, the BRAM works as the shared memory between MB0 system and MB1 system. But the two xps_bram controllers are connected to each port of BRAM with *PLB_0* and *PLB_1* as shown in Fig.4.22. In our design, two xps_bram controller modules are created to realize the MPI design.

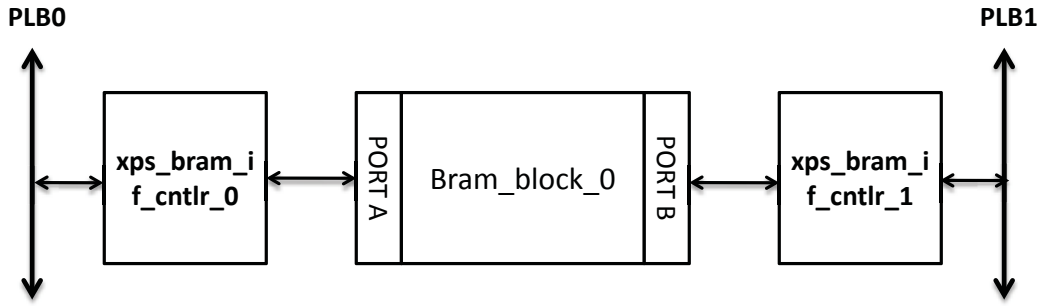


Figure 4.22: MPI design with one bram_block and two bram controllers.

```
BEGIN bram_block
 PARAMETER INSTANCE = bram_block_0
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = xps_bram_if_cntlr_0_PORTA
 BUS_INTERFACE PORTB = xps_bram_if_cntlr_1_PORTB
END

BEGIN xps_bram_if_cntlr
 PARAMETER INSTANCE = xps_bram_if_cntlr_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_SPLB_NATIVE_DWIDTH = 32
 PARAMETER C_BASEADDR = 0x83c20000
 PARAMETER C_HIGHADDR = 0x83c3ffff
 BUS_INTERFACE SPLB = mb_plb_0
 BUS_INTERFACE PORTA = xps_bram_if_cntlr_0_PORTA
END

BEGIN xps_bram_if_cntlr
 PARAMETER INSTANCE = xps_bram_if_cntlr_1
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_SPLB_NATIVE_DWIDTH = 32
 PARAMETER C_BASEADDR = 0x83c20000
 PARAMETER C_HIGHADDR = 0x83c3ffff
 BUS_INTERFACE SPLB = mb_plb_1
 BUS_INTERFACE PORTA = xps_bram_if_cntlr_1_PORTB
END
```

- **xps_ethernetlite v2.00a** - The Ethernet Lite MAC supports the IEEE Std. 802.3 Media Independent Interface (MII) to industry standard Physical Layer devices. It communicates with a processor via a PLB interface with the speed of 10 Mbps and 100 Mbps. Taking into account the connection between the HTPCP, GOLD-RTR and Contorl PC. At least, two Ethernet Lite MACs (Ethernet_MAC_0 and Ethernet_MAC_1) are necessary to connect with MB0 and MB1 separately. Each Ethernet MAC is controlled by one bus, and communicate transparently via MPI.

```
BEGIN xps_ethernetlite
 PARAMETER INSTANCE = Ethernet_MAC_0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_SPLB_CLK_PERIOD_PS = 16000
 PARAMETER C_RX_PING_PONG = 1
 PARAMETER C_TX_PING_PONG = 1
 PARAMETER C_BASEADDR = 0x81000000
 PARAMETER C_HIGHADDR = 0x8100ffff
 BUS_INTERFACE SPLB = mb_plb_0
 PORT PHY_rst_n = fpga_0_Ethernet_MAC_PHY_rst_n
 PORT PHY_crs = fpga_0_Ethernet_MAC_PHY_crs
 PORT PHY_col = fpga_0_Ethernet_MAC_PHY_col
 PORT PHY_tx_data = fpga_0_Ethernet_MAC_PHY_tx_data
 PORT PHY_tx_en = fpga_0_Ethernet_MAC_PHY_tx_en
 PORT PHY_tx_clk = fpga_0_Ethernet_MAC_PHY_tx_clk
 PORT PHY_rx_er = fpga_0_Ethernet_MAC_PHY_rx_er
 PORT PHY_rx_clk = fpga_0_Ethernet_MAC_PHY_rx_clk
 PORT PHY_dv = fpga_0_Ethernet_MAC_PHY_dv
 PORT PHY_rx_data = fpga_0_Ethernet_MAC_PHY_rx_data
 PORT IP2INTC_Irpt = Ethernet_MAC_0_IP2INTC_Irpt
END

BEGIN xps_ethernetlite
 PARAMETER INSTANCE = Ethernet_MAC_1
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_SPLB_CLK_PERIOD_PS = 16000
 PARAMETER C_RX_PING_PONG = 1
 PARAMETER C_TX_PING_PONG = 1
 PARAMETER C_BASEADDR = 0x81000000
 PARAMETER C_HIGHADDR = 0x8100ffff
 BUS_INTERFACE SPLB = mb_plb_1
 PORT PHY_rst_n = fpga_1_Ethernet_MAC_PHY_rst_n
 PORT PHY_crs = fpga_1_Ethernet_MAC_PHY_crs
 PORT PHY_col = fpga_1_Ethernet_MAC_PHY_col
 PORT PHY_tx_data = fpga_1_Ethernet_MAC_PHY_tx_data
 PORT PHY_tx_en = fpga_1_Ethernet_MAC_PHY_tx_en
 PORT PHY_tx_clk = fpga_1_Ethernet_MAC_PHY_tx_clk
 PORT PHY_rx_er = fpga_1_Ethernet_MAC_PHY_rx_er
 PORT PHY_rx_clk = fpga_1_Ethernet_MAC_PHY_rx_clk
 PORT PHY_dv = fpga_1_Ethernet_MAC_PHY_dv
 PORT PHY_rx_data = fpga_1_Ethernet_MAC_PHY_rx_data
 PORT IP2INTC_Irpt = Ethernet_MAC_1_IP2INTC_Irpt
END
```

- **xps_gpio v1.00a & v1.00b** - The XPS GPIO is a 32-bit peripheral that attaches to the PLB. Two xps_gpios (v1.00a) are created as MII_MDC_MDIO_GPIO_0 and MII_MDC_MDIO_GPIO_1, which connect to the PHY serial management bus of each xps_ethernetlite as shown in Fig.4.23. Moreover, two modified xps_gpios (v1.00b) are created as xps_gpio_0 and xps_gpio_0, which connect to the two I/O pins (*gpio0* and *gpio1*) of LEON3 system respectively. Note that xps_gpio_0 is modified as input direction of MB0 system, and the xps_gpio_1 is modified as output direction of MB1 system, corresponding to the PORT *gpio0* and *gpio1* in leon3mp_v2_1_0.mpd. There are four xps_gpios created in our design.
- **xps_intc v1.00a** - The XPS INTerrupt Controller (XPS INTC) concentrates multiple interrupt inputs from peripheral devices to a single interrupt output to the system processor. The registers for checking, enabling and acknowledging interrupts are accessed through a slave interface for the PLB bus. The number of interrupts can be tailored to the target system. In our design, two XPS_INTC (xps_intc_0 and xps_intc_1) modules are created for each MB.
- **xps_timer v1.00a** - The XPS timer/counter is a 32-bit timer module that

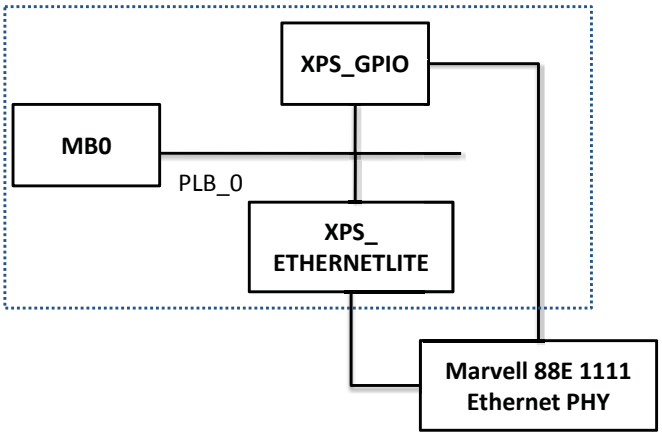


Figure 4.23: Xps_ethernetlite and xps_gpio block diagram.

```
BEGIN xps_gpio
 PARAMETER INSTANCE = xps_gpio_0
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_GPIO_WIDTH = 32
 PARAMETER C_IS_DUAL = 0
 PARAMETER C_IS_BIDIR = 0
 PARAMETER C_ALL_INPUTS = 1
 PARAMETER C_BASEADDR = 0x81420000
 PARAMETER C_HIGHADDR = 0x8142ffff
 BUS_INTERFACE SPLB = mb_plb_0
END

BEGIN xps_gpio
 PARAMETER INSTANCE = xps_gpio_1
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_GPIO_WIDTH = 32
 PARAMETER C_IS_DUAL = 0
 PARAMETER C_IS_BIDIR = 0
 PARAMETER C_ALL_INPUTS = 0
 PARAMETER C_BASEADDR = 0x81400000
 PARAMETER C_HIGHADDR = 0x8140ffff
 BUS_INTERFACE SPLB = mb_plb_1
 PORT GPIO_IO_O = xps_gpio1
END
```

```
BEGIN xps_intc
 PARAMETER INSTANCE = xps_intc_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x81800000
 PARAMETER C_HIGHADDR = 0x8180ffff
 BUS_INTERFACE SPLB = mb_plb_0
 PORT Irq = Interrupt
 PORT Intr = xps_timer_0_Interrupt& Ethernet_MAC_0_IP2INTC_Irpt
END

BEGIN xps_intc
 PARAMETER INSTANCE = xps_intc_1
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x81800000
 PARAMETER C_HIGHADDR = 0x8180ffff
 BUS_INTERFACE SPLB = mb_plb_1
 PORT Irq = Interrupt
 PORT Intr = xps_timer_1_Interrupt & Ethernet_MAC_1_IP2INTC_Irpt
END
```

attaches to the PLB bus. In our design, two xps_timer modules (xps_timer_0

and xps_timer_1) are created for each MB.

```
BEGIN xps_timer
 PARAMETER INSTANCE = xps_timer_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_COUNT_WIDTH = 32
 PARAMETER C_ONE_TIMER_ONLY = 0
 PARAMETER C_BASEADDR = 0x83c00000
 PARAMETER C_HIGHADDR = 0x83c0ffff
 BUS_INTERFACE SPLB = mb_plb_0
 PORT Interrupt = xps_timer_0_Interrupt
END

BEGIN xps_timer
 PARAMETER INSTANCE = xps_timer_1
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_COUNT_WIDTH = 32
 PARAMETER C_ONE_TIMER_ONLY = 0
 PARAMETER C_BASEADDR = 0x83c00000
 PARAMETER C_HIGHADDR = 0x83c0ffff
 BUS_INTERFACE SPLB = mb_plb_1
 PORT Interrupt = xps_timer_1_Interrupt
END
```

- **xps_uartlite v1.00a**- The XPS Universal Asynchronous Receiver Transmitter (UART) Lite Interface connects to the PLB bus. It provides the controller interface for asynchronous serial data transfer. In our design, two uartlite components (RS232_Uart_0 and RS232_Uart_1) are created as the stdin/stdout interface of each MB. Combined with the Hypertrm, we can check the results of elf executable file for each MB. Since only one physical RS232 interface exists on the GR-CPCI-XC4V board, it is necessary to create an uart_selector to choose the display of each UART.

```
BEGIN xps_uartlite
 PARAMETER INSTANCE = RS232_Uart_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_ODD_PARITY = 0
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0x84000000
 PARAMETER C_HIGHADDR = 0x8400ffff
 BUS_INTERFACE SPLB = mb_plb_0
 PORT RX = uart_selector_0_rx_A
 PORT TX = uart_selector_0_tx_A
END

BEGIN xps_uartlite
 PARAMETER INSTANCE = RS232_Uart_1
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BAUDRATE = 9600
 PARAMETER C_DATA_BITS = 8
 PARAMETER C_ODD_PARITY = 0
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0x84000000
 PARAMETER C_HIGHADDR = 0x8400ffff
 BUS_INTERFACE SPLB = mb_plb_1
 PORT RX = uart_selector_0_rx_B
 PORT TX = uart_selector_0_tx_B
END
```

- **uart_selector v1.00a** - This module allows to ”multiplex” two UART Lite Interface

(RS232_Uart_0 and RS232_Uart_1) through one physical port. An input signal will decide which input (A or B) is active. It can be selected by a push button on the board.

```
BEGIN uart_selector
 PARAMETER INSTANCE = uart_selector_0
 PARAMETER HW_VER = 1.00.a
# PORT tx_D = uart_selector_0_tx_D
 PORT tx_C = uart_selector_0_tx_C
 PORT tx_B = uart_selector_0_tx_B
 PORT tx_A = uart_selector_0_tx_A
# PORT rx_D = uart_selector_0_rx_D
 PORT rx_C = uart_selector_0_rx_C
 PORT rx_B = uart_selector_0_rx_B
 PORT rx_A = uart_selector_0_rx_A
 PORT sel_AnB = uart_selector_0_sel_AnB
 PORT sel_CnD = uart_selector_0_sel_CnD
 PORT tx_pin = fpga_0_RS232_Uart_1_TX
 PORT rx_pin = fpga_0_RS232_Uart_1_RX
END
```

- **proc_sys_reset v2.00a**- This is a reset control block which is very useful. It has a port for an external reset input, which will perform resets to the internal MB components, the processor bus structures, and the peripherals in a very specific order. It also has internal reset ports which connect to the processor itself. In our design, one reset control block is created to control all the system (MB0, MB1 and LEON3 system), aims to synchronize the three systems.

```
BEGIN proc_sys_reset
 PARAMETER INSTANCE = proc_sys_reset_0
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_EXT_RESET_HIGH = 0
 PARAMETER C_NUM_BUS_RST = 1
 PORT Slowest_sync_clk = sys_clk_s
 PORT Dcm_locked = Dcm_all_locked
 PORT Ext_Reset_In = sys_rst_s
 PORT MB_Reset = mb_reset
 PORT Bus_Struct_Reset = sys_bus_reset
# PORT MB_Debug_Sys_Rst = Debug_SYS_Rst
 PORT Peripheral_Reset = sys_periph_reset
END
```

- **clock_generator v2.01a** - The clock generator module provides clocks according to the clock requirements. In our design, the clock generator module generates 100 MHz to all the processor systems (MB0, MB1, LEON3s), aims to synchronize all the cores.

- **leon3mp_0 v2.00a** - The leon3mp module has four LEON3 processors work together with their private i/dcache, but share the bus, L2 memory (DPRAM_0 and DPRAM_1) and L3 memory (SRAM0) inside as it is mentioned in Section 4.3.2. The elf executable file (software routine) fits into the SRAM0 with the capacity of 8MB. The input data (CC-WAV) fits into the DPRAM_0 with the

```
BEGIN clock_generator
 PARAMETER INSTANCE = clock_generator_0
 PARAMETER HW_VER = 2.01.a
 PARAMETER C_EXT_RESET_HIGH = 1
 PARAMETER C_CLKIN_FREQ = 50000000
 PARAMETER C_CLKOUT0_FREQ = 100000000
 PARAMETER C_CLKOUT0_BUF = TRUE
 PARAMETER C_CLKOUT0_PHASE = 0
 PARAMETER C_CLKOUT0_GROUP = NONE
 PARAMETER C_CLKIN_BUF = FALSE
 PORT CLKOUT0 = sys_clk_s
 PORT CLKIN = dcm_clk_s
 PORT LOCKED = Dcm_all_locked
 PORT RST = net_gnd
END
```

capacity of 1 KB. And the output data (result) fits into the DPRAM_1 with the capacity of 1 KB. In addition, there are two grgpios and one uart connecting to the AHB bus. This IP core is created by the netlist file (leon3mp.edf) and integrated into the XPS project.

```
BEGIN leon3mp
 PARAMETER INSTANCE = leon3mp_0
 PARAMETER HW_VER = 2.00.a
 BUS_INTERFACE MFSL1 = i_fsl_v20_1
 BUS_INTERFACE SFSL0 = i_fsl_v20_0
 PORT clk = sys_clk_s
 PORT resetn = sys_rst_s
 PORT FSL_Rst = sys_periph_reset
 PORT led = leon3mp_0_led
 PORT errorn = leon3mp_0_errorn
 PORT address = leon3mp_0_address
 PORT data = leon3mp_0_data
 PORT ramsn = leon3mp_0_ramsn
 PORT ramoen = leon3mp_0_ramoen
 PORT rwen = leon3mp_0_rwen
 PORT romsn = leon3mp_0_romsn
 PORT iosn = leon3mp_0_iosn
 PORT oen = leon3mp_0_oen
 PORT read = leon3mp_0_read
 PORT writen = leon3mp_0_writen
 PORT brdyn = leon3mp_0_brdyn
 PORT bexcn = leon3mp_0_bexcn
# ##leon3mp_0_gpio0
# PORT gpio0 = leon3mp_0_gpio0
# ##fpga_0_LEDs_2Bit_GPIO_IO
# PORT gpio1 = leon3mp_0_gpio1
 PORT emdio = leon3mp_0_emdio
 PORT etx_clk = leon3mp_0_etx_clk
 PORT erx_clk = leon3mp_0_erx_clk
 PORT erxd = leon3mp_0_erxd
 PORT erx_dv = leon3mp_0_erx_dv
 PORT erx_er = leon3mp_0_erx_er
 PORT erx_col = leon3mp_0_erx_col
 PORT erx_crs = leon3mp_0_erx_crs
 PORT etxd = leon3mp_0_etxd
 PORT etx_en = leon3mp_0_etx_en
 PORT etx_er = leon3mp_0_etx_er
 PORT emdc = leon3mp_0_emdc
 PORT sdcke = leon3mp_0_sdcke
 PORT sdcsn = leon3mp_0_sdcsn
 PORT dsuact = leon3mp_0_dsuact
 PORT dsubre = leon3mp_0_dsubre
 PORT dsuen = leon3mp_0_dsuen
 PORT txd1 = uart_selector_0_tx_C
 PORT rxd1 = uart_selector_0_rx_C
 PORT FSL1_M_Write_debug = leon3mp_0_FSL1_M_Write_debug
 PORT FSL1_M_Data_debug = leon3mp_0_FSL1_M_Data_debug
 PORT FSL1_M_Clk_debug = leon3mp_0_FSL1_M_Clk_debug
END
```

After overview the MHS design of MB systems, the next step is simply to run the options of ”Hardware → Generate Netlist” and ”Hardware → Generate Bitstream”.

Afterwards, we operate the option of ”Device Configuration → Download bitstream” on the target board.

