Performance and Area Evaluations of Processor-based Benchmarks on FPGA Devices

by

Jiunn-Tyng, Kao

A Doctoral Thesis submitted in partial fulfilment of the requirement for Master of Philosophy of Loughborough University

January 2014

Abstract

The computing system on SoCs is being long-term research since the FPGA technology has emerged due to its personality of re-programmable fabric, reconfigurable computing, and fast development time to market. During the last decade, uni-processor in a SoC is no longer to deal with the high growing market for complex applications such as Mobile Phones' audio and video encoding, image and network processing. Due to the number of transistors on a silicon wafer is increasing, the recent FPGAs or embedded systems are advancing toward multi-processor-based design to meet tremendous performance and benefit this kind of systems are possible. Therefore, is an upcoming age of the MPSoC. In addition, most of the embedded processors are soft-cores, because they are flexible and reconfigurable for specific software functions and easy to build homogenous multi-processor systems for parallel programming. Moreover, behavioural synthesis tools are becoming a lot more powerful and enable to create datapath of logic units from high-level algorithms such as C to HDL and available for partitioning a HW/SW concurrent methodology.

A range of embedded processors is able to implement on a FPGA-based prototyping to integrate the CPUs on a programmable device. This research is, firstly represent different types of computer architectures in modern embedded processors that are followed in different type of software applications (eg. Multi-threading Operations or Complex Functions) on FPGA-based SoCs; and secondly investigate their capability by executing a wide-range of multimedia software codes (Integer-algometric only) in different models of the processor-systems (uni-processor or multi-processor or Co-design), and finally compare those results in terms of the benchmarks and resource utilizations within FPGAs. All the examined programs were written in standard C and executed in a variety numbers of soft-core processors or hardware units to obtain the execution times. However, the number of processors and their customizable configuration or hardware datapath being generated are limited by a target FPGA resource, and designers need to understand the FPGA-based tradeoffs that have been considered - Speed versus Area.

For this experimental purpose, I defined benchmarks into DLP / HLS catalogues, which are "data" and "function" intensive respectively. The programs of DLP will be executed in LEON3 MP and LE1 CMP multi-processor systems and the programs of HLS in the LegUp Co-design system on target FPGAs. In preliminary, the performance of the soft-core processors will be examined by executing all the benchmarks. The whole story of this thesis work centres on the issue of the execute times or the speed-up and area breakdown on FPGA devices in terms of different programs.

Keywords:

Soft-core, Multi-core, Co-design, Electronic-system-level, High-level-synthsis, Thread-level-parallelism, Data-level-parallelism, Symmetric-multi-processor, RISC, VLIW, LEON3

Acknowledgements

This research was supported by the research group of Electronic Systems Design, School of Electronic and Electrical Engineering, Loughborough University, UK. I first would like to thank my supervisor and this research leader, Dr Vassilios Chouliaras. Secondly, I would like to thank my lab colleagues; David Stevens for the rearrange of Lempel-Ziv-Welch compression C code. He is a master of C programming language and Linux Ubuntu operating system and always opens for an answer; and Mark Milward for simulations of the LE1 CMP benchmarks. He is an expert in FPGA and SoC design. His help and advice have greatly helped my experimental analysis and philosophy.

TABLE OF CONTENTS

TABLE OF ABBREVIATIONS

Abbreviation	Expansion
ADD	Addition
ADPCM	Adaptive Differential Pulse Code Modulation
AES	Advanced Encryption Standard
AHB	Advanced High-performance Bus
ALAP	As Late As Possible
ALM	Adaptive Logic Module
ALU	Arithmetic Logic Unit
ALUT	Adaptive LUT
AMBA	Advanced Microcontroller Bus Architecture
ANSI	American National Standards Institute
APB	Advanced Peripheral Bus
ASAP	As Soon As Possible
ASIC	Application Specific Integrated Circuit
BCC	Bare-C Cross-compiler system
BITOP	Bitwise Operation
c-step	control step
CAD	Computer Aided Design
CLB	Configurable Logic Block
CM	Convolution Matrix
CM60x60	Convolution Matrix with 60x60 output data
CMOS	Complementary Metal Oxide Semiconductor
CMP	Chip Multi Processor
CPU	Central Processing Unit
CPUID	CPU Identification
D-cache	Data Cache
D-RAM	Data RAM
DCT	Discrete Cosine Transform
DDR SDRAM	Double Data Rate SDRAM
DE2	Development and Education 2
DE4	Development and Education 4