4.3.3.3 LEON3 System and MB System Integration

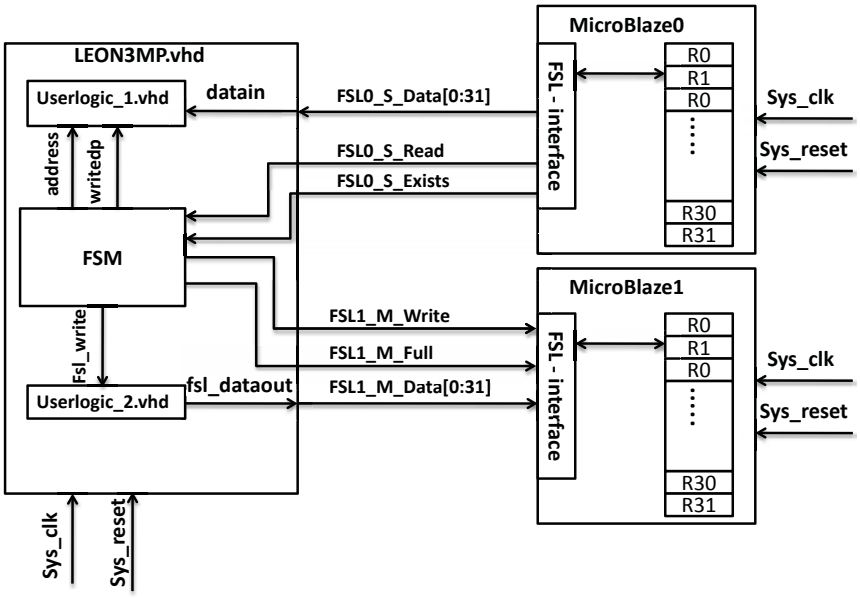


Figure 4.24: FSL detailed connections between LEON3 system and MB system.

Fig.4.24 presents the detailed connections between the LEON3 system and MB system, where MB0 is the Master of the Userlogic_1 in LEON3 system, and the Userlogic_2 combined with FSM is the Master of MB1. The integration of LEON3 system and MB system works as co-processor system, which can save the timing overhead on the bus transactions. In order to integrate the LEON3 netlist (*leon3mp.edf*) as an IP core into the XPS project, there are three steps design as follows:

1. Integrated **leon3mp v2.00a** as an IP core into the XPS project (*system.xmp*):

   We can invoke the *Create and Import Peripheral wizard* from XPS to integrate the LEON3 netlist (*leon3mp.edf*) into the XPS project. By selecting the “*Hardware → Create or Import Peripheral*“, click ”*Next*” to select ”*Create templates for a new peripheral*” option in *Select flow* as shown in Fig.4.25. Then we give the Name “*leon3mp*“ which is the same name as the top-level VHDL design entity (the version is 2.00a), and choose the bus as FSL for this peripheral. We assign the numbers of 32-bit input and 32-bit output are both “*1*“. Finally, it will automatic “*generate ISE and XST project files*“ to help us implement the peripheral by XST flow under the directory of $your\_project\_path/pcores$ and “*generate template driver files*“ to help us implement software interface under the directory of $your\_project\_path/drivers$.

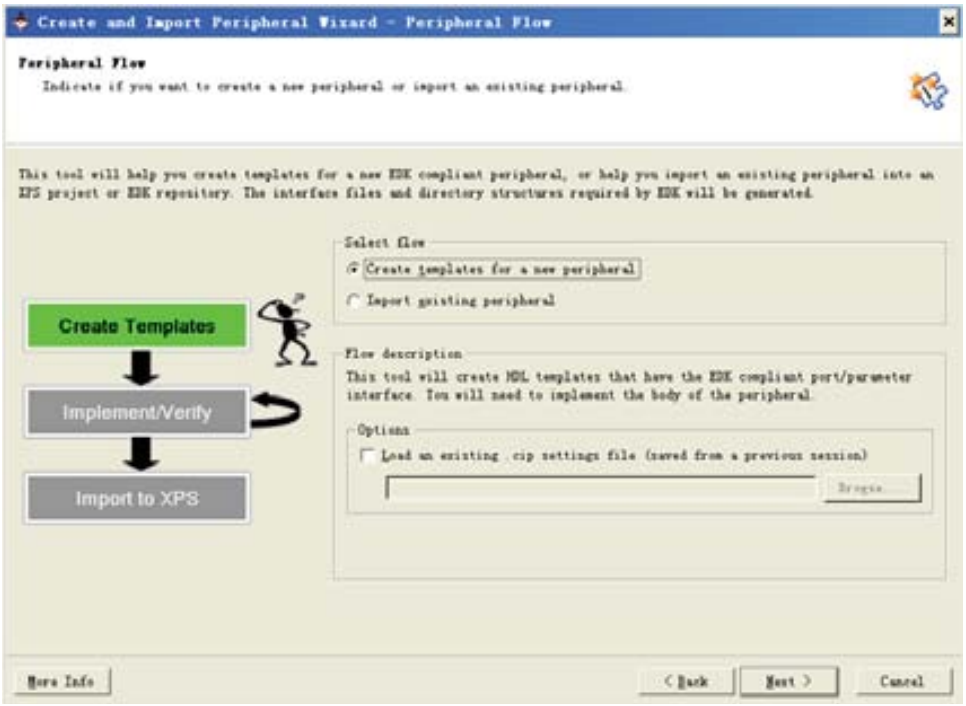


Figure 4.25: Create templates for a new peripheral.

Afterwards, we can copy all the VHDL codes of the LEON3 top-level and IP cores design in *hdl* folder in the directory of $/pcores/leon3mp\_v2\_00\_a/vhd$, and create the Microprocessor Peripheral Definition (MPD) file, Peripheral Analyze Order (PAO) file and Black Box Definition (BBD) file in the *data* folder in the same directory. The MPD file defines the interface of the peripheral. The PAO file contains a list of HDL files that are needed for synthesis, and defines the analyze order for compilation. The BBD file manages the file locations of optimized hardware netlists for the black-box sections of our IP core design.

Finally we copy the *leon3mp.edf* file created by Synplify Pro in the *netlist* folder. The file structure in the XPS project should look like in Fig.4.26. Then repeat the last step again by selecting "*Import existing peripheral*" in order to regenerate the *leon3mp v2.00a* again. When we generate it successfully, the *leon3mp v2.00a* IP core will appear in the XPS project as shown in Fig.4.27.

2. Integration in SW: The next step is to test the LEON3 IP core by the software. The C program is very simple, just writes some data to the LEON3 core and reads it back by MBs. For writing into the LEON3 system by FSL, the predefined functions can be found in a dedicated directory ($your\_project\_path/drivers/leon3mp\_v2\_00\_a$). And the non-blocking write and read commands are defined in the $mb\_interface.h$ file in the directory ($your\_project\_path/drivers/microblaze\_0/include$). For the reference design, we will introduce in details in the next Section.

3. Verification in HW: The verification of the hardware can be done by GRMON. The

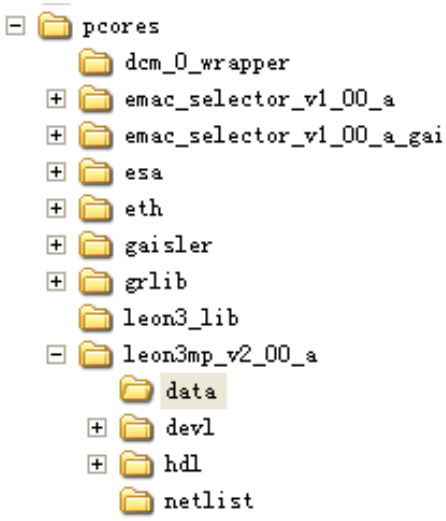


Figure 4.26: File structure of IP cores in LEON3 system.

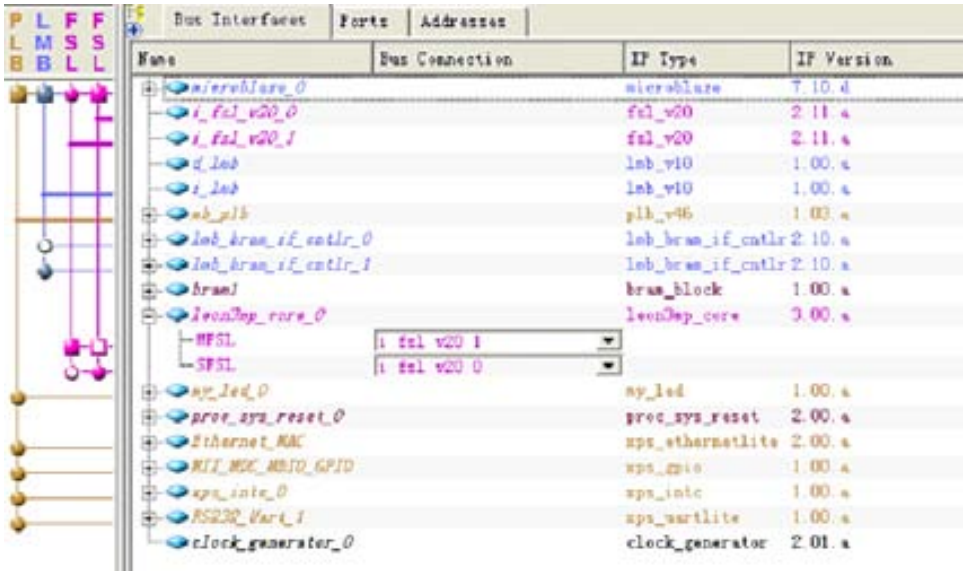


Figure 4.27: FSL connections between Microblaze_0 and leon3mp_core_0 in Xilinx XPS Project.

aim is to verify the data transfer between the LEON3 system and the MB system. For the reference design, we will introduce in details in Section 4.4.

4.4 Seven Software Routines

In the HTPCP system design, the integration between two processors system should be validated by the software routines, in particular the interface designs (MPI, FSL, GPIO, Multi-processor Interrupt). The cables and design tools that needed for the software routines are presented in Table 4.V.

Table 4.V: The cables and design tools are needed for the software routines.

Cables	2 crossover Ethernet Cables	Eth0 to GOLD-RTR/ Eth1 to Control PC	Routine 1-2
	1 Xilinx USB JTAG cable	Download and debug the program files	Routine 1-7
	1 RS232 USB cable	Serial communications (HyperTerminal)	Routine 1-7
Design Tools	XPS 10.1	Hardware Design	Routine 1-7
	SDK 10.1	Software Design	Routine 1-7
	GRMON	Debug and monitor the LEON3 processors by DSU	Routine 3-7
		Download and execute the applications	
		Access to all the memory of LEON3	

The following SW routines are located under the applications tab in the left panel of Xilinx EDK, and complied by Xilinx SDK or LEON3 *sparc-elf-gcc* compiler:

- Routine1: GOLD-RTR → ETH0 → MB0 → BUF1 → MB1 → ETH1 → CONTROL PC.
- Routine2: CONTROL PC → ETH1 → MB1 → BUF0 → MB0 → ETH0 → GOLD-RTR.
- Routine3: MB0 → FSL_Slave link → DPRAM_0 (LEON3).
- Routine4: DPRAM_1 (LEON3) → FSL_Master link → MB1.
- Routine5: GR_GPIO0 (LEON3) → XPS_GPIO0 (MB0).
- Routine6: XPS_GPIO1 (MB1) → GR_GPIO1 (LEON3).
- Routine7: Four LEON3 processors controlled by MP IRQ Controller.

Routine 1 and 2 are used to test the MPI transmission in TEs design, Routine 3 and 4 are used to test the two FSL links transmission between PE and TE design, Routine 5 and 6 are used to test the two gpio connections between PE and TE design. Routine 7 is used to test the multi-processor interrupt controller in PE design.

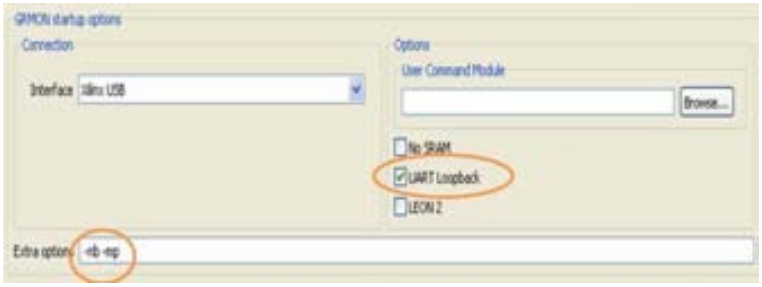
Table 4.VI: The executable files location, the initial address, and the initial components of LEON3 system in seven SW routines.

Routines	MB0 Bram	MB1 Bram	LEON3 Sram	Initial Addresses (LEON3)	Initial Components (LEON3)
1	Routine_1.elf				
2		Routine_2.elf			
3	Routine_3.elf			0xa0000000	DPRAM_0
4		Routine_4.elf	TEST4.EXE	0xa0100000	DPRAM_1
5	Routine_5.elf		TEST5.EXE	0x80000900	GR_GPIO0
6		Routine_6.elf	TEST6.EXE	0x80000800	GR_GPIO1
7			IRQMP.EXE	0x80000200	MP_IRQ CTRL

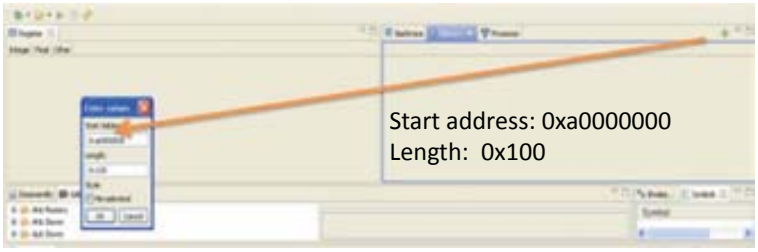
These Routines will be compiled into six ELF files (*Routine_n.elf*) under the directories of their SDK projects, and stored in the BRAMs of MB0 and MB1

separately. The EDK supports two processors to work simultaneously, so that we can obtain the results of two routines by the UART0 of MB0 and the UART1 of MB1. Since there is only one UART port on the design board, a UART selector is designed to select which UART will display on the hyper-terminal by the push button S1. In addition, four ELF files for LEON3 system (*TEST4.EXE, TEST5.EXE, TEST6.EXE and IRQMP.EXE*) are compiled by **space-elf-gcc**, that can be used to execute the following tasks of LEON3 processors:

- TEST4.EXE: write the packet to DPRAM_1.
- TEST5.EXE: read the data from GRGPIO0.
- TEST6.EXE: write the data to GRGPIO1.
- IRQMP.EXE: control the task flow of each LEON3.



a) Initial the GRMON



a) Initial the result window at 0xa000000

Figure 4.28: GRMON initialization at 0xa0000000 and 0xa0100000.

Fig.4.28 shows the method to display the contents of memory, register file of LEON3 components by GRMON. The initialization of GRMON is configured by the non-breakpoint (-nb), multi-processing mode (-mp) and UART loopback options. Then we set the initial addresses of the memory window at 0xa0000000 (DPRAM_0) and 0xa0100000 (DPRAM_1). The LEON3 system will display in the GRMON by the same bitstream file on board, which means that the integration of MB system and LEON3 system is successful. By the same way, we can display the contents of GRGPIO0 and GRGPIO1 by setting the initial addresses of the memory window at 0x80000900 (GRGPIO0) and at 0x80000800 (GRGPIO1); Or display the contents of the register file of MP interrupt controller by setting the initial address of memory window at 0x80000200 (MP interrupt controller).

4.4.1 MPI Transmission between TEs

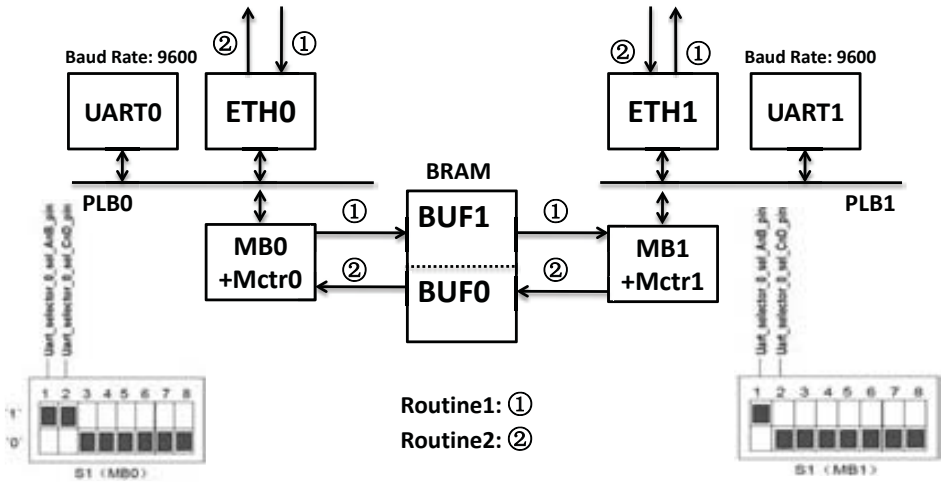


Figure 4.29: Routine 1 and Routine 2 diagram.

Routine 1 and Routine 2 aim at validating the lossless communication between two Ethernet devices via the MPI interface as shown in Fig.4.29.

- Routine 1 is controlled by MB0 that guarantee the packets transmission from GOLD-RTR to Control PC:
  - Initialize the memory (BRAM). (**init_memory** subroutine)
  - Initialize the network. (**init_network** subroutine)
  - The main program enters a finite loop to send the ten packets (0,1,2...9) to BUF1. (**write_to_buffer1** subroutine)
  - MB0 receives the packages from BUF0. (**read_from_buffer0** subroutine)
- Routine 2 is controlled by MB1 that guarantee the packets transmission from Control PC to GOLD-RTR:
  - Initialize the memory (BRAM). (**init_memory** subroutine)
  - Initialize the network. (**init_network** subroutine)
  - The main program enters a finite loop to send the ten packets (10,11,12...19) to BUF0. (**write_to_buffer0** subroutine)
  - MB1 receives the packages from BUF1. (**read_from_buffer1** subroutine)

While the Routine 1 and 2 are working simultaneously, the packets transmission between two Ethernet devices transparently. The results in Table 4.VII present that the two Ethernet devices can achieve no package loss transmission.

Table 4.VII: Test results of Routine 1 and 2.

Routine 1	MB0 write to BUF1	0	1	2	3	4	5	6	7	8	9	0
Routine 2	MB1 read from BUF1		0	1	2	3	4	5	6	7	8	9
Routine 2	MB1 write to BUF0	10	11	12	13	14	15	16	17	18	19	10
Routine 1	MB0 read from BUF0		10	11	12	13	14	15	16	17	18	19

4.4.2 FSL Transmissions between PE and TE Design

Routine 3 and 4 aim at validating the data transmission between LEON3MP system and MB system via two FSL links as shown in Fig.4.30.

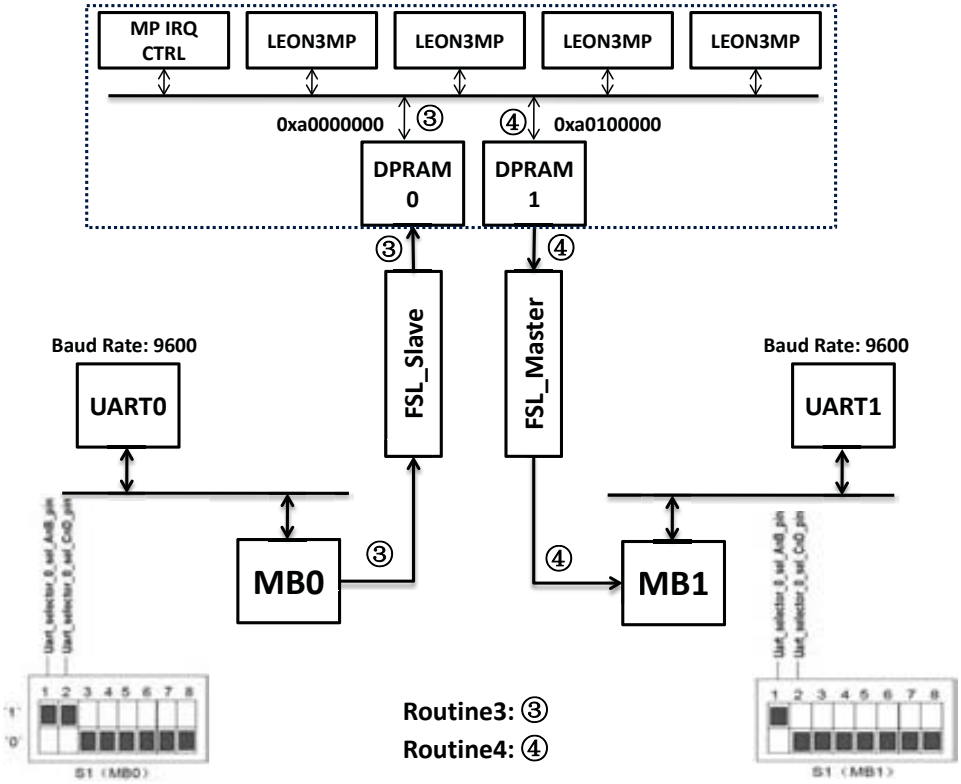


Figure 4.30: Routine 3 and Routine 4 diagram.

- Routine 3 is controlled by MB0 that guarantee the data transmission from the FSL_Slave link to the $DPRAM\_0$ (LEON3):
  - Write the data from MB0 to $DPRAM\_0$ (0xa0000000) by FSL_Slave link (i_fsl_v20_0).

    microblaze_nbwrite_datafsl(i+0xAAAAAAAA,0);

    * The first command indicates the data structure written to the FSL_Slave link ($i\_fsl\_v20\_0$).

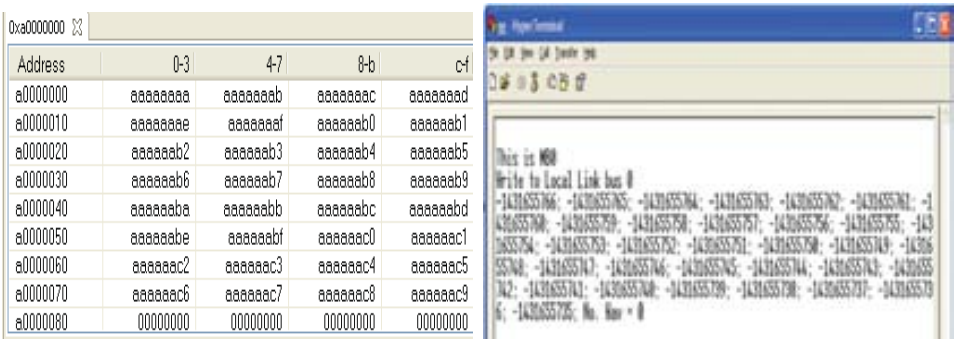
    - The second command indicates the number of the FSL links, in our case is 0.
  - Check if the written data is correct in $DPRAM\_0$ (0xa0000000) by GRMON.

- Routine 4 is controlled by the MB1 that guarantee the data transmission from $DPRAM\_1$ (LEON3MP)to the FSL_Master link.
  - Compile the C code (TEST5.c) by **space-elf-gcc** compiler, and generate the elf executable file (TEST5.EXE).
  - Load and run the elf executable file (TEST5.EXE) by GRMON, which is used to write the data from LEON3s to $DPRAM\_1$ (0xa0100000).
  - Read the data from $FSL\_Masterlink$ (i_fsl_v20_1).

    microblaze_nbread_datafsl(data_back, 1);

    - The first command is the variable which stores the data structure that read from the FSL_Master link ($i\_fsl\_v20\_1$).
    - The second command indicates the number of FSL link, in our case is 1.
  - Check if the read data is correct by UART1.

The results in Fig.4.31-a show that the received data in $DPRAM\_0$ ($0xa0000000$) by the GRMON, which is transmitted from the FSL_Slave link to the $DPRAM\_0$ (LEON3). The results in Fig.4.31-b shown that the data which is written from MB0 to FSL_Slave link by Routine3.elf. The two results are coherent, which can provide the validation of FSL_Slave link transmission between two processor systems in Routine 3.



a) Routine 3 result in GRMON (LEON3) b) Routine 3 result in HyperTrm (MB0)

Figure 4.31: Routine 3 results in GRMON and Hypertrm.

The only difference is the format of displaying as shown in Table 4.VIII. GRMON displays in *hex-decimal* format, and Hypertrm displays in *integer* format. Note that the Microblaze is 32 bits processor, the same as the FSL_Slave link transmission, therefore, this system can only handle 32-bit data transfer.

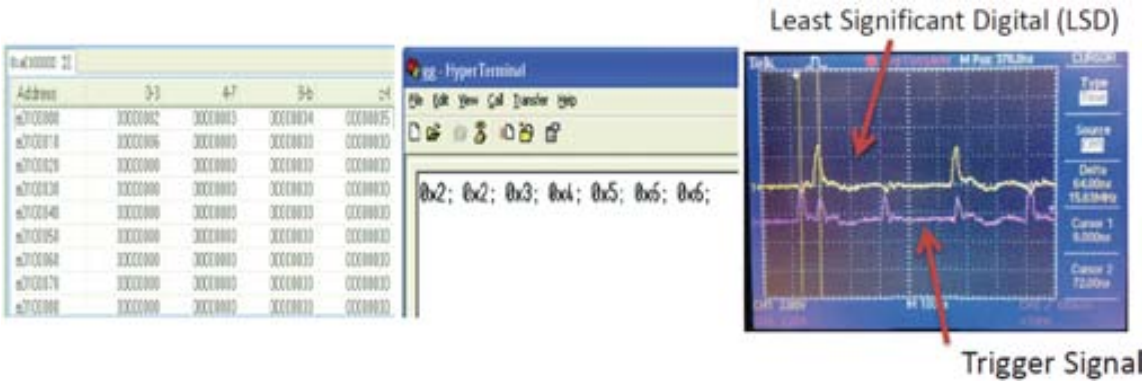
The results in Fig.4.32-a show that the written data in $DPRAM\_1$ (0xa0100000) by the elf executable file (TEST5.EXE), which is captured by GRMON. The results

Table 4.VIII: Test results of Routine 3.

Result in DPRAM_0 (HEX)	aaaaaaaa	aaaaaaab	aaaaaaac	aaaaaaad	aaaaaaae
Result in MB0 (DEC)	−1431655766	−1431655765	−1431655764	−1431655763	−1431655762
Result in MB0 (HEX)	ffffffffaaaaaaaa	ffffffffaaaaaaab	ffffffffaaaaaaac	ffffffffaaaaaaad	ffffffffaaaaaaae

Table 4.IX: Test results of Routine 4.

Optimization level	O0	O1	O2
Test 1 (write 2,3,4,5,6)	0x2 0x2 0x3 0x4 0x5 0x6 0x6	0x2 0x2 0x3 0x4 0x4 0x5 0x6	0x2 0x2 0x3 0x4 0x4 0x6 0x5
Test 2 (write 15,16,17,18,19)	0xf 0x10 0x11 0x12 0x13	0xf 0xf 0x10 0x10 0x11 0x11 0x12 0x12 0x13 0x13	0xf 0xf 0x10 0x10 0x11 0x11 0x12 0x12 0x13 0x13



a) Routine 4 result in GRMON (LEON3) b) Routine 4 result in Hypertrm (MB1) c) LSD test in oscilloscope

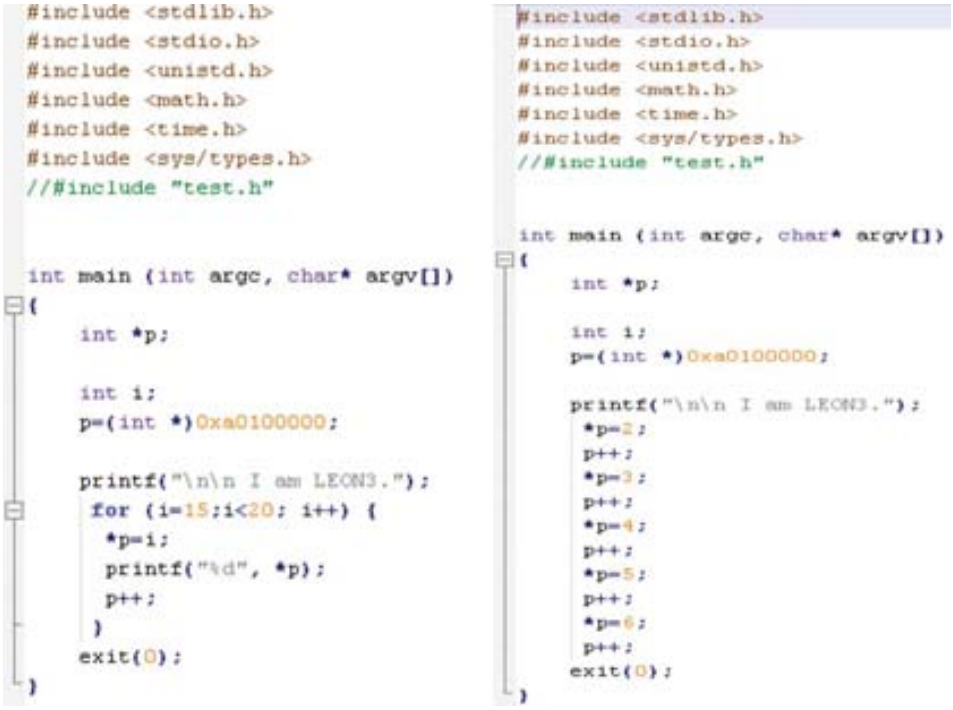
Figure 4.32: Routine 4 results in GRMON and Hypertrm.

in Fig.4.32-b show that the transferred data from FSL_Master Link to MB1. Note that the two results are not coherent, initially, we infer that the reason is the synchronization issue of the two processor system.

Fig.4.32-c presents the the Least Significant Digital (LSD) tested by oscilloscope, the LSD result (01010) is coherent with the result 010, 011, 100, 101, 110 triggered by a trigger signal. Therefore, we conclude that the problem is not the hardware factor but the software factor. Two tests in Fig.4.33 can explore the software factor by choosing the compiler optimization levels (-O0) and the loop format in C code. The results in Table 4.IX show that the *Test 2* compiled by O0 can obtain the correct results in Routine 4.

4.4.3 GPIO Connection between PE and TE

Routine 5 and 6 aim at validating the communication between the GPIOs back and forth in LEON3MP system and MB system as shown in Fig.4.34.



```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include <sys/types.h>
//#include "test.h"


int main (int argc, char* argv[])
{
    int *p;

    int i;
    p=(int *)0xa0100000;

    printf("\n\n I am LEON3.");
     for (i=15;i<20; i++) {
      *p=i;
      printf("%d", *p);
      p++;
     }
    exit(0);
}
```

Test 1

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <time.h>
#include <sys/types.h>
//#include "test.h"


int main (int argc, char* argv[])
{
    int *p;

    int i;
    p=(int *)0xa0100000;

    printf("\n\n I am LEON3.");
     *p=2;
     p++;
     *p=3;
     p++;
     *p=4;
     p++;
     *p=5;
     p++;
     *p=6;
     p++;
    exit(0);
}
```

Test 2

Figure 4.33: Two software tests for Routine 4.

- Routine 5 is controlled by MB0 that guarantee the data transmission back and forth from the GR_GPIO0 (LEON3) to XPS_GPIO0 (MB0):
  - Compile the C code (TEST4.c) by **space-elf-gcc** compiler, and generate the elf executable file (TEST4.EXE).
    - Initial the registers of GR_GPIO0.

```
volatile unsigned int *data =(int *) 0x80000900;
//<gpio register's base address>;
volatile unsigned int *output = (int *) 0x80000904;
//<gpio register's base address + 0x4>;
volatile unsigned int *direction =(int *) 0x80000908;
//<gpio register's base address + 0x8>;
```

    - Assign the value to output data registers.

```
/* Assign the value to output data registers */
  *output = 0x12345678;
/* Enable all outputs */
  *direction = 1;
```

  - Load and run the elf executable file (TEST4.EXE) by GRMON, which is used to write the data from LEON3s to GR_GPIO0 (0x80000900).
  - Check if the read data is correct by UART0.

```
gpio_check = XGpio_DiscreteRead(&GpioOutput0, 1);
```

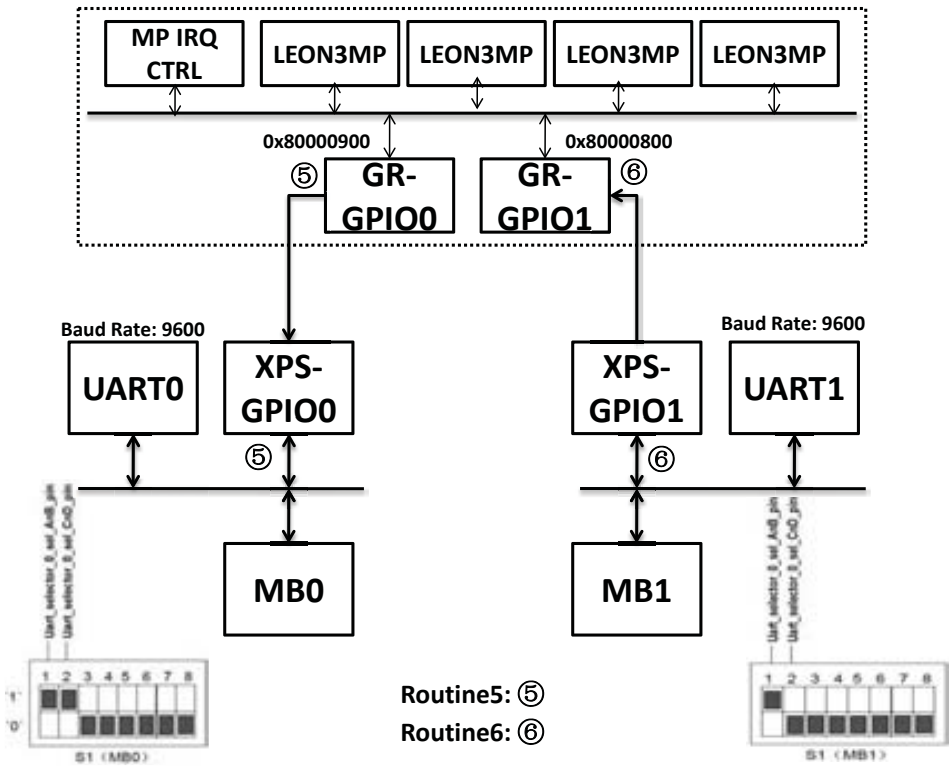


Figure 4.34: Routine 5 and Routine 6 diagram.

- The first command is the address of XPS_GPIO0.
- The second command indicates the channel of the XPS_GPIO0. In our case, we use channel 1, since XPS_GPIO0 is single channel.

- Routine 6 is controlled by MB1 that guarantee the data transmission back and forth from the XPS_GPIO1 (MB1) to GR_GPIO1 (LEON3):

  – Write the data from the XPS_GPIO1 (MB1) to GR_GPIO1 (LEON3).

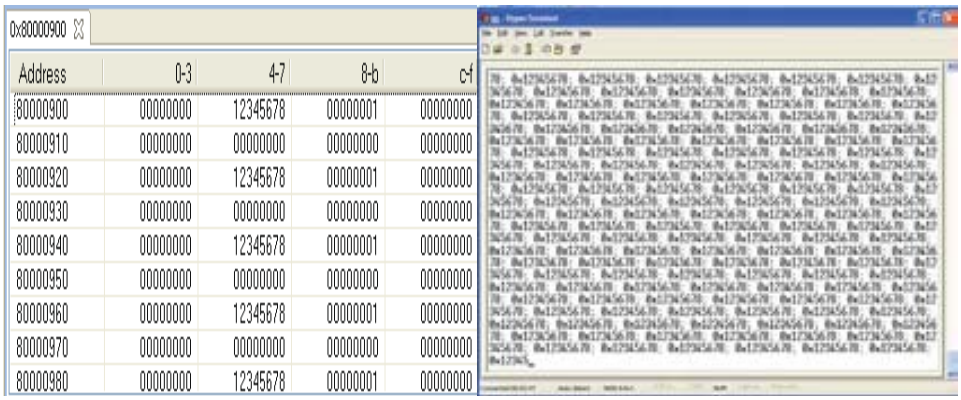
```
XGpio_SetDataDirection(&GpioOutput1, 1, 0x0);
XGpio_DiscreteWrite(&GpioOutput1, 1, data);
data++;
```

  - The first command indicates the data direction is output.
  - The second command indicates the address of XPS_GPIO1, the channel of XPS_GPIO1, and the data structure of the transmitted data.
  - The third command indicates the transmitted data.

  – Check if the read data in GR_GPIO1 (0x80000800) is correct by the elf executable file (*TEST6.EXE*) in GRMON. Compile the C code (*TEST6.c*) by **space-elf-gcc** compiler, and generate the elf executable file (*TEST6.EXE*).