DFADD	Double Precision Floating-Point Addition
DFDIV	Double-precision Floating-Point Division
DFG	Data Flow Graph
DFMUL	Double-precision Floating-Point Multiplication
DFSIN	Double-Precision Floating-Point Numbers of Sine function
DFT	Discrete Fourier Transform
DIV	Divider
DLL	Delay Locked Loop
DLP	Data Level Parallelism
DSE	Design Space Exploration
DSP	Digital Signal Processing
DSU	Debug Support Unit
EDA	Electronic Design Automation
ESL	Electronic System Level
FF	Flip-Flop
FFT	Fast Fourier Transform
FIFO	First In First Out
FIR	Finite Impulse Response
Fmax	Maximum Frequency
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
FSL	Fast Simplex Link
FU	Functional Unit
GB	Gigabyte
GCC	GNU Compiler Collection
GDS	Global Positioning System
GPR	General Purpose Register
GPS	Global Positioning System
GRLIB	Gaisler Research Library
GSM	Global System for Mobile
GUI	Graphical User Interface
HDL	Hardware-Description-Language
HI	Multiply and Divide Register Higher Result
HLS	Electronic System Level
HP	Hewlett Packard

HW	Hardware
I-cache	Instruction Cache
I-RAM	Instruction RAM
IALU	Integer-ALU
IC	Integrated Circuit
IFE	Instruction Fetch Engine
ILP	Instruction Level Parallelism
IMULT	Integer-MULT
IP	Intellectual Property
IR	Integer Register
IRQ	Interrupt Controller Register
IU	Integer Unit
JAL	Jump and Link
JPEG	Joint Photographic Experts Group
JTAG	Joint Test Action Group
KB	Kilobyte
LAB	Logic Array Block
LAT	Latency
LC	Logic Cell
LE	Logic Element
LLVM	Low Level Virtual Machine
LMB	Local Memory Bus
LO	Multiply and Divide Register Lower Result
LP	Linear Program
LPC	Linear Predictive Coding
LRR	Least Recently Replaced
LRU	Least Recently Used
LSU	Load Store Unit
LUT	Look Up Table
LZW	Lempel Ziv Welch
LZW45K	Lempel Ziv Welch with size of 45KB input data
MAC	Media Access Control
MB	Megabyte
MHz	Megahertz
MIG	Memory Interface Generator

MIMD	Multiple Instruction Multiple Data
MIPS	Microprocessor without Interlocked Pipeline Stages
MISD	Multiple Instruction Single Data
MMU	Memory Management Unit
MP	Multi Processor
MP3	MPEG-1 Audio Layer-3
MPEG	Moving Picture Experts Group
MPSoC	Multi-processor System on Chip
ms	millisecond
MUL	Multiplier
MUX	Multiplexer
NISC	No Instruction Set Computer
NOP	No Operation
NRE	Non Recurring Engineering
ns	nanosecond
OPB	On-chip Peripheral Bus
PAR	Place and Route
PB	Petabyte
PC	Program Counter
PDA	Personal Digital Assistant
PDF	Portable Document Format
PIC	Peripheral Interface Controller
PLB	Processor Local Bus
PLL	Phase Locked Loop
PROC	Processor
ps	picosecond
RAM	Random Access Memory
RAMB	RAM Block
RC	Resource Constraint
REG	Register
REM	Reminder
RISC	Reduced Instruction Set Computing
RTL	Register Transfer Level
SDC	System of Difference Constraint
SDRAM	Synchronous Dynamic RAM