```
volatile unsigned int *p =(int *) 0x80000800;
//<gpio register's base address>;
volatile unsigned int *direction =(int *) 0x80000808;
//<gpio register's base address + 0x8>;
```

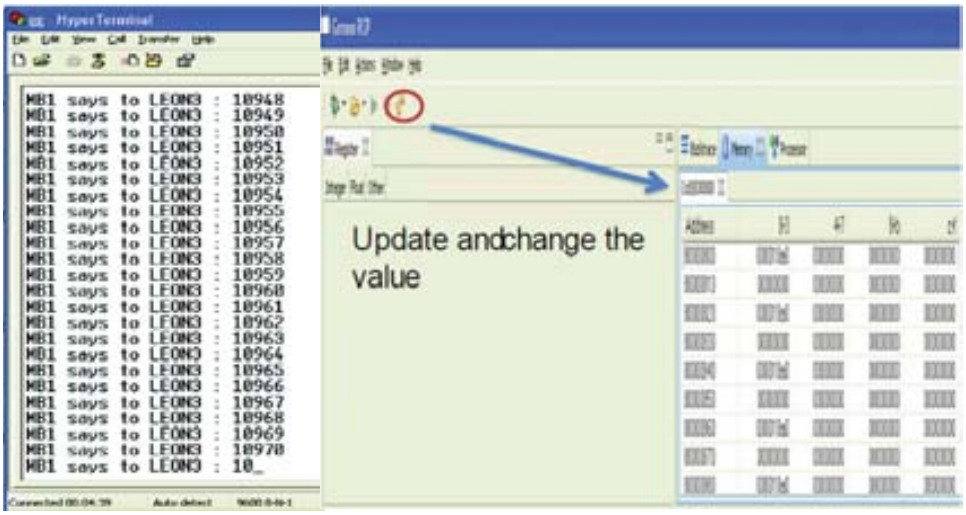
```
/* Enable all outputs */
*direction = 0;
while (1)
{
 printf("I received: %d", *p);
}
exit(0);
```



a) Routine 5 result in GRMON b) Routine 5 result in Hypertrm

Figure 4.35: Routine 5 results in GRMON and Hypertrm.

Fig.4.35-a present the written data (0x12345678) by the elf executable file (TEST4.EXE) to GR_GPIO0 (0x80000900) in GRMON. The results in Fig.4.35-b show that the captured data from GR_GPIO0 (LEON3) to XPS_GPIO0 (MB0) by UART0. The two results are coherent, which can provide the validation of the GPIO connection from GR_GPIO0 (LEON3) to XPS_GPIO0 (MB0) works correctly in Routine 5.



a) Routine 6 results in Hypertrm b) Routine 6 results in GRMON

Figure 4.36: Routine 6 results in Hypertrm and GRMON.

Fig.4.36-a shows the increasing data written by MB1 to XPS_GPIO1. The results in Fig.4.36-b present the captured data at GRGPIO1 (0x80000800) in GRMON (constantly update). Also we can capture the data by the elf executable file (TEST6.EXE). The two results are coherent, which can provide the validation of the GPIO connection from GRGPIO1 (LEON3) to XPSGPIO1 (MB1) works correctly in Routine 6.

4.4.4 Multi-processor Interrupt Controller between PEs

Routine 7 aim at validating the control flow of the multi-processor interrupt controller with two LEON3 processors as shown in Fig.4.37.

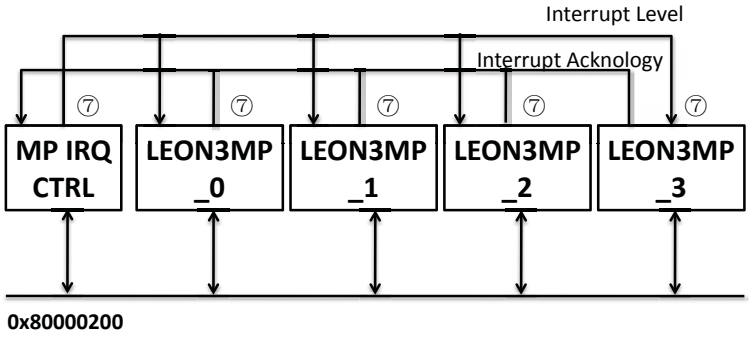


Figure 4.37: Routine 7 diagram.

The processor status can be monitored through the Multiprocessor Status Register presented in Fig.4.38. The STATUS field in this register indicates if a processor is halted (1) or running (0). A halted processor can be reset and restarted by writing a 1 to its status field. After reset, all processors except processor 0 are halted. When the system is properly initialized, processor 0 can start the remaining processors by writing to their STATUS bits. As shown in Fig.4.39, we can always use these statements to detect the CPU ID, and write the function for each processor.

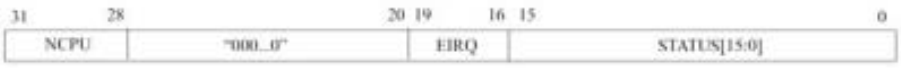


Figure 171. Multiprocessor status register

[31:28] NCPU. Number of CPU's in the system -1 .
[19:16] EIRQ. Interrupt number (1 - 15) used for extended interrupts. Fixed to 0 if extended interrupts are disabled.
[15:1] Power-down status of CPU [*n*]: '1' = power-down, '0' = running. Write with '1' to start processor *n*.

Figure 4.38: Multiprocessor status register.

Fig.4.40-a presents that the system has two processors. Note that the NCPU indicates the number of CPUs is 2. And both of them are running. However, in Fig.4.40-b, we halt both processors by the following code. These two results can provide the validation of multi-LEON3 works in parallel in Routine 7.

```
//Get cpu id

static int cpu_id, id = 0xF;

__asm__ __volatile__("rd %%asr17,%0\n\t":"=r" (id) : );

cpu_id = ((id >> 28) & 0xff);
```

Figure 4.39: Detect the CPU ID in C.

```
volatile int* CPU_STAT = (volatile int*) 0x80000210;