SHA	Secure Hash Algorithm
SHARC	Super Harvard Architecture Single Chip Computer
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SMP	Symmetric Multi-Processor
SoC	System on a Chip
SOPC	System on a Programmable Chip
SPARC	Scalable Processor Architecture
SPMD	Single Program Multiple Data
SRA	Square Root Approximate
SRAM	Static Random Access Memory
SUB	Subtraction
SVT	Single Vector Trapping
SW	Software
TB	Terabyte
TC	Timing Constraint
TLP	Thread Level Parallelism
UART	Universal Asynchronous Receiver/Transmitter
UMA	Uniform Memory Access
VCO	Voltage Controlled Oscillator
VEX	VLIW Example
VHDL	VHSIC (Very-High-Speed-IC) HDL
VLIW	Very Long Instruction Word
VLSI	Very Large Scale Integration
XML	Extensible Markup Language
µC	Microcontroller
µP	Microprocessor
µs	microsecond
µs	microsecond

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

1.1 Motivation

In historical computing, the computer processor was based on vacuum tubes and started the design of stored-program (Memory-Control-FUs) to form the fundamental of computer architectures known as von Neumann (1947) [1]. The vacuum tube was replaced by much smaller and reliable transistor in the 1950s. By the 1960s, electronic processing elements began to appear based on semiconductor devices as such many transistors in a chip to form an Integrated-Circuit (IC) and thus led to the generation of the microprocessor [2]. In accordance with Moore's Law, the number of transistors on the same chip area nearly doubles every two years [3]. Following this trend for half a century, the silicon chips have become inexpensively and a lot more complicated designing and very elated computing problems can ever be solved. Over the decades, semiconductor devices are able to accommodate the whole system (CPUs, system-bus, peripherals and internal memory) to form an embedded system, known as *System-on-a-Chip (SoC).* SoCs are widely used in portable devices such as PDAs, digital cameras, GPS units, MP3 players, and smart phones; larger systems such as air traffic control and missile systems [4]. *Embedded processors* are CPUs typically used in an embedded system. These processors are usually smaller silicon space compared, sustain lower performance, and need much less power compared to the server or desktop CPUs. Smart phones are the perfect example of an embedded system. The embedded processor is typically as an Application-Specific-Integrated-Circuit (ASIC) hard-core which built into a "floorplan" of a fixed IC or a soft-core that described by the *Hardware-Description-Languages (HDLs)* for a customized configuration then synthesize into a programmable fabric logic such as *Field-Programmable-Gate-Array (FPGA)* devices. During the last decade, multi-processor systems became available in computers and in *Multi-Processor System-on-Chip (MPSoC)* devices to achieve higher computing performance; and they continue to grow in further on the market [5]. At the present time, electronic products have become much more competitive and the product life cycle much shorter [6]; therefore engineers attempt to search a cheaper, easier and faster way to develop new chips. Reusable designs such as *Intellectual-Property (IP)* cores are instrumental when designing a new chip. While the post-fabrication of the chips has been greatly improved, the numbers of the transistors that can be fitted in the same area on a die chip are increased. In addition, the overall energy consumption of the chips has been reduced due to the dynamic power decrease when transistor becomes smaller and leakage current problem has also been solved by Triple-oxide approach [7, 8]. However, Moore's Law will not be sustained forever as physical limitations cause problems such as heat dissipation. Moreover, it is expensive to design and fabricate a much more powerful processor on a single chip. Other factors such as, the

clock rates of embedded CPUs are increased slowly up to Gigahertz and now parallelism in multiple cores will be via for prospect development in the future. Thence, research trends have been moved on "*Multi-core Processor*" rather than the single-core architecture. Each of the cores executes its own instruction stream concurrently to speed up the program execution for parallelism computing, such as the "Thread-Level-Parallelism (TLP)". By using a large number of presenting manufactured processors to gain a better performance is a valid design avenue. FPGAs contain tens of thousands reprogrammable logic elements are this kind of device that possible to implement such MPSoC systems on a programmable chip. Another method is the use of the "Electronic-System-Level (ESL)". This design flow typically composes a complex part such as a mixture of Hardware-and-Software (HW/SW) synergism which is widely utilized in the modern SoC systems [9]. As field-programmable logic turned out and gate-level interconnection of the hardware circuits may configure by HDLs, software programs are able to generate in the hardware circuit. Meanwhile, the compiler-based language of RTL algorithm synthesis has greatly improved and become matured during the last decade, C programs has been enabled to convert to HDLs to generate hardware units easily. To follow these perspective, electronic designers can implement software in high-level objectives and shift HW/SW boundaries to make the tradeoffs to achieve desired performance [10]. However, a versatile system-level toolchain is indispensable for the "*HW/SW Co-design*", such as partitioning theme needs to be deceived in which part of the system is better to be implemented in hardware or software piece and each component maybe represents in different description language to its corresponding portion [11].

1.2 Aims and Objectives of this Research

This thesis effort evaluates the performance and area of embedded processors on modern developed FPGA boards. The embedded "processor-systems" within the FPGA are able to various architectural patterns so as to satisfy distinct digital processing requirements, which are classified into three levels of single-core, multi-core processor and Co-design that corresponds to different computing methodologies. The objectives of the following illustrations and experiments are to present the efficient resolutions for divergent computational methodologies in embedded processors on FPGA devices.

First of all, before considering any speed-up implementations, there is always at least one general-purpose processor in the system, hence the ***Single-core Processor*** performance ought to be reviewed. The evaluated uni-processor for the following adhering experiments is LEON3, LE1, and Tiger-MIPS soft-cores respectively. To do this, all adopted benchmark suits are being executed in those processors and synthesized on FPGAs to compare their performance and area. Secondly, for ***Multi-core Processor*** designs, I introduce Symmetric Multi-Processor (SMP) systems of LEON3 Multi-Processor (MP) and LE1 Chip-Multi-Processor (CMP), which "parallel computing" codes can be applied and improved by a number of these processors. In the other hand, it also can be implemented in a HW/SW platform by assigning a function to accelerate in the hardware. Thirdly, "function complexity" codes such as CHStone (a collection of programs see page 56 and 87) are being profiled and executed in the LegUp ***HW/SW Co-design*** system. Finally, to compare and conclude those two benchmarks manipulations (Data and Function levels) that performs in different processor-systems (Multi-core and Co-

design) in terms of their execution times (*Speed*) and FPGA logic blocks cost (*Area*). Thus, lead to the art of the *Speed&Area Tradeoffs.*

The LEON3 and LE1 soft-cores are chosen because they are multi-processor available to be connected and open source; the LEON3 SoC platform provides a rich reusable IP library for designs [12]; and LE1 is a new development soft processor in our research department and highly paramertisable architectural [13]. LegUp framework is chosen for the Co-design (with a Tiger-MIPS soft-core [14]) because the tool is new development of C to HDLs and also open source codes; and gives a large amount of American-National-Standards-Institute (ANSI) C programs. For the software applications of this thesis, I concern only about ***Integer Arithmetic*** programs. For mapping onto this software on the processors, the examined benchmarks are arranged for data and function intensive programs. Convolution-Matrix (CM) and Lempel-Ziv-Welch (LZW) are widely used in image format; both of them are "data-intensive" algorithms in digital processing. CHStone are selected from wide-range applications in region of arithmetic, media processing, security and microprocessor [15]; they are "function-intensive" algorithms in digital processing. For Speed&Area tradeoffs burden, I formulate a proper solution of automated Design-Space-Exploration (DSE) for the FPGA-based MPSoC design and verify the subject.