static void cpu_0(void);//Task for cpu 0

static void cpu_1(void);//Task for cpu 1

catch_interrupt(irqhandler(0), 0);
printf("\n\n Irqhandler is 0x%x.\r\n", irqhandler(0));
catch_interrupt(irqhandler(1), 1);
printf("\n\n Irqhandler is 0x%x.\r\n", irqhandler(1));
```

0x80000200

Address	0-3	4-7	8-b	c-f
80000200	00000000	00000000	00000000	00000000
80000210	18000002	00000000	18000002	18000002
80000220	00000000	00000000	00000000	00000000
80000230	18000002	00000000	18000002	18000002
80000240	00000000	00000000	00000000	00000000
80000250	00000000	00000000	00000000	00000000
80000260	00000000	00000000	00000000	00000000
80000270	00000000	00000000	00000000	00000000
80000280	00000000	00000000	00000000	00000000

a) Routine 7 result in GRMON (Before elf app)

0x80000200

Address	0-3	4-7	8-b	c-f
80000200	00000000	00000000	00000000	00000000
80000210	18000000	00000000	18000000	18000000
80000220	00000000	00000000	00000000	00000000
80000230	18000000	00000000	18000000	18000000
80000240	00000000	00000000	00000000	00000000
80000250	00000000	00000000	00000000	00000000
80000260	00000000	00000000	00000000	00000000
80000270	00000000	00000000	00000000	00000000
80000280	00000000	00000000	00000000	00000000

b) Routine 7 result in GRMON (After elf app)

Figure 4.40: Routine 7 results in GRMON.

4.5 Summary

In this Chapter, a novel parallel platform, named as HTPCP, is presented to realize the realtime post-processing for GNSS-R application. Moreover, two problems of HTPCP are proposed and solved, 1) Parallelize the inherent serial output of the GOLD-RTR instrument by TEs and 2) Post-processing the multi-channel (I and Q) correlators in parallel by PEs. We focus on the hardware design of HTPCP, exploit the designs of TEs and PEs and their interfaces (FSL, GPIOs, MPI). Finally the integration of two processor systems are implemented in the XPS project. In order to guarantee the real-time characters and minimize the memory requirement in space level, we operate the

post-processing algorithm on HTPCP and verify the functions of the interface designs and IP cores by seven SW routines.

Web Reference

[*Xilinx EDK*] http://www.xilinx.com/support/documentation/sw_manuals/edk_ctt.pdf

Part V

GNSS-R Post-Processing Application and Implementation

GNSS-R Post-Processing Experiment Results

5.1 Introduction

The HTPCP platform emphasizes on the three characteristics of the post-processing design: SW design flexibility, real-time controllability and parallel processing. The first characteristic interprets that the post-processing algorithm can be changed during the campaign, aims at retrieving the coherence between the test model and the received signal. The second characteristic stresses that the processing time for each waveform can be measured by each clock cycle. The third characteristic indicates that the multi-channel waveforms can be processed in parallel, to achieve the non-dataloss communications and parallel processing performance. In this Chapter, two experiments are presented based on the real campaign data. The results provide an important insight to the operations of the complete system (GOLD-RTR + HTPCP + control PC). The following Sections are organized as follows:

- Description to the experiment constrains, related with the hardware design and the structure of the waveforms.
- Introduction to the post-processing algorithm with three logic steps: data reduction, coherent integration and incoherent integration. Meanwhile, it is possible to straightforwardly extract the related timing parameters.
- Introduction to the demonstration architecture.
- Introduction to the campaign on real data.
- Illustrative numerical results for the two experiments.
- Main conclusions.

5.2 Experiment Constrains

The GNSS signals reflected off the Earth‘s surface not only carry information about the navigation, but also represent the constitution of the reflecting area (ocean, land,

ice and snow etc.). It is an alternative purpose of the GNSS signals. Essentially, the relative delay between the direct and the reflected signals can provide more information i.e. altimetry, roughness, permittivity parameters (temperature, salinity or humidity) etc..

The experiment constrains of the GNSS-R [*GNSS-R*] post-processing application is presented in this Section. The case study focus on the the functions and the purposes of the GOLD-RTR instrument, GR-CPCI-XC4V board and the control PC, where obtained results are a direct outcome of this project. The output waveforms of GOLD-RTR and the output integrated waveforms of the Control PC are addressed in the following subsections, aims at providing the input/output waveform structures of the post-processing algorithm.

5.2.1 Hardware Design Constrains

The hardware design of the entire system (GOLD-RTR instrument + HTPCP + control PC) is used to support the two experiments.

1. GOLD-RTR is an instrument which is capable to correlate the GPS signals that provided by an antennas and report the results as waveforms shown in Fig.5.1. The results of GOLD-RTR are sent by an ethernet network link and received by a control PC.

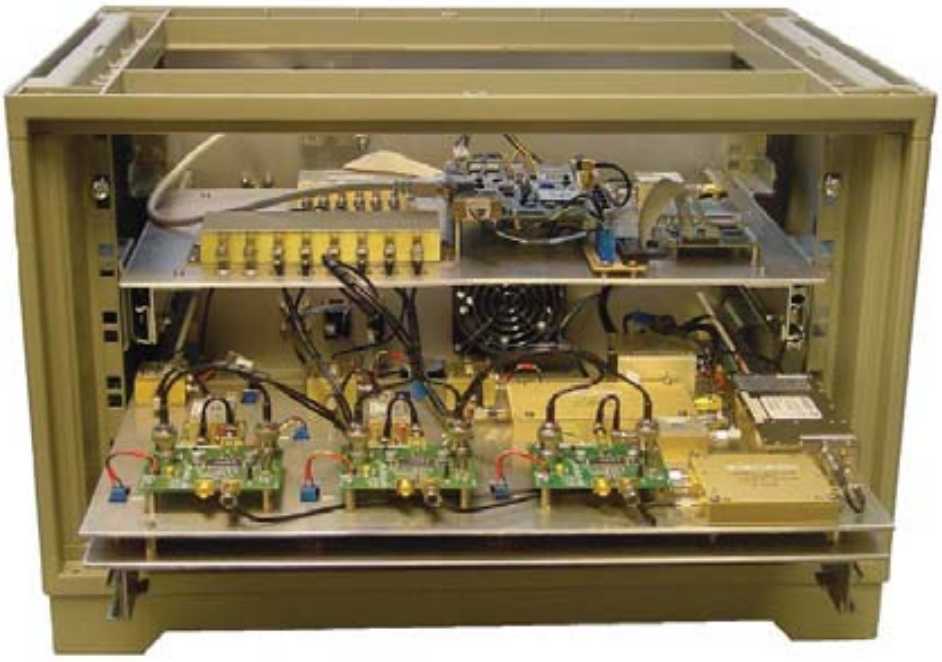


Figure 5.1: GOLD-RTR instrument.

Internally, the GOLD-RTR contains an Altera development board that uses an

FPGA implementing the follow hardware:

- Correlators: contain an array of 10 correlators of 64 items (each item contains 2 bytes).
- C/A code generators: contain an array of 10 C/A code generators.
- The first NIOS processor is to manage the Novatel GPS receiver, aims at programming the correlators and the C/A code generators.
- The other NIOS processor, is to retrieve the UDP/IP packets and manage the system configuration that allow to program the correlators, PRNs and timestamps by the Control PC.

2. The control PC will execute an post-processing algorithm which is capable to integrated and store all the waveforms. Fig.5.2 presents the main window of the control PC used by the operators.

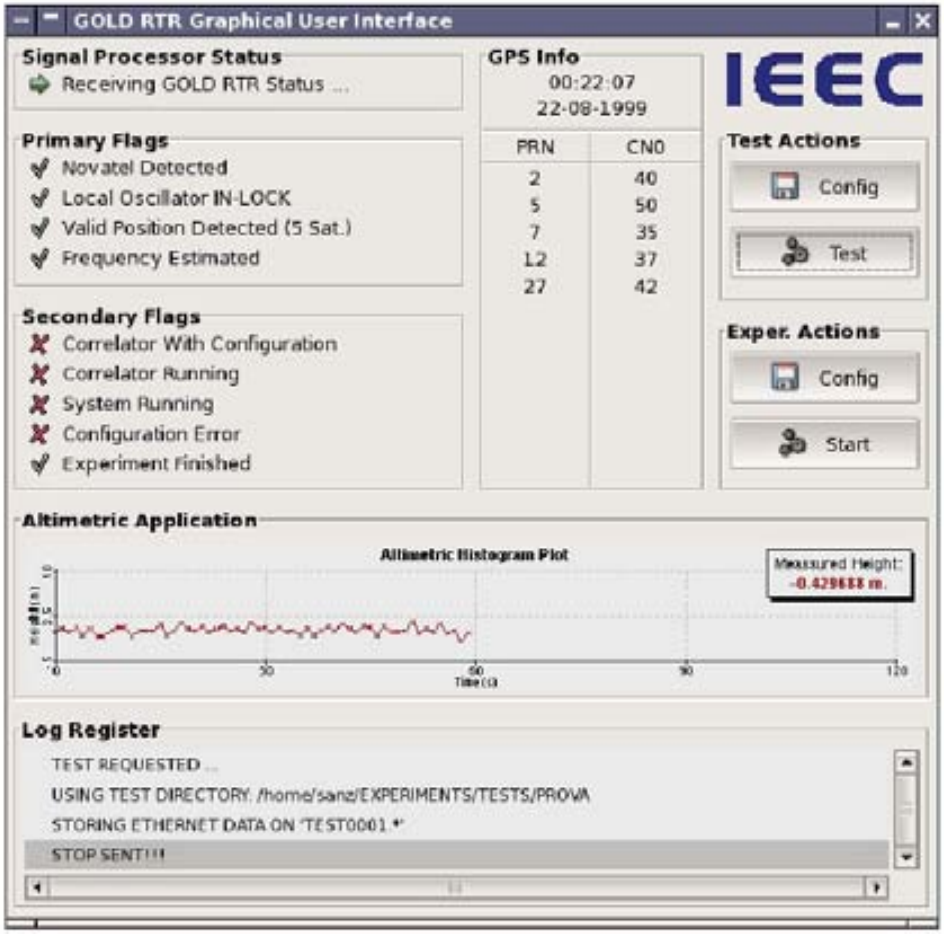


Figure 5.2: The main window of the Control PC.

3. The idea of this experiment is to allow the operators interconnect the GR-CPCI-XC4V board (plus GR-CPCI-2ETH-SRAM-8M extension board) with the GOLD-

RTR and the Control PC as shown in Fig.5.3. It can achieve the following two tasks:

- Configure the ethernet devices on the GR-CPCI-XC4V board (plus GR-CPCI-2ETH-SRAM-8M extension board), in order to bypass the communication between the two original devices (GOLD-RTR and the control PC).
- Reduce the amount of output data of GOLD-RTR by the post-processing algorithm, in order to allow the storage of all integrated waveforms (results) generated by the experiment during the continuous time.

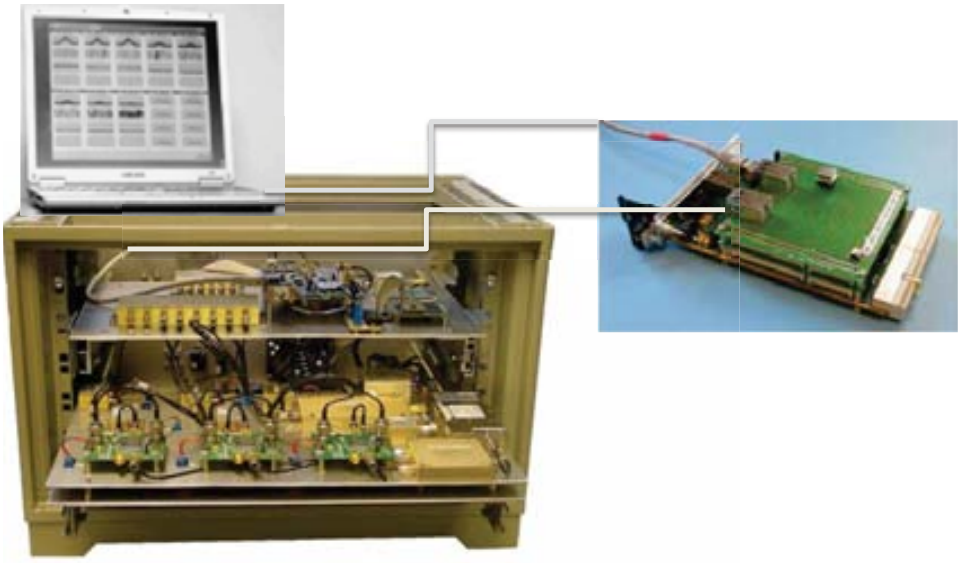


Figure 5.3: The complete system.

5.2.2 GOLD-RTR Output Waveforms

The GOLD-RTR output waveforms carry the information that we required for the post-processing design in GR-CPCI-XC4V board. The waveforms are the correlation results between the signal received by the antenna (converted to digital signal by an 1bit ADC) and the predicted C/A code. The GOLD-RTR uses the GPS information provided by the GPS receiver (Novatel) and some mathematical calculations to do the C/A code prediction. Fig.5.4 shows the structure of the waveforms packet. Each waveform contains 160 bytes, which are separated for 32 bytes of header information and 128 bytes of the waveform contents:

- The header contains the doppler, PRN, elevation, azimuth, timestamps and others information of the processed satellite.
- The contents of the waveforms contain 64 complex numbers (1 bytes for the real part and 1 byte for the imaginary part).

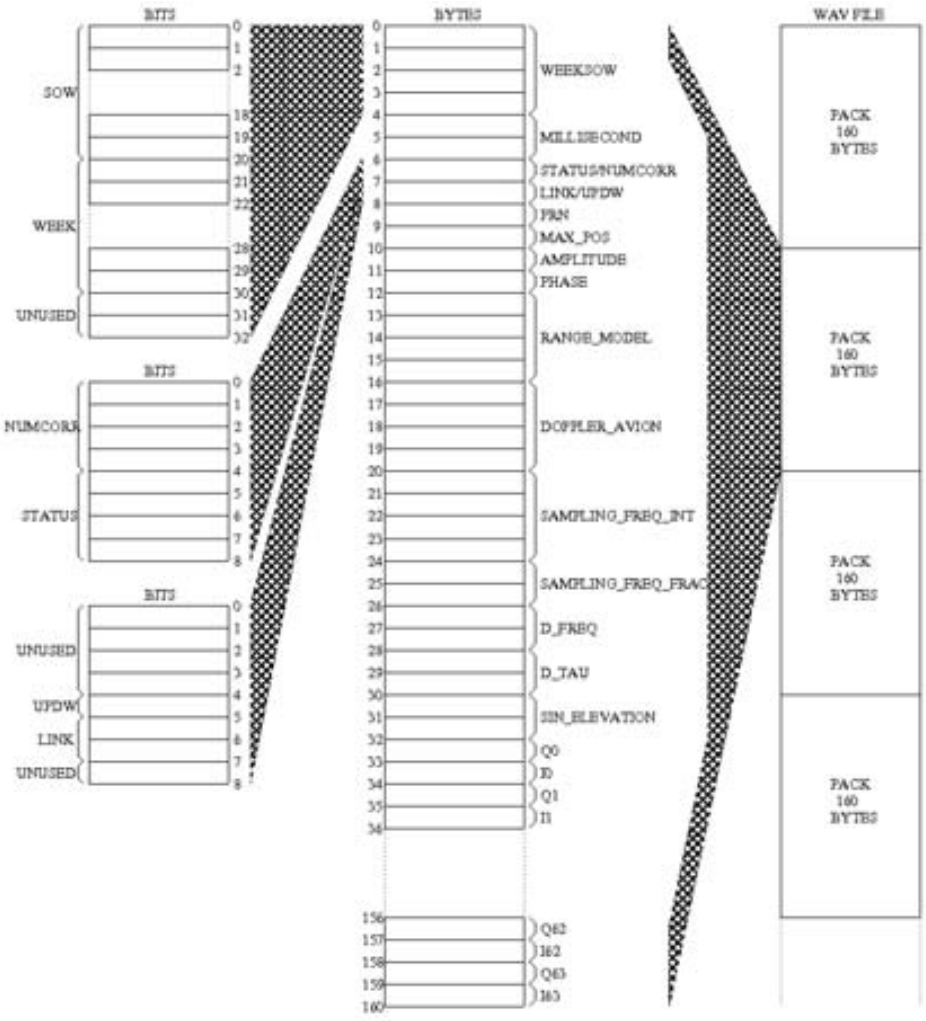


Figure 5.4: The structure of the waveforms packet (160 bytes).

Fig.5.5 presents the graphical representation of the GOLD-RTR output waveforms.

5.2.3 Control PC Output Integrated Waveforms

The waveforms which are intercepted by the GR-CPCI-XC4V development board, must be processed by the LEON3 processors and should not be retransmitted to the control PC. The integrated waveforms are obtained via coherent integration of NUM_COHERENT waveforms and incoherent integration of NUM_UNCOHERENT waveforms. The coherent/incoherent algorithm we will describe in the Section 5.3. As soon as the LEON3 processors finish the integration of the waveforms, the GR-CPCI-XC4V board must send the results to the control PC by the integrated waveform structure as shown in Fig.5.6.

The integrated waveforms are a stream of concatenated logs of 300 bytes, each of them associated to one integrated waveform. There are 44 bytes used for the header

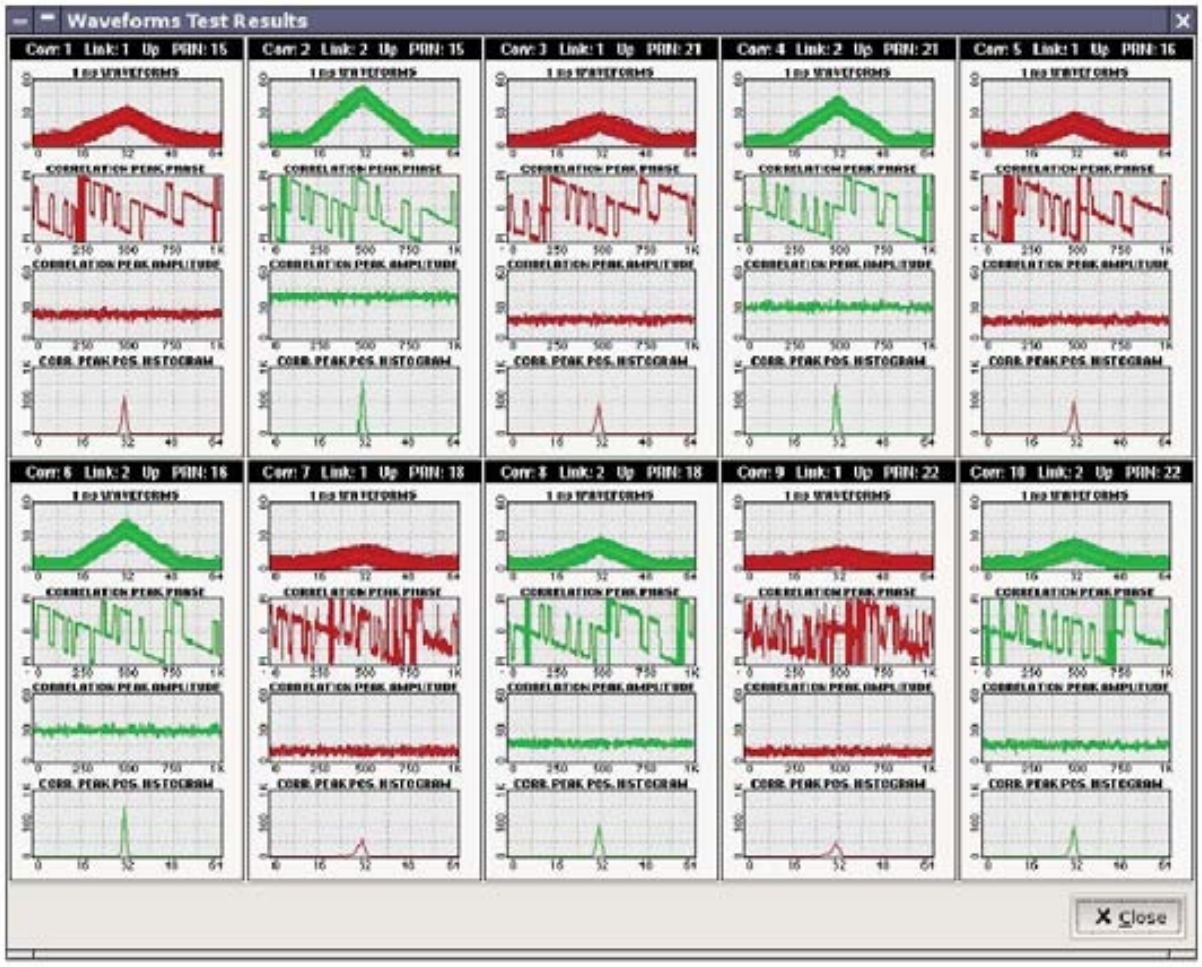


Figure 5.5: The graphical representation of the GOLD-RTR output waveforms.

and the remaining 256 bytes used for the 64 values of the integrated waveform (4 bytes each).

5.3 Post-Processing Algorithms and Timing Parameters

This section outlines the post-processing algorithm to be implemented in the HTPCP board. This section is written by Toni Rius, mainly introduces the three logical steps implemented by the post-processing algorithm: data reduction, coherent integration and incoherent integration. And how to extract the timing parameters: 1) Coherence time; 2) Coherence Integration Time; 3) Incoherence Integration Time.

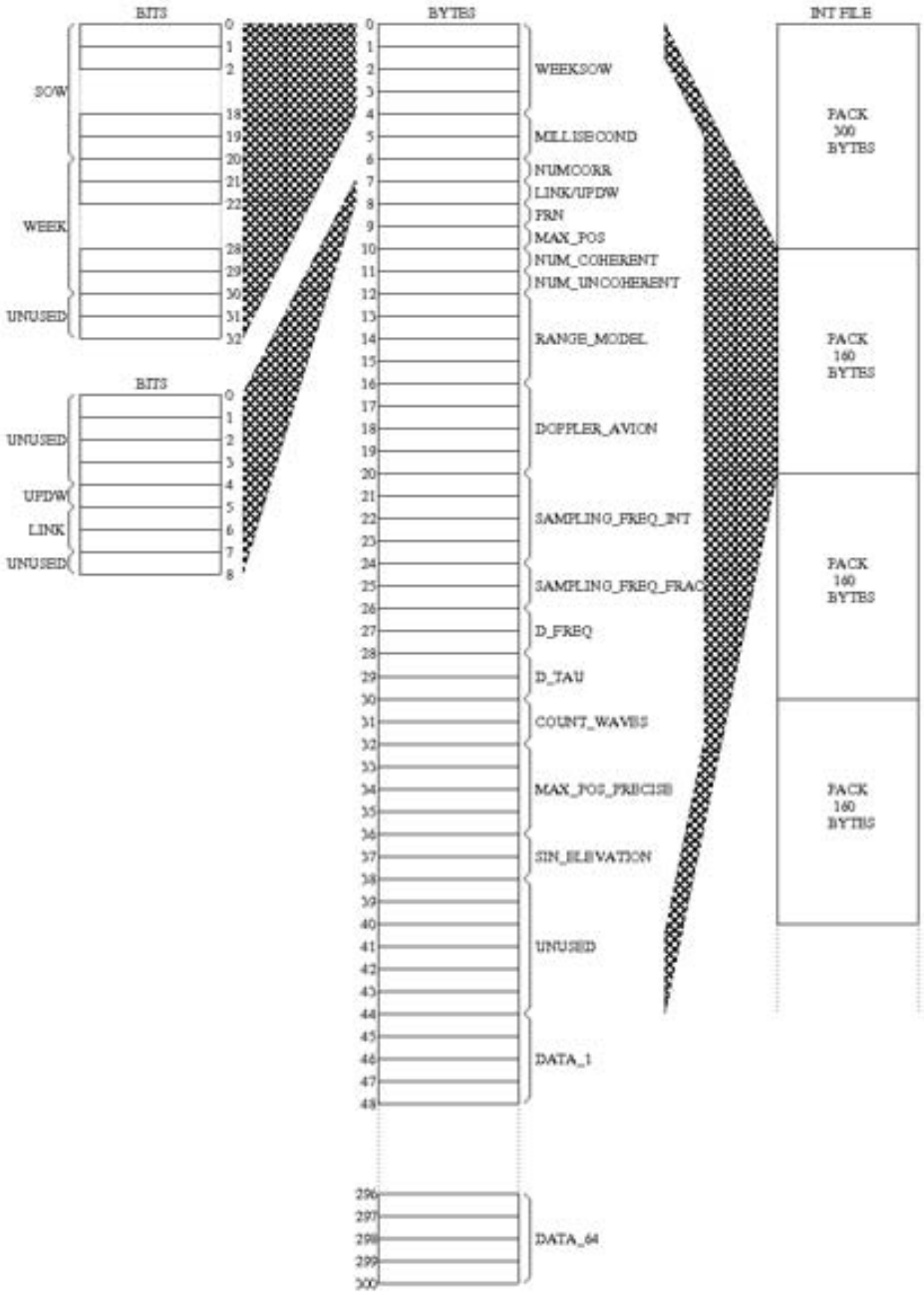


Figure 5.6: The structure of the integrated waveforms packets (300 Bytes).

5.3.1 Data Reduction

Let's use an example to fix the idea of the data reduction. The output data of GOLD-RTR during T seconds can be presented as a set of complex numbers:

$$\omega_{lag,msec,channel}. \tag{5.1}$$

where lag is the correlator lag number $[0:63]$; $msec$ is the tag corresponding to

the clock tick after the first epoch $[0 : 1000 \cdot T - 1]$; $channel$ is the tag corresponding to channel number $[0 : 9]$.

The first value of $msec$ corresponds to a 1-Pulse-Per-Second (PPS) mark. Therefore the $msec$ value could be associated unambiguously to the instrument clock. Consequently, the size of the array $\omega_{lag,msec,channel}$ is $64,0000 \cdot T$. We can define the coherent intervals in msec (divided by 0.001) as shown below:

$$N = \frac{T}{0.001}, \tag{5.2}$$

There are another two parameters (N_{coh} and N_{incoh}) are used to control the coherent integration time T_{coh} and the incoherent integration time T_{inc}. We assume that $T_{inc} \geqq T_{coh}$ and $T_{inc} \leqslant 1$ second, where T, T_{coh} and T_{inc} are defined in second. The definitions of coherent integration intervals N_{coh} and incoherent integration interval N_{incoh} are shown below:

$$N_{coh} = \frac{T}{T_{coh}}, \tag{5.3}$$

$$N_{incoh} = \frac{T}{T_{inc}}, \tag{5.4}$$

For each correlation channel, assume that $T = 60$ sec, $T_{coh} = 0.002$ sec and $T_{inc} = 0.100$ sec, the integration algorithm should reduce the 1-msec complex waveforms from $N = 60,000$ to $N_{coh} = 30,000$ by coherently integration, and reduce the integrated waveforms from $N_{coh} = 30,000$ to $N_{incoh} = 600$ by incoherently integration. Therefore the compression factor is 0.01.

5.3.2 Coherent Integration

In general, the post-processing algorithm can be applied with different functions, depending on the purpose of remote sensing. It is expected that our deployed system can reduce the amount of data by coherent/incoherent integration functions.

In addition to the 1-msec complex waveforms, the GOLD-RTR also generate two flags for each correlation channel $b^1_{msec,channel}$ and $b^2_{msec,channel}$. Flag $b^1_{msec,channel}$ indicates if the observed satellite for this correlation channel occur a possible transition of the navigation bit during a certain msecs. The flag is set to **false**, if there is a possible transition, thus the complex waveform is considered to be non-valid. Otherwise the flag is set to **true**, and the complex waveform will be integrated in the coherent integration summations. The other flag $b^2_{msec,channel}$ indicates the correctness of the valid 1-msec waveform that is generated by the GOLD-RTR. The flag is set to **true**, if the valid 1-msec waveform has been generated by the GOLD-RTR without errors. Otherwise the flag is set to **false**, in which case the corresponding waveform is ignored in the coherent integration summations.