1.3 Objectives of the Research and Goals

The objectives of the necessary research to achieve the stated aims are as follows:

- **Define the types of parallel computing**
  This describes different types of parallelism (Flynn's Taxonomy) for multi-core processor systems.

- **Overview of the ESL design**
  This describes Behavioural / ESL design process that leads to HW/SW flows for Co-design systems.

- **Overview of FPGA silicon devices**
  This describes the fundamental FPGA structures and states the advantage / disadvantage comparing to ASIC devices. Moreover, make reliable resource predictions on FPGAs.

- **Overview of the embedded processors on FPGAs**
  State the popular used of hard/soft processors and presented their specifications (eg. architectures, bus / FUs support, and LUTs utilization) including of available FPGA boards to be targeted at different vendors.

- **Define the different embedded processor-systems on FPGAs**
  There are Homogeneous: multi-core architecture of all general-purpose processors, Co-design: a general-purpose processor with a hardware accelerator, and Heterogeneous: multi-core architecture of a mixture type (hard/soft) of cores.

The goal of the experimental work consists following:

- **Research of computing methodologies**
  This describes the methodologies of Data-Level-Parallelism (DLP) for data-intensive algorithms and High-Level-Synthesis (HLS) for function-intensive algorithms. They are sub-modus of TLP / ESL design and central motivations of this thesis work.

- **Design Space Exploration on FPGAs**
  State the formulations on FPGA platforms associated that area increasing corresponding to input applications with program speed-up of Speed&Area tradeoffs and an equation for converting Altera&Xilinx logic cells for the purpose of comparing resource utilizations between the two vendors.

After these definitions of this thesis work, the following experiment is presented:

- **Classify soft-core processors**
  This describes the configurations of LEON3, LE1, and Tiger-MIPS cores for the following experiments in this study. Homogeneous LEON3 and LE1 multi-core systems will be built for data-intensive programs. A fixed configuration Tiger-MIPS core comes with LegUp-system, which forms a general-purpose processor in that system.

- **Benchmark programs**
  List all the benchmark suits in this thesis study. More detail of DLP and HLS programs will be described in each chapter of different implementations.

- **Benchmarks of all programs in SW flow and resource utilization of single-core**
  This experiment will run all the C codes in each processor and provide the average execution times in each of them, thus the speed of each soft-core is obtained. As resource information is recorded from the synthesis results, these will evaluate Speed&Area Efficiency from DSE equations.

- **Benchmarks of DLP in Parallel-SW flow and resource utilization in LEON3 and LE1**
  This experiment will run the CM / LZW programs (DLP) in LEON3 and LE1 multi-core processor systems. Hence, speed-up ration by the number of processors in each system are obtained. As resource information in each system is recorded from the synthesis results, these will evaluate Speed&Area Efficiency from DSE equations.

- **Benchmarks of HLS in HW&Hybrid flows and resource utilization in LegUp Systems**
  Using the code profiling tool to identify the most called functions to the CHStone programs (HLS) programs; and execute HW / Hybrid flows according to results of profiled in LegUp systems. As resource information in these systems is recorded from synthesis results, these will evaluate Speed&Area Efficiency of SW/HW/Hybrid systems from DSE equations.

The logical schedule of this thesis is represented in Fig 1.1.