Based on the values of the flag $b^1_{msec,channel}$, there are four possible cases:

1. No navigation bit transition within the second.
2. A navigation bit transition in the first millisecond.
3. A navigation bit transition in the last millisecond.
4. A navigation bit transition not in the first or in the last millisecond.

The first three cases are treated equally. Assume that we want to process 1 second's waveforms, and do the coherent integration every 10 msecs (0,1,2,3,4,5,6,7,8,9). There is only one navigation bit could happen during this period, since the receiver knows the position of the satellite with a small error ($<$300km/1msec). If the navigation bit dose not happen, or it happens in msec 0 or in msec 9, we can just ignore such msec, and do the coherent integration with the rest of msecs. The integrated power of these waveforms can be computed after the coherent integration time as shown below:

$$\Omega_{lag,i,channel} = |\sum \omega_{lag,k,channel}|^2, \tag{5.5}$$

where the summation includes all the valid complex waveforms with time stamps k, within the coherent integration time interval $[(i-1)\cdot T_{coh} : i\cdot T_{coh} - 0.001]$.

In the fourth case, the navigation bit could happens at any other msecs, then we need to decide if the navigation bit has changed or not. We assume that the navigation bit happens at the position n, the valid waveforms are obtained in such interval can be divided in two subsets:

- 0,1,2,3... n-1;
- n+1 .... 9.

The navigation bit could have two values: +1 or -1. Therefore, the sum of these two sets waveforms could have the positive sign or the negative sign as shown in Fig.5.7. At the transition time, they could have the same value or the different value of the summation as shown below:

1. if navigation bit is equal to +1,
   $\Omega^{+}_{lag,i,channel} = |\sum \omega_{lag,k,channel} + \sum \omega_{lag,l,channel}|^2$,
2. if navigation bit is equal to -1,
   $\Omega^{-}_{lag,i,channel} = |\sum \omega_{lag,k,channel} - \sum \omega_{lag,l,channel}|^2$.

With these two separate sets, the integrated power of the waveforms can be computed as the following two cases:

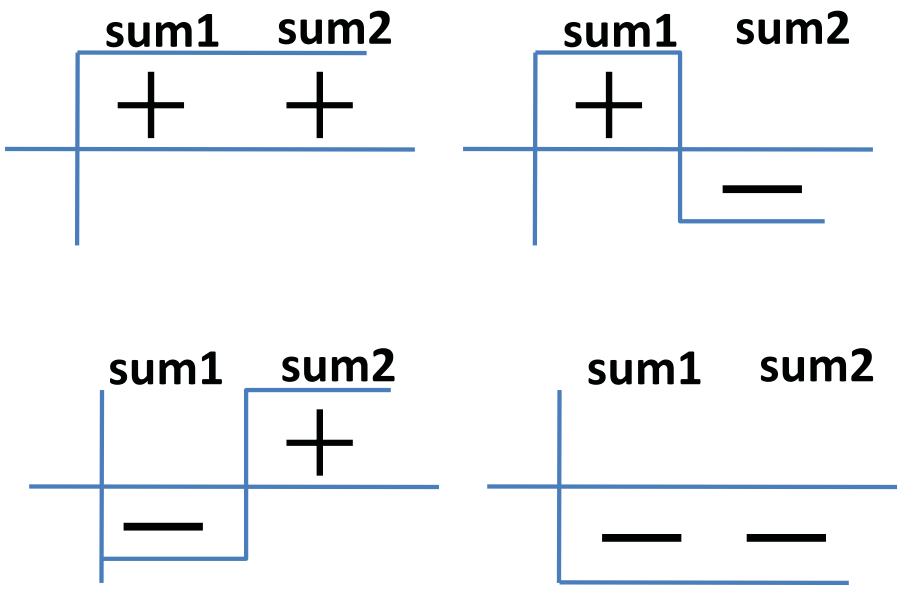


Figure 5.7: Four cases occur with the sum of these two sets waveforms during the transit time.

If $\Omega^{+}_{lag,i,channel} > \Omega^{-}_{lag,i,channel}$,

we set

$\Omega_{lag,i,channel} = \Omega^{+}_{lag,i,channel}$.

Otherwise, we set

$\Omega_{lag,i,channel} = \Omega^{-}_{lag,i,channel}$.

5.3.3 Incoherent Integration

With the incoherent correlation interval $j = 0, \ldots, N_{incoh} - 1$ that has been addressed in the previous section, we can estimate the average power of the coherent integration waveforms in the interval j as shown below:

$$\bar{\Theta_{lag,j,channel}} = \frac{1}{N_{val,j}} \sum \Omega_{lag,i,channel}, \tag{5.6}$$

where the summation extends to the $N_{val,j}$ valid values of $\Omega_{lag,i,channel}$ within the incoherent integration interval $[(j-1) \cdot T_i : j \cdot T_j - T_{coh}]$.

5.3.4 The Coherence Time τ_{coh} and the Coherence Integration Time T_{coh}

The coherence time τ_{coh} is determined by the campaign surrounding. The coherence integration time T_{coh} is determined by the campaign instrument.

Here we use two cases study to illustrate their differences as shown in Table 5.I. In case one, we use Nikon camera to take photos with different objectives in the same time duration (1 s). Since the objectives are different, we use different speeds of shutter to get the clear images. In case two, we use GOLD-RTR to capture the reflected signals from various surfaces. Based on the different types of surface, we can take 1 ms, 2 ms or 5 ms as coherent integration time in order to get the useful information. Based on the different ranges of the surface, we can take the same time duration (10 ms) as the coherent time, in order to obtain the information of a certain range of the reflection plane.

Table 5.I: Case study to illustrate the τ_{coh} and T_{coh}.

Instrument	NIKON Camera			GOLD-RTR		
Coherent integration time	1/1000s	1/500s	1/200s	1ms	2ms	5ms
Objective	football game	person	city view	fluctuated Sea	lake	creek
Coherent time	1 s	1 s	1 s	10 ms	10 ms	10 ms.

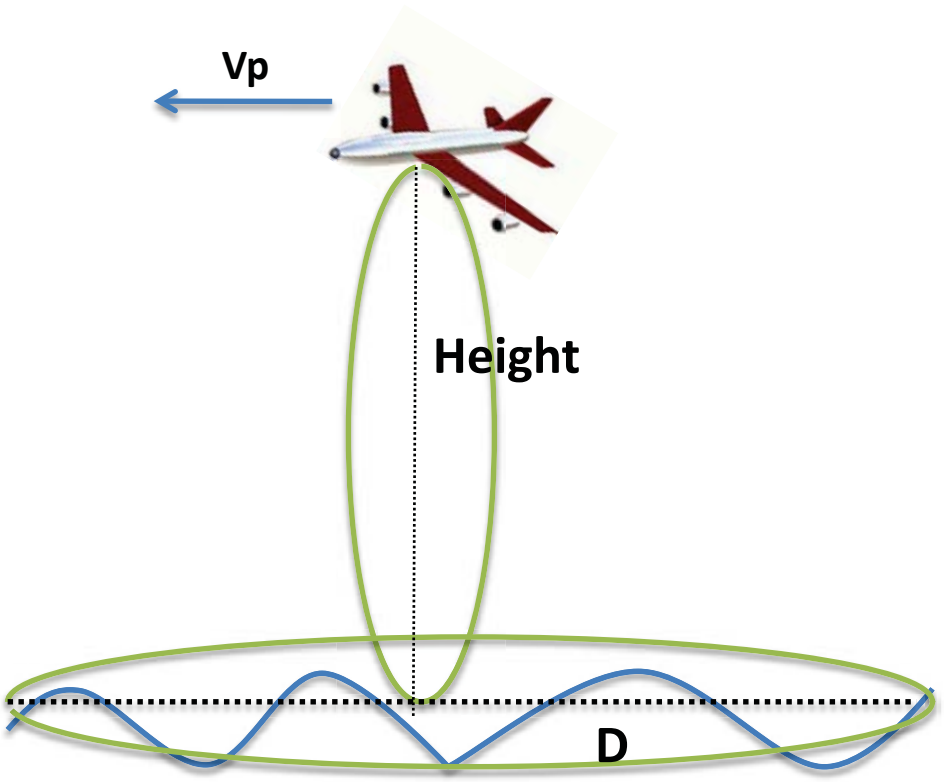


Figure 5.8: Coherence time τ_{coh} and coherence integration time T_{coh}.

From these two case study, we learn that the coherence time τ_{coh} is defined as the time interval in which the signal could be tracked. In the GNSS-R application, this coherent time is very large for the direct signal. However, for the reflected signal in a random surface (i.e. the sea surface), this coherent time mainly depend on the following two factors:

- the temporal scale of the sea surface variability, and
- the velocity of the receiver parallel to the sea surface.

As shown in Fig.5.8, if the sea surface is changing fast, the coherence integration time T_{coh} will be short, because the phase of the reflected signal will be unpredictable at short scales. Similarly, if the receiver V_p moves very fast, the collected data will be reflected at different patches of the sea surface with the random phase variations. Consequently, we need to integrate the signal (i.e: as complex numbers) coherently during a short coherent integration interval T_{coh} on the order of the coherence time τ_{coh}. There are two relevant questions that related to the processing strategy (see [1]):

- How large should be the coherent integration time T_{coh} of a complex waveform?
- Which sampling rate should be used to independently measure the complex waveforms?

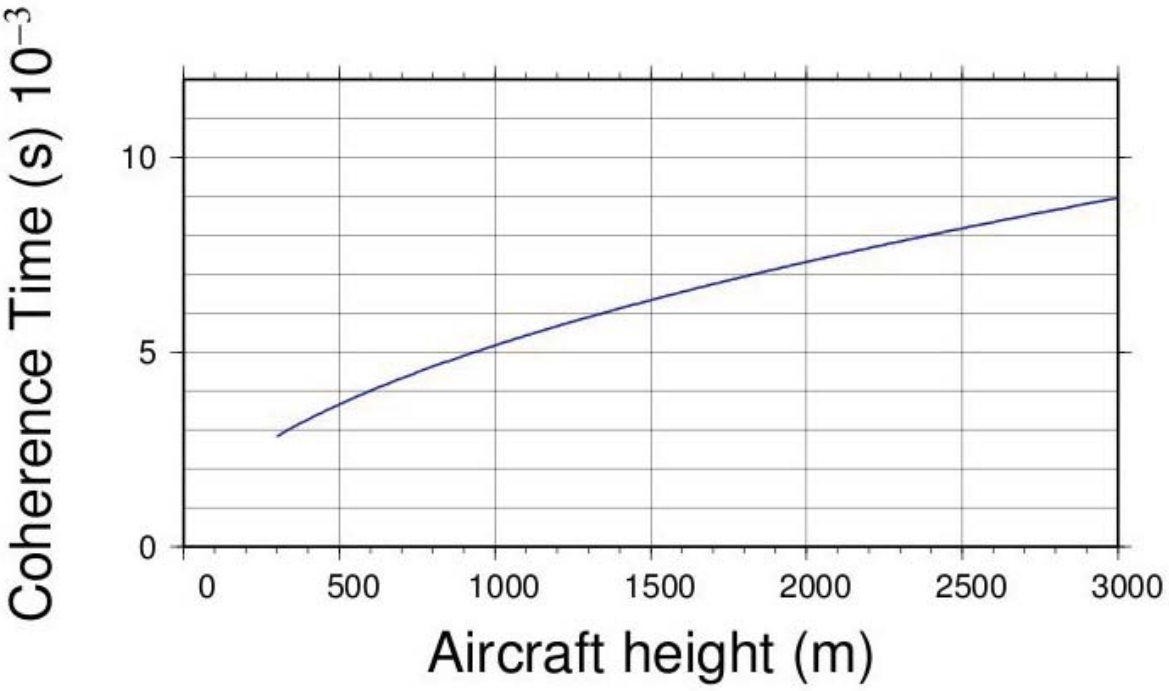


Figure 5.9: Define the coherent time by the Van Citter-Zernike theorem.

The answer of both questions is in terms of a single parameter: the coherence time τ_{coh}. Before defining the concept precisely, we present a recipe based on the van Citter-Zernike theorem, to compute an order of the magnitude of τ_{coh} (see [2]).

$$\tau_{coh} = \frac{\lambda_{L1}}{2v_p}\sqrt{\frac{R_r}{2c\tau cos\theta}} \tag{5.7}$$

where v_p is the perpendicular velocity, R_r is the height of the receiver, c is the speed of light, τ is the correlator delay and θ is the incidence angle.

This equation explains the main sources of the variability of τ_{coh}. In particular, it introduces its dependency on the the correlator delay τ, and the incidence angle θ. Using such recipe, we have plotted the coherence integration time τ_{coh} as a function of the aircraft height as shown in Fig.5.9. Assuming a horizontal velocity $v_p = 75m/s$, at the correlator delay $\tau = 30m$, and at an incidence angle $\theta = 0$. We can see that the airplane is flying at heights around 3000 meters during the coherence time $\tau_{coh} = 10ms$.

5.3.5 Incoherence Integration Time T_{incoh}

After each coherent integration time T_{coh}, we compute the power of waveforms. To reduce the noise-like contributions to the power waveforms, we integrated the further during an incoherence integration time T_{incoh}. Assume that the number of waveforms N need to be integrated incoherently, there is a function of the track resolution ΔL and the coherence time τ_{coh} trough the approximate equation:

$$N = \Delta L/(v_p \cdot T_{coh}) \tag{5.8}$$

Where we assume that the coherence time τ_{coh} is equal to the coherent integration interval T_{coh}, and the horizontal velocity of the spacecraft is v_p. The incoherence integration time T_{incoh} needed to collect these N independent power waveforms as shown below:

$$T_{incoh} = N \cdot \ T_{coh} \tag{5.9}$$

5.4 Demonstration Architecture

The original idea is that the GOLD-RTR sends data to the Control PC directly. However, with the increasing complexity of the calculation and the storage issue of the received data, we design a black box named HTPCP as the intermediary to solve the processing and transmission issues. As shown in Fig.5.10, this demonstration aims at displaying the following tasks by two MicroBlaze and four LEON3 processors:

- ETH0 captures all network traffic and bypass it to ETH1 using the BRAM memory.
- ETH1 sends data to the Control PC.
- MicroBlaze0 bypasses the waveform packets to 4 LEON3 processors using RAM0 memory, meanwhile LEON3 processors send a synchronization signal to MicroBlaze0 via GPIO0.

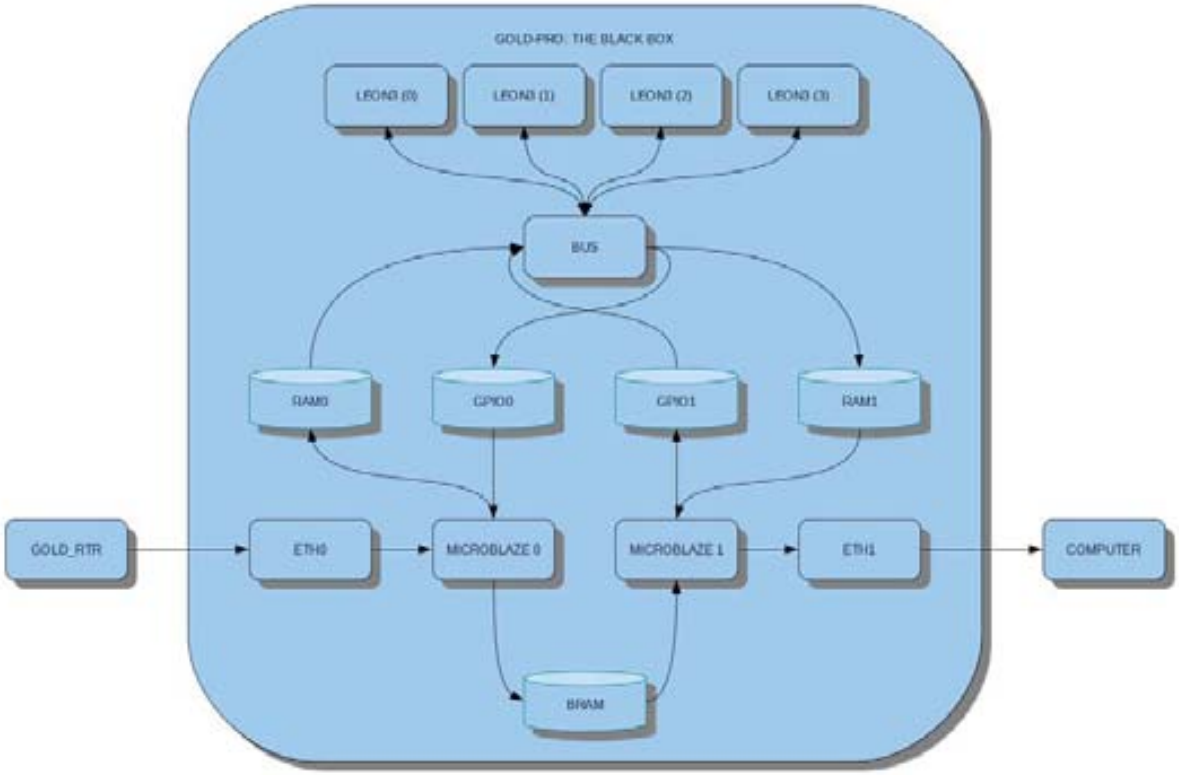


Figure 5.10: HTPCP demonstration diagram.

- Four LEON3 processors send the calculated results to MicroBlaze1 using RAM1 memory, meanwhile MicroBlaze1 processor send a synchronization signal to LEON3 processors via GPIO1.

In the following subsections, we firstly introduce the overall design concept of HTPCP as a black box, and then briefly introduce the tasks of each processor.

5.4.1 The Black Box - HTPCP

The main objective of this system is to detect the waveform packets and do the post-processing calculations to reduce the huge amount of data. There are mainly two operations: packets transmission and packets processing.

The system can actuate as a black box, which can capture all traffics between GOLD-RTR and Control PC as a sniffer, and bypass the packets to the real destination transparently. The reality ethernet transmission exists the packets of ICMP or ARP protocols [*ARP protocol*], which must be transmitted from the Control PC to the GOLD-RTR and viceversa. This requirement forces us to implement a bypass to route all the received packets from the ETH1 to the ETH0 respectively. The trick is to configure the MAC addresses [*MAC address*] by the values of the real devices (GOLD-RTR and Control PC), and bypass all the ICMP, ARP and other control packets. The packets

transmission is controlled by two Microblaze processors. Therefore, the system works transparently without modifying the software routines in the actual devices. With this declaration, we design the following software routines for packets transmission:

- The software routine must configure the Ethernet devices (ETH0 and ETH1) to emulate the actual devices (Control PC and GOLD-RTR):
  - The ETH0 must be configured with the MAC address of the Control PC.
  - The ETH1 must be configured with the MAC address of the GOLD-RTR.
- When the Ethernet devices are configured, we can begin the operational mode: bypass all packets and intercept the detected streams.
  - If the packet is not a waveform packet, it must be sent to the real destination.
  - If the packet is a waveform packet, it must be stored to the RAM0, and it will not be sent to the Control PC.

Table 5.II: The amount of data comparison between original system and complete system.

GOLD-RTR + Control PC			GOLD-RTR + Black box + Control PC		
160 B	1 correlator	millisecond	300 B	1 correlator	100 millisecond
1600B	10 correlator	1 millisecond	3000 B	10 correlator	100 millisecond
1600000B = ˜1.53 MB	10 correlator	each second	30000 B = ˜29.29 KB	10 correlator	each second
5760000000 B = = ˜5.36 GB	10 correlator	each hour	108000000 B = ˜102.99 MB	10 correlator	each hour
138240000000 B = ˜128.75 GB	10 correlator	each day	2592000000 bytes = ˜2.41 GB	10 correlator	each day

The packets processing is controlled by four LEON3 processors that allow more calculations complexity and reduce the amount of data. Assume that all the corerlators are working together, the original system (GOLD-RTR + Control PC) can generates 128.75 GB data per day as shown in Table5.II. However, the demonstration system (GOLD-RTR + Black box + Control PC) can reduce the amount of data by the integration time (Coherent integration time * Incoherent integration time). The reduction ratio of the data can be calculated directly by:

- Reduction Ratio = (Total amount data / Integration time) * (300 / 160)
- Reduction Ratio = (Total amount data/ (Coherent integration time * Incoherent integration time)) * 1.875