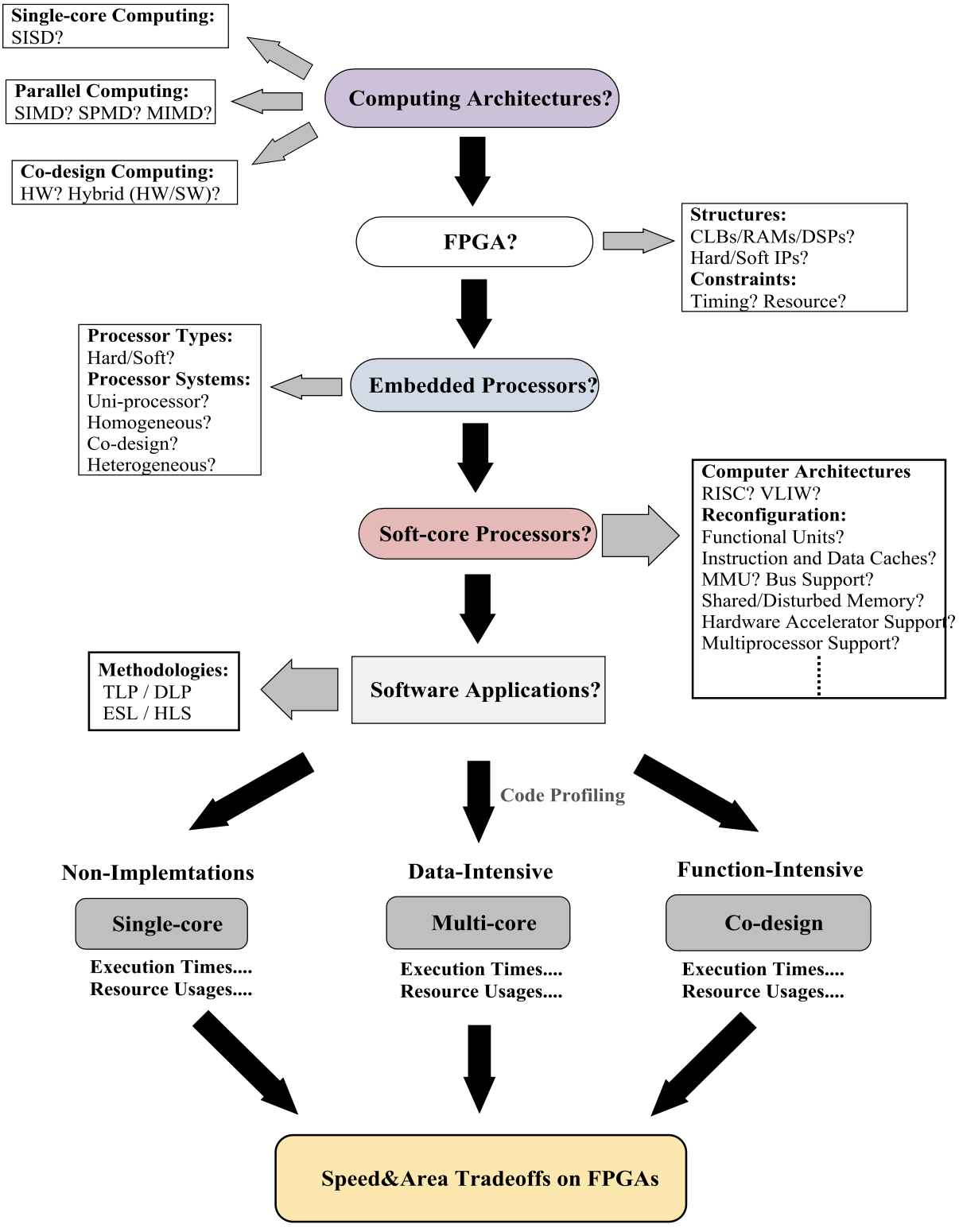

Figure 1.1 The fundamental design-flow of this thesis work.

1.4 Organization of the Thesis

The research presented in this thesis is organized in six chapters:

Chapter 2: This chapter a) outlines the history and breakthrough of the computer technology for this thesis work demands, b) compares the advantages and disadvantages of ASIC&FPGA design flows; c) defines parallel computing architectures, d) introduces abstraction-levels in ESL design e) presents physical structure of FPGA devices and configurable, f) List currently popular hard and soft-core embedded processors on FPGAs, their architectural characterizations, and resource usages lead to change maximum frequency, g) state architectures of multi-core processor system and memory types on FPGA design.