For example, if we use the coherent integration time of 10 msec and the incoherent integration time of 10 msec, therefore, the integration time of each correlator is 100 msec. The system will generate 2.41 GB data per day. The reduction ratio is shown below:

- Reduction Ratio = Total amount * 0.01875

- 1.875% of needed space
- 98.125% of reduced data

As we know, the maximum values of integration time is 1 second (1000 msec). Therefore, if we use the coherent integration time of 20 msec and the incoherent integration time of 50 msec, we can get the maximum integration time of each correlator is 1000 msec. The reduction ratio is shown below:

- Total amount * 0.001875
- 0.1875% of needed space
- 99.8125% of reduced data

5.4.2 MicroBlaze Processors

The system implements two MicroBlaze processors (MicroBlaze0 and MicroBlaze1) that control the communication between the ETH0 and ETH1, and other components in the system.

- The MicroBlaze0 has the follow two tasks:
  1. Receive the packets from GOLD-RTR by the ETH0 device:
     - If the received packet contains waveforms, then write them to the RAM0.
     - If the received packet dose not contains waveforms, then writes them to the BRAM memory.
  2. Read packets from the BRAM and sends them to GOLD-RTR by ETH0.
     - If the MAC address is not set, configure the ETH0 MAC address by the first packet sent to the GOLD-RTR.
- The MicroBlaze1 has the follow three tasks:
  1. Receive the packets from the Control PC by the ETH1 device, and writes them to the BRAM.
  2. Read the packets from the BRAM and sends them to the Control PC by the ETH1.
     - If the MAC address is not set, configures the ETH1 MAC address by the first packet sent to the Control PC.
  3. Read the integrated waveforms from the RAM1 and sends them to the Control PC by the ETH1.

5.4.3 LEON3 Processors

The HTPCP implements four LEON3 processors that calculate the complex waveforms for each correlators in the system. In our experiment, we only use four LEON3 processors to calculate the waveforms of four correlators. In the future, we can add up to 10 LEON3 processors to calculate the waveforms of ten correlators. The LEON3 processors have the follow eight tasks:

1. Initialize the processor ID using interrupts
2. Initialize the memory structures
3. Receive data from the RAM0
4. Send synchronism to the MicroBlaze0 using the GPIO0
5. Integrate the waveforms until the integration finish
6. Write the integrated waveforms to RAM1
7. Wait the synchronism from the MicroBlaze1 using the GPIO1
8. Restart the memory structures

The main problem of programming these four LEON3 processors is that all processors execute the same C code. It needs an identifier to know which portion of the memory is used to read and write. This problem was fixed by an initialization function of an interrupt controller that allow us to identify each processor with an unique ID.

5.5 Campaign on Real Data

This experiment has been applied to the waveforms that is obtained from real GPS signals reflected on the sea surface. This campaign was supported by ESA contract ECN 20069/06/NL/EL and the Spanish National Space Plan ESP2005-03310. O. Nogués- Correig, S. Rib´ó, J. Torrobella, J. Sanz (ICE-IEEC/CSIC) made the operation of the GOLD-RTR instrument possible. The campaign is taken during ESA airborne campaigns to support the L-band radiometric mission, SMOS, and in collaboration with the Helsinki University of Technology (TKK), the Technical University of Denmark (TUD), and the French Institute for the Exploitation of the Sea, (IFREMER). A detailed description of the campaign can be found in Cardellach et al. [3].

The waveforms are collected with the GOLD-RTR instrument [4], that is designed and manufactured by the IEEC. The instrument provides complex waveforms every millisecond. This dedicated hardware receiver contains 640 complex correlators organized into 10 channels of 64-lags each, computing 1 complex Delay Map (DM) per channel simultaneously. The system can be programmed to correlate signals

from 10 different satellites at each correlation channel to obtain DMs in different geometries. Or it can be programmed to process the same signal in different channels under different frequency shifts, thus obtaining the Delay-Doppler Map (DDM).

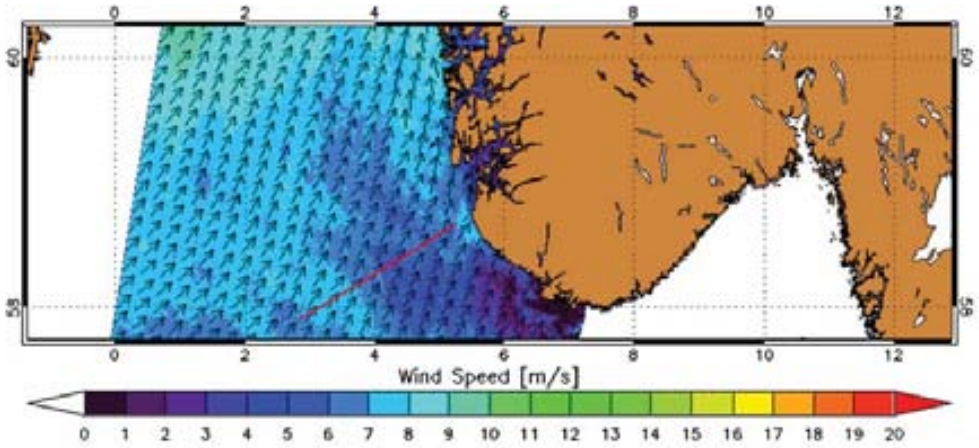


Fig. 13. Wind velocity field from analysis of the ENVISAT SAR observations taken simultaneously to the April 15 2006 flight, 36 min after the series presented in Figs. 11 and 12. The values of the wind direction, interpolated along the trajectory, result in an average up-wind direction of 212° azimuth, wind velocity pointing to 32°, in agreement with the up-/down-wind signatures in Fig. 12. SAR data copyright ESA 2006, processing by BOOST Technologies following Monaldo et al. (2004).

Figure 5.11: The map of the experiment campaign CO10.(Adapted from Cardellach et al. [3])

The GOLD-RTR equipment was mounted on the Skyvan aircraft and crossed 100 km of the Norwegian Coast (longitude between 3° and 5°, latitude between 57.5° and 58.5°, see Fig.5.11) at an altitude of 3000 m and speed of 75 m/s, during 12 flights in April 2006. The figure was generated using Generic Mapping Tools version 4 by Wessel and Smith (1998).

The experiments presented in this study have been performed on the waveforms integration for 1 s, aims at obtaining the incoherent sum of 100 waveforms, each coherently integrated for 10 ms. To do the experiment, the follow components are needed:

- Computer that runs the GOLD-RTR simulator.
- Computer that runs the GOLD-RTR controller and receiver.
- Experiment data provided from a real experiment campaign (CO10).
- The GR-CPCI-XC4V development board with its mezzanine board GR-CPCI-2ETH-SRAM-8M that runs all 3 programs (MicroBlaze0, MicroBlaze1 and LEON3).
- Xilinx parallel JTAG programming cable IV that is used to load the programming files (*leon3mp.bit*) into the PROM/SDRAM of the GR-CPCI-XC4V development board. Also it supports to debug with the GRMON.
- Two Ethernet patch cables with the RJ45 micron connectors is to realize the LAN connection between GOLD-RTR and HTPCP, also the LAN connection between HTPCP and Control PC, in order to realize the high speed data transmission.

- RS232 to RS232 cable is to realize the command line communication between PC and FPGA board, The windows Hyper terminal is used to see the commands by UART.

5.6 Experiment Results

To demonstrate the correct operation of the entire system (GOLD-RTR + HTPCP + Control PC), we conduct the following two experiments:

1. Using the original system to verify the system accuracy. (GOLD-RTR emulator + the Control PC)

   The first experiment allows us to check the accuracy of the system by sending, receiving and storing all the waveforms. It determines the accuracy of the test environment and the quality percent of the waveforms that we have obtained from the GOLD-RTR emulator and the receiver computer. To do this, we run 100 complete experiments on the GOLD-RTR emulator with the input data of the campaign CO10, and send all output data directly to the Control PC (without the development board). Table 5.III shows the results of the 10 executions, Table5.IV shows the minimum lost ratio, Table5.V shows the maximum lost ratio, Table5.VII shows the averages lost ratio.

Table 5.III: The ten execution results for experiment one.

ID	Files	Size	Waveforms	Seconds	Lost wav.	Lost size	Ratio	Lost ratio
0000	49	10199172800	63744830	6449	397560	63609600	99.38%	0.62%
0001	49	10199661440	63747884	6449	394506	63120960	99.38%	0.62%
0002	49	10196793440	63729959	6449	412431	65988960	99.36%	0.64%
0003	49	10181816480	63636353	6449	506037	80965920	99.21%	0.79%
0004	49	10179097280	63619358	6449	523032	83685120	99.18%	0.82%
0005	49	10201467200	63759170	6449	383220	61315200	99.4%	0.6%
0006	49	10196092320	63725577	6449	416813	66690080	99.35%	0.65%
0007	49	10153285760	63458036	6449	684354	109496640	98.93%	1.07%
0008	49	10201892000	63761825	6449	380565	60890400	99.41%	0.59%
0009	48	10059387520	62871172	6449	1271218	203394880	98.02%	1.98%

Table 5.IV: The minimum lost ratio of experiment one.

ID	Files	Size	Waveforms	Seconds	Lost wav.	Lost size	Ratio	Lost ratio
0008	49	10201892000	63761825	6449	380565	60890400	99.41%	0.59%

With the first experiment, we conclude that the test environment is very acceptable, because the most important parameter (lost ratio) is between 0.59% and 1.98% and the average is 0.84%. This deviation is produced by the storage of the amount of data produced by the GOLD-RTR (12.21 Mbits/second). The

Table 5.V: The maximum lost ratio of experiment one.

ID	Files	Size	Waveforms	Seconds	Lost wav.	Lost size	Ratio	Lost ratio
0009	48	10059387520	62871172	6449	1271218	203394880	98.02%	1.98%

Table 5.VI: The averages lost ratio of experiment one.

Files	Size	Waveforms	Seconds	Lost wav.	Lost size	Ratio	Lost ratio
48.9	10176866624	63605416.4	6449	536973.6	85915776	99.16%	0.84%

expected result of the first experiment is that the Control PC lost some waveforms caused by the huge amount of data.

2. Using the complete system to verify the system improvement. (GOLD-RTR emulator + HTPCP + Control PC).

   The second experiment is to use the complete system to verify the system improvement. This experiment allows us to integrate the waveforms that is generated by the GOLD-RTR in Campaign CO10, for more information see [3]. In order to reduce the high amount of data, we use 10 ms as coherent integration time and 10 ms as incoherent integration time to implement the coherent/incoherent integration algorithm on HTPCP board. After several tests, we can capture the same results in the control PC as shown below:

Table 5.VII: The results of experiment two.

Captured files	49
Captured size	10262782400 bytes
Captured waveforms	64142390
Captured seconds	6449 (1h 47' 29")
Lost waveforms	347610
Lost size	55617600 bytes
Capture ratio	~99.46%
Lost ratio	~.54%

This execution result is as expected, the HTPCP produces the same amount of data 234Kbits/second without the dataloss. The results of the this experiment demonstrated that the control PC captures 100% of the integrated waveforms and the complete system runs as expected.

5.7 Summary

In this Chapter, firstly we present the experiment constrains of the GNSS-R post-processing application and introduce the functions of the GOLD-RTR instrument, GR-CPCI-XC4V board and the control PC and the structure of the generated waveforms.

Secondly, we introduce the three logical steps to implement the post-processing

algorithm, data reduction, coherent integration and incoherent integration. We have introduced the calculation of the ratio of data reduction, which can be determined by the waveforms formats (normal waveform format / integrated waveform format, 300 bytes / 160 bytes) and the integration time (Coherent integration time * Incoherent integration time).

Thanks to the feature of HTPCP design, it allow us to interconnect the development board between GOLD-RTR and the Control PC without any additional changes, the HTPCP operates as a black box ready to work. Regarding to the processors implemented in HTPCP: MicroBlaze and LEON3, we have demonstrated that they can work correctly. They communicate by the shared memory and GPIO devices to send and receive messages by predefining the data structures. These data structures define the synchronization method which we have used to communicate all different elements that compound the entire system. The concurrence performance can be performed by the multiple LEON3 processors that they can access to the same shared memory zone by different semaphores.

To demonstrate the correct operation of the entire system (GOLD-RTR + HTPCP + Control PC), we conduct two experiments. In the first example experiment, the original system loses data all times caused by the saturation of the receiver computer (0.84% as AVG). In the second experiment, we demonstrates how HTPCP can reduce the data and maintain the integrity of the results through a transparent operational mechanism. Thanks to the coherent and incoherent integration algorithms, the waveforms can be used with the same accuracy as the original system for more scientific purposes.

As the conclusion, we must say that the most important work is the data structures and semaphore definition, due to the different processors platform (MicroBlaze and LEON3). Since all the system need to use these data structures and semaphores to do the tasks, and it is necessary to have a correct definition in the software layer in order to accomplish and guarantee the final objectives of the development: transmit the waveform and reduce the science data by the post-processing algorithm. We can see that the use of the complete system allows the data reduction and solves the problem caused by the data mass storage.

Bibliography

[1] H. You, J. Garrison, G. Heckler, and Smaljlovic, "The autocorrelation of waveforms generated from ocean-scattered gps signals," *IEEE Geosci. Remote. Sens. Lett.*, vol. 3, pp. 76 – 80, 2006.

[2] C. Zuffada, T. Elfouhaily, and S. Lowe, "Sensitivity analysis of wind vector measurements from ocean reflected gps signals," *Remote Sens. Environm.*, vol. 88, no. 3, pp. 341 – 350, 2003.

[3] E. Cardellach and A. Rius, "A new technique to sense non-gaussian features of the sea surface from l-band bi-static gnss reflecitons," *Journal of Remote Sensing of Environment*, vol. 112, no. 6, pp. 2927–2937, 2008.

[4] O. Nogués-Correig, E. Cardellach-Galí, J. Sanz-Camderrós, and A. Rius, "A gps-reflections receiver that computes doppler-delay maps in real time," *IEEE Transactions on Geoscience and Remote sensing*, vol. 45, no. 1, pp. 156–174, 2007.

Web Reference

[*GNSS-R*] http://www.ice.csic.es/research/gold_rtr_mining/gnssr.php

[*Sniffer*] http://en.wikipedia.org/wiki/Packet_analyzer

[*Experiments*] http://www.ice.csic.es/research/gold_rtr_mining/campaigns.php

[*MAC address*] http://en.wikipedia.org/wiki/Mac_address

[*IP address*] http://en.wikipedia.org/wiki/IP_address

[*IP protocol*] http://en.wikipedia.org/wiki/Internet_Protocol

[*UDP protocol*] http://en.wikipedia.org/wiki/User_Datagram_Protocol

[*ICMP protocol*] http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol

[*ARP protocol*] http://en.wikipedia.org/wiki/Address_Resolution_Protocol

Part VI

Overall Conclusions

Overall Conclusions

First of all, the work presented in this document clearly shows that the goal of the GNSS-R post-processing system is to efficiently compute the collect waveforms by GOLD-RTR. All the relevant details have been explained for the real-time computation over ten independent correlation channels. Moreover, the method for reaching this goal is probably as important if not more than the result itself. The relevance of this method is due to its novelty in combining the computation power of SMP and the transmission power of NOC, to create a novel HTPCP architecture in order to realize the various post-processing algorithms in real-time.

The initial approach was focused on the conventional multi-task parallel system design (SMP), during an in-depth study into the parallel workload and its mathematical modeling. However it goes beyond this to study the broader scope of different levels design (the configurable processors and their development tools, memory hierarchy design etc.), it reveals how the pipeline stall affects the system throughput. The final and most important step was to obtain a detailed simulation result by MPARM emulator that would make this framework an implementable approach in real systems. Comparison of the values of the lag to the affection of different components in the subsystem, memory subsystem and the separate bus design could be the keypoint for this timing degrading in the parallel system. Ignoring the relative cost of the system, we compared the execution time with the increasing parallel application. We found that the standard deviation and speedup ratio can not reach our timing requirement. In order to solve the synchronization and cache migration issues at the system level, we need to speculate at the hardware level by changing the memory hierarchy and communication system in the hardware design.

In the end, an HTPCP architecture was obtained that employs a post-processing system as the fundamental platform for implementing various post-processing algorithms. Furthermore, two problems of HTPCP are proposed and resolved, 1) the parallelization of the inherent serial output of the GOLD-RTR instrument by Transmission Elements (TEs) and 2) the post-processing of the multi-channel (I and Q) correlators in parallel by Processing Elements (PEs). We focus on the hardware design of HTPCP, exploit the designs of TEs and the design of PEs. In order to guarantee real-time transmission, various interface designs (MPI and FSL) have been studied, in order to integrate two processor systems (LEON3 and Microblaze). The theoretical algorithm was initially developed for a generic post-processing algorithm, in order to characters and minimize the memory requirement in terms of space.

We operate the same post-processing algorithm on the HTPCP board and verify the correctness of each IP core designs in the HW design by seven SW routines.

Finally, a case study for the post-processing algorithm shows the ratio of data reduction, which can be determined by the waveforms formats (normal waveform format / integrated waveform format, 300 bytes / 160 bytes) and the integration time (Coherent integration time * Incoherent integration time). The compression factor is 0.01.

A feature of the HTPCP design allows us to interconnect the development board with the GOLD-RTR and the control computer without any additional changes; the HTPCP operating as a black box which is ready to work. Regarding the processors which we have used in HTPCP, MicroBlaze and LEON3, we have demonstrated that they can work correctly. They communicate by the shared memory and GPIO devices and send and receive messages by predefining the data structures. These structures define the synchronization method which we have used to communicate all different elements that form the entire system. Using the multiple LEON3 processors, the concurrence performance can access the same shared memory zone by different semaphores.

To demonstrate the correct operation of the entire system (GOLD-RTR + HTPCP + Control PC), we conduct two experiments. In the first, the original system loses data at all times due to the saturation of the receiver computer (0.84% as AVG). The second experiment demonstrates how HTPCP can reduce the data and maintain the integrity of the results through a transparent operational mechanism. Thanks to the coherent and incoherent integration algorithms, the waveforms can be used with the same accuracy as the original system for more scientific purposes.

As a conclusion, we must say that the most important thing is the data structures and semaphore definition, due to the different processor platforms (MicroBlaze and LEON3). Since the whole system needs to use these data structures and semaphores to carry out tasks, it is necessary to have a correct definition in the software layer in order to accomplish and guarantee the final objectives of the development: transmit the waveform and reduce the scientific data by means of the post-processing algorithm. We can see that the use of the complete system allows for data reduction and solves the problem caused by data mass storage.

Appendix A

Input Waveform Format

Waveform Input Names	Address	Format	Size(B)	Value(Hex)	Value(DEC)
WeekSow	*p + 0	Int	4	15a86524	363357476
millisecond	*p + 4	Short	2	0000	0
status_NumberCorrelator	*p + 6	Char	1	01	1
link_updw	*p + 7	Char	1	50	80
prn	*p + 8	Char	1	13	19
max_pos	*p + 9	Char	1	17	23
amplitude	*p + 10	Char	1	08	8
phase	*p + 11	Char	1	2E	46
range_model	*p + 12	Int	4	00001635	5685
Doppler_avion	*p + 16	int	4	00013414	78868
sampling_freq_int	*p + 20	Int	4	02625a00	40000000
sampling_freq_frac	*p + 24	Short	2	0000	0
d_freq	*p + 26	Short	2	0000	0
d_tao	*p + 28	Short	2	b4	180
sin_elevation	*p + 30	Short	2	Fffff7b2	4294965170
data[128]	*p + 32	Char	1	..	..

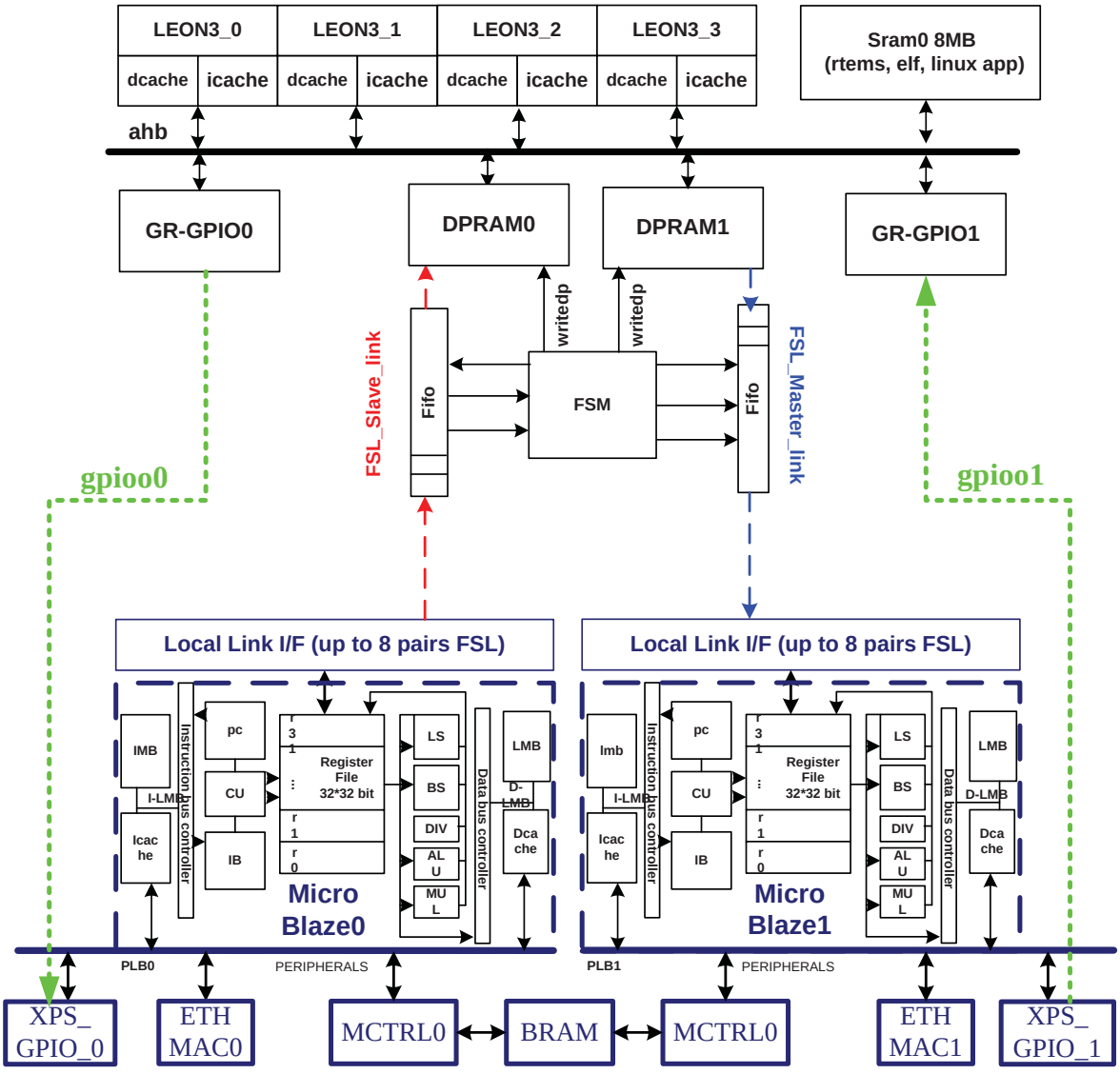
Appendix B

Output Waveform Format

Waveform Output Names	Address	Format	Size(B)	Value(Hex)	Value(DEC)
WeekSow	*w +365+ 0	Int	4	15a86524	363357476
millisecond	*w +365+ 4	Short	2	01f4	500
NumberCorrelator	*w+365 + 6	Char	1	01	1
link_updw	*w +365+ 7	Char	1	50	80
prn	*w +365+ 8	Char	1	13	19
num_coherent	*w +365+ 9	Int	4	01	1
num_uncoherent	*w+365 + 13	Int	4	000003e8	1000
range_model	*w+365 + 17	Int	4	00001635	5685
doppler_avion	*w+365 + 21	Int	4	00013414	78868
sampling_freq_int	*w+365 + 25	Int	4	02625a00	40000000
d_freq	*w+365 + 29	Short	2	0000	0
d_tao	*w+365 + 31	Short	2	00B4	180
Complete	*w+365 + 33	Char	1	0000	0
Polarization	*w+365 + 34	Char	1	58	88
data[64]	*w+365 + 35	Float	4	..	random

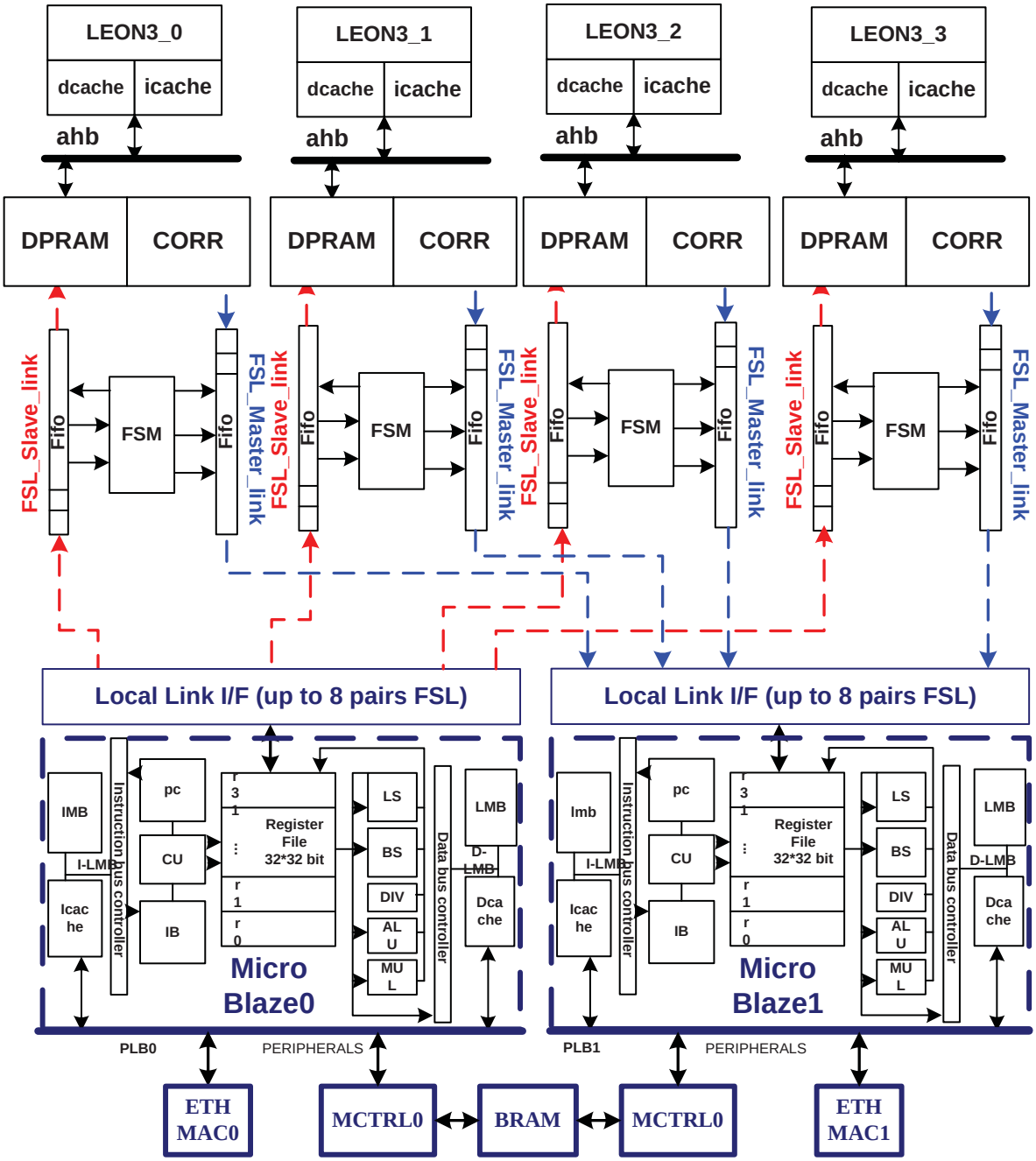
Appendix C

Solution 1: Four LEON3 processors share one software routine.


LEON3_0
LEON3_1
LEON3_2
LEON3_3
dcache
icache
Sram0 8MB
(rtems, elf, linux app)
ahb
GR-GPIO0
DPRAM0
DPRAM1
GR-GPIO1
writedp
writedp
FSL_Slave_link
Fifo
FSM
Fifo
FSL_Master_link
gpioo0
gpioo1
Local Link I/F (up to 8 pairs FSL)
Local Link I/F (up to 8 pairs FSL)
IMB
I-LMB
Icac he
Instruction bus controller
pc
CU
IB
Register File 32*32 bit
LS
BS
DIV
AL U
MU L
LMB
D-LMB
Dca che
Data bus controller
Micro Blaze0
Micro Blaze1
PLB0
PLB1
PERIPHERALS
XPS_ GPIO_0
ETH MAC0
MCTRL0
BRAM
MCTRL0
ETH MAC1
XPS_ GPIO_1

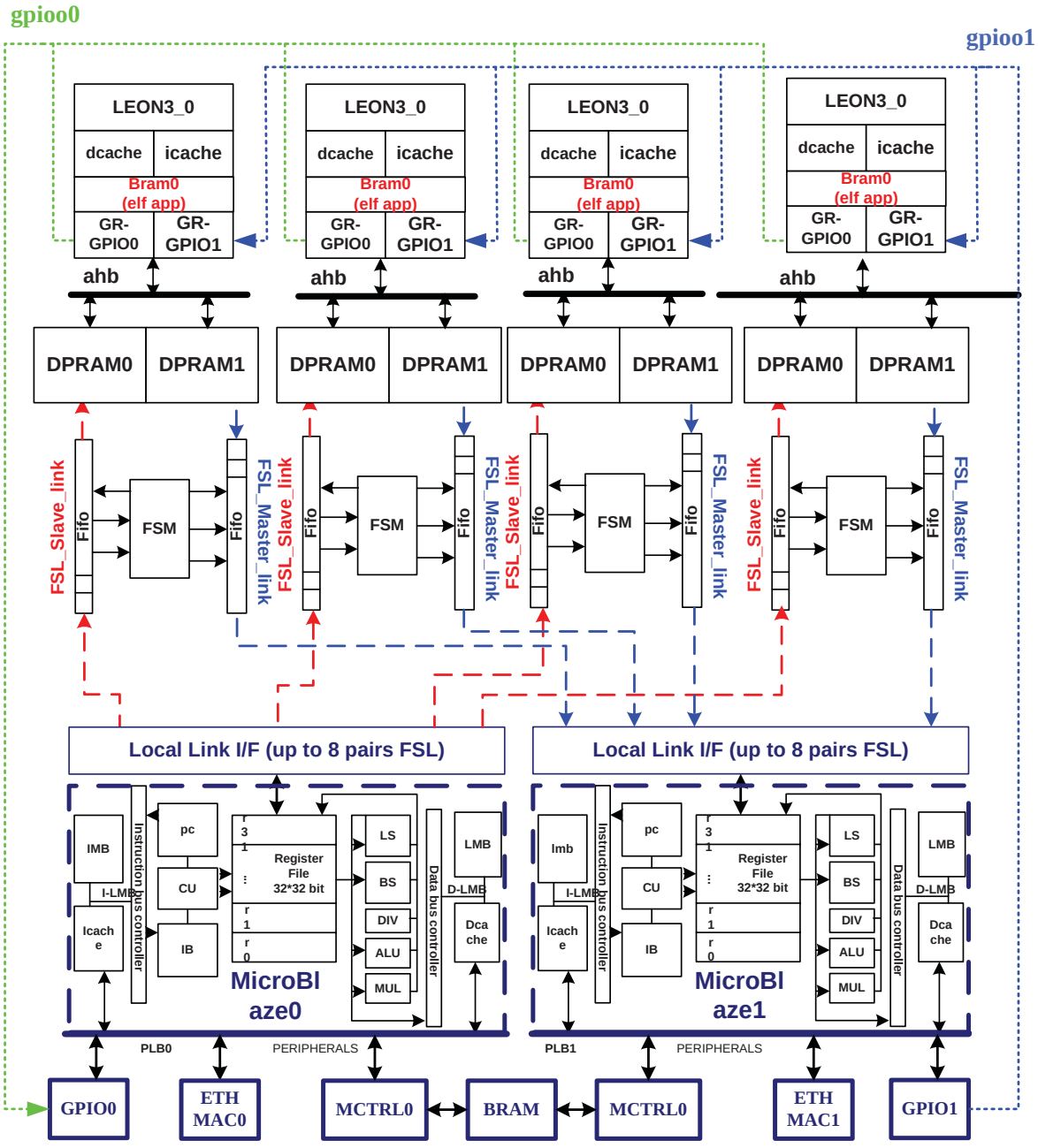
Appendix D

Solution 2: Four LEON3 cores work with four hardware routines.


LEON3_0
dcache
icache
ahb
DPRAM
CORR
LEON3_1
dcache
icache
ahb
DPRAM
CORR
LEON3_2
dcache
icache
ahb
DPRAM
CORR
LEON3_3
dcache
icache
ahb
DPRAM
CORR
FSL_Slave_link
Fifo
FSM
Fifo
FSL_Master_link
Local Link I/F (up to 8 pairs FSL)
Local Link I/F (up to 8 pairs FSL)
IMB
I-LMB
Icac he
Instruction bus controller
pc
CU
IB
r 31
Register File 32*32 bit
r 1
r 0
LS
BS
DIV
AL U
MU L
Data bus controller
LMB
D-LMB
Dca che
Micro Blaze0
Micro Blaze1
PLB0
PERIPHERALS
PLB1
PERIPHERALS
ETH MAC0
MCTRL0
BRAM
MCTRL0
ETH MAC1

Appendix E

Solution 3: Four LEON3 processors execute four software routines in parallel.


gpioo0
gpioo1
LEON3_0
dcache
icache
Bram0
(elf app)
GR-GPIO0
GR-GPIO1
ahb
DPRAM0
DPRAM1
FSL_Slave_link
FSL_Master_link
Fifo
FSM
Local Link I/F (up to 8 pairs FSL)
IMB
I-LMB
Icache
Instruction bus controller
pc
CU
IB
Register File 32*32 bit
LS
BS
DIV
ALU
MUL
LMB
D-LMB
Dcache
Data bus controller
MicroBlaze0
MicroBlaze1
PLB0
PLB1
PERIPHERALS
GPIO0
ETH MAC0
MCTRL0
BRAM
MCTRL0
ETH MAC1
GPIO1

Appendix F

Commands from Control PC to HTPCP

Data	Lengths	Description
‘O‘	1	Check communication (It is capital letter ‘O‘, not number zero.)
‘Q‘	1	Quit
‘B‘	1	Start
‘E‘	1	Stop acquiring
‘C‘+ data	<1500	Configuration/Control Data

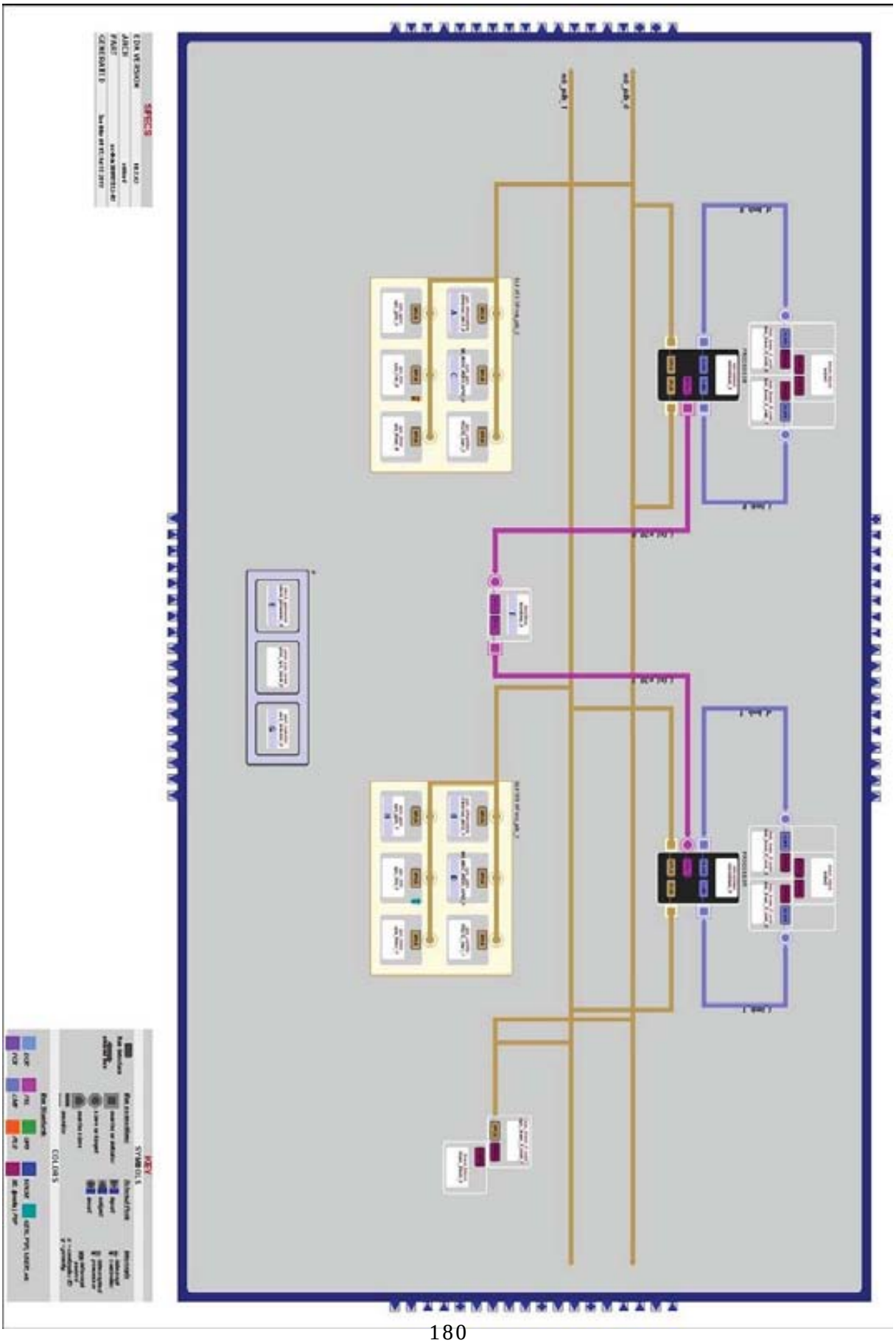
Appendix G

Commands from GOLD-RTR to HTPCP

Data	Lengths	Description
‘H‘	1	Check communication
‘A‘+ bytes	14	Reserved
‘F‘+data(64)	66	Flags
‘Q‘	1	Close application
‘O‘	1	Check communication OK
‘W‘ + data(800)	802	5 Waveforms in 5 blocs of 160 Bytes
’R’ + data(1240)	1242	RINEX

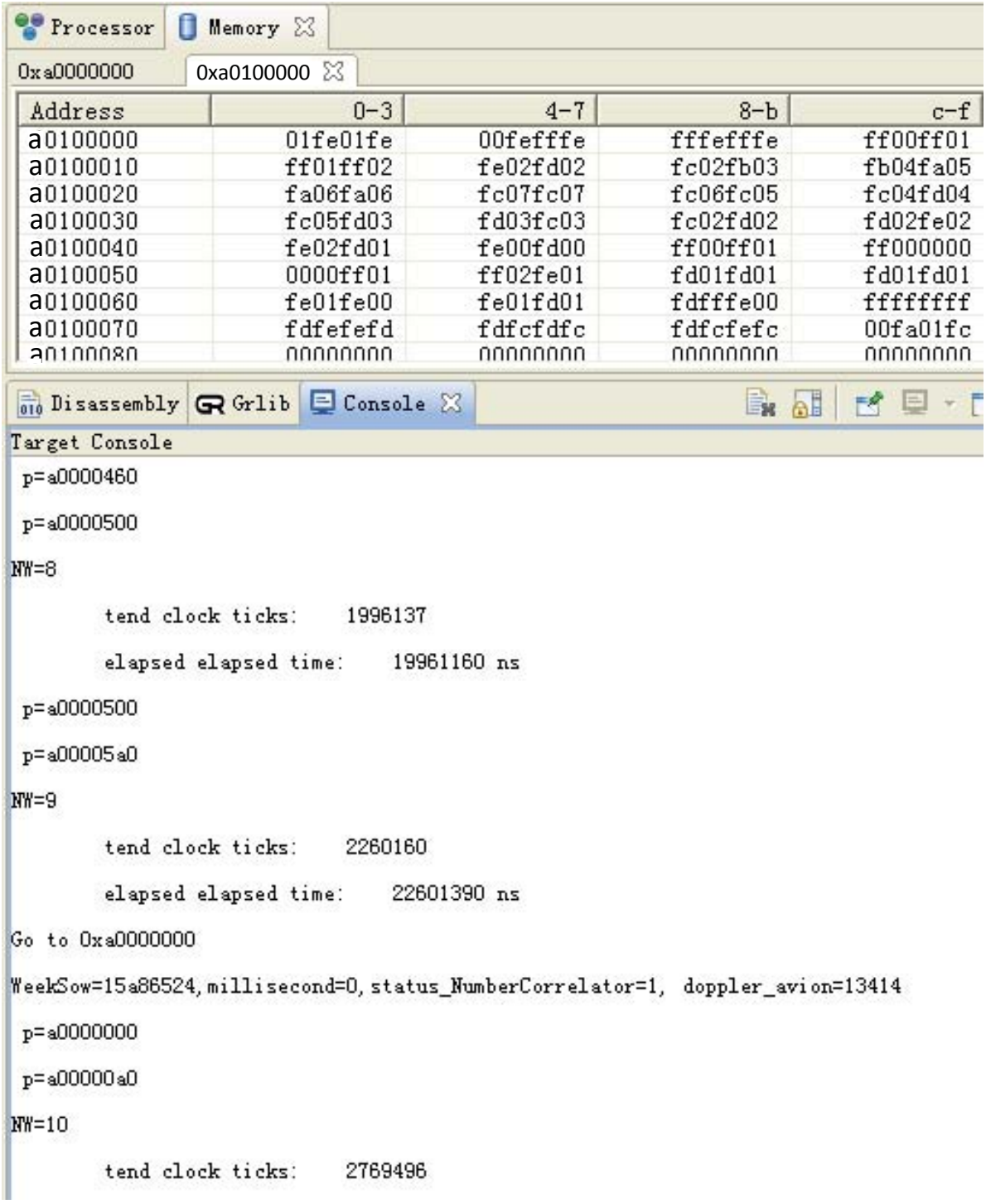
Appendix H

Block Diagram of HTPCP in EDK



Appendix I

The Execution Time “printf“ in GRMON


Processor
Memory
0xa0000000
0xa0100000
Address 0-3 4-7 8-b c-f
a0100000 01fe01fe 00fefffe fffefffe ff00ff01
a0100010 ff01ff02 fe02fd02 fc02fb03 fb04fa05
a0100020 fa06fa06 fc07fc07 fc06fc05 fc04fd04
a0100030 fc05fd03 fd03fc03 fc02fd02 fd02fe02
a0100040 fe02fd01 fe00fd00 ff00ff01 ff000000
a0100050 0000ff01 ff02fe01 fd01fd01 fd01fd01
a0100060 fe01fe00 fe01fd01 fdfffe00 ffffffff
a0100070 fdfefefd fdfcfdfc fdfcfefc 00fa01fc
Disassembly
Grlib
Console
Target Console
p=a0000460
p=a0000500
NW=8
tend clock ticks: 1996137
elapsed elapsed time: 19961160 ns
p=a0000500
p=a00005a0
NW=9
tend clock ticks: 2260160
elapsed elapsed time: 22601390 ns
Go to 0xa0000000
WeekSow=15a86524, millisecond=0, status_NumberCorrelator=1, doppler_avion=13414
p=a0000000
p=a00000a0
NW=10
tend clock ticks: 2769496