Chapter 3: This chapter introduces detail of the computing methodologies: DLP and HLS. For the DLP flow, how a program executing a separated data in multi-core processor is introduced. These state the art of the data-intensive computing for the SMPs method in chapter 5. For the HLS process, such as Scheduling, Allocation, and Binding operations from behavioural to RTL descriptions in Co-design are introduced. These establish the art of the function-intensive computing for the LegUp-system in chapter 6. Moreover, I present an approach to measure properties of the processors on FPGA-based by using consequence equations. This terminology defines the Speed&Area Efficiency estimation across each methodology of all design flows in chapter 4, 5, and 6 respectively.

Chapter 4: This chapter introduces soft-core processors: LEON3, LE1, and Tiger-MIPS theirs configurable parameters, computer architectures, bus support, and hierarchic memory components for essential design implementations. To determine the default LEON3 and LE1 cores in this research used, a large number of C source codes are introduced in the following experimental course. These programs will execute in those single-core processors respectively of the *SW* flow to obtain the average execution times to examine the performance of each processor. The Speed&Area Efficiency of the area breakdown on silicon against the mean execution times in each of the processors will be represented. FPGA Platform: Altera Stratix IV EP4 (Tiger-MIPS) and Xilinx Virtex-6 LX240T (LEON3&LE1). Both of the Altera&Xilinx FPGAs are *40-nm* process.

Chapter 5: This chapter introduces the comparisons of Instruction-Level-Parallelisms (ILP) in LEON3 (32-bit) and LE1 (32*2-bit) cores respectively and configurability in multi-core processor system of LEON3 MP and LE1 CMP. The numbers of LEON3 and LE1 cores are arranged for SMP systems. The numbers of experimental cores are from 1 up to 6. Method of reading processor ID from each LEON3 or LE1 cores is introduced. This allowed a single program to be written with a dataset splitting across each available processor that recognized by a unique CPU identification. A parallel method of the Single-Program-Multiple-Data (SPMD) process of CM and LZW applications that developed for this study is introduced. They will be executed in both systems of the *Parallel-SW* flow and results of the execution times and area. The Speed&Area Efficiency of the programs speeds up by the number of the cores in those two SMP systems will be presented and discussed. FPGA Platform: Xilinx Virtex-6 LX240T (*40-nm* process).

Chapter 6: This chapter represents an HLS tool: LegUp and illustrates its architectural components, the Tiger-MIPS processor can be augmented with a hardware accelerator. CHStone benchmark collections are provided well example codes for hardware generations in HW/SW Co-design. Initially, profiling codes using Kcachegrind tool to decide which function of the programs to be accelerated in the Hybrid system. The non-profiled codes will execute in *HW* (LegUp-HW) flow; and the most and the second called functions in *Hybrid1* and *Hybrid2* (LegUp-Hybrid) respectively. The Speed&Area Efficiency of HW&Hybrid1&Hybrid2&SW flows in LegUp-systems will be presented and discussed. FPGA Platform: Altera Cyclone II Family (*90-nm* process).

Chapter 7: In this chapter, I summarize the characterizations of LEON3, LE1, and Tiger-MIPS soft processor configuration on FPGA-based and the performance evaluation of each uni-processor in chapter 4 and lead to compare their advantage and disadvantage in terms of computing results. Moreover, to summarize the Speed&Area Efficiency of the DLP / HLS process in chapter 5 and 6 respectively. Moreover, CM and LZW programs can be implemented in both of the methodologies will be a good example to be compared and discussed this affair. Furthermore, state any remarks or further works for the following computational flows that will promote to enhance the processor-system designed for FPGA-based.

Chapter 2

Background Research and State of the Art

2.1 Introduction

Computer technology has risen in the world around the 1950s. In the beginning, machine code (assembly language) was the only means of programming a computer. When ICs were introduced in the 1960s to form the earliest microprocessors, *C programming* started to be developed to translate high-level (human) language to machine instructions. Computer architectures were defined by Michael J. Flynn in 1966 [16]. There are different types of parallel computing architectures: Bit-Level, Instruction-Level, Data-Level and Thread-Level paralleling. The "*Bit-Level*" were then implemented by doubling the word size of instruction sets during 1970s until 1985 when VLSI microprocessors were developed. "Instruction-Level" is basically the instruction pipelining organizations of the CPUs, some of them such as Superscalar and Very-Long-Instruction-Word (VLIW) architectures (were invented by J. Fisher in 1983) [17] have parallel instruction sets to achieve very high performance. However, Bit-level and Instruction-level implementations of the microprocessors are "internal parallelism" which usually fixed by the manufacturers; and Data-level and Task-level implementations are "programming parallelism" that typical customizing by software engineers.

There are two main types of semiconductor devices for ASIC and FPGA platforms. ASIC devices are integrated circuit designed for particular applications, rather than intended for general-purpose. Historically, an ASIC commercial Intel 4004 4-bit microprocessor was first introduced in 1971 by Federico Faggin [18]. FPGAs (general-purpose logic) are ICs designed to be reconfigured by a customer after standard mass-production. FPGAs are configured to accommodate any digital function and results attractive performance than ASICs for some specified applications. FPGAs contain programmable logic components, called *Configurable-Logic-Blocks (CLBs)* that are fundamental building blocks to be configurable interconnected together. It appeared in the mid-1980s [19] and the first commercial FPGA devices – XC2064 was released by Xilinx Inc. in 1985 [20]. The XC2064 had only 64 CLBs with 3-input LUTs [21]; after that, FPGA device engineering continued to grow during the mid-1990s; meanwhile, telecommunications for data processing and dedicated applications developed rapidly. Around the same time, HLS started to be researched and became commercially available by Synopsys (an EDA company) as Behavioural Compiler in 1994 [22]. Since the speed and size capacity of FPGAs had been greatly succeeded, the reusable IP cores and platform-based design started to rise in the late 1990s [23]. During the 2000s, there had been shifted for the fast ESL algorithm modelling that facilitate

synthesis, verification, and debugging of complex FPGA and SoC [22]. The algorithmic compilers (C to RTL) for HLS toolsets were released in 2004 [24].

The FPGA or SoC implementations usually include a number of embedded processors (hard or soft-core), DSP building blocks, and other IPs. Processors in embedded system come in three main categories: *Microprocessors (μP)*, Microcontrollers (μC), and *Digital-Signal-Processors (DSP)*. Microprocessors (32~64-bits) are standard processors without the memory and peripherals. Microcontrollers (8~32-bits) have built-in memory, peripherals, and many other components. Digital signal processors (8~64-bits) are specially designed for processing complete-intensive applications such as audio in speech signal processing and video in digital image processing. The wide ranges of the common architectures in embedded processors are ***μPs***: x86, ARM, Blackfin, MIPS, PowerPC, and SPARC; ***DSPs***: SHARC and SigmaDSP; ***μCs***: AVR and PIC.

In contrast to FPGAs, ASICs demonstrate superiority in terms of speed and smaller chip sizes, are ideal for high volume applications; usually have longer pipeline depths (eg. The Krait processor has 11-stage pipeline [25]) and able to be clocked at higher frequencies. At the present time, the high-end hard (ASIC) processors have optional 1, 2, 4 cores of 32-bit ARMv7 Cortex-A9 processors with 8-stage pipeline that maximum clock rate is around 1.2 GHz, optimized Level 1 and up to 1MB Level 2 Caches [26]. The advantages of FPGA soft-core processors comparing to ASIC hard-core within SoCs are customizable components (multi-processor support, buses, peripherals, internal memory, and controllers for above), obsolescence mitigation (open HDL codes for the reconfiguration), and hardware acceleration (hardware and software implementations and tradeoffs). However, for advanced embedded processor designs, the design toolchains are more numerous if the designers intend to make a hardware and software platform (eg. Co-design systems); or a specific variable type of IP cores in a platform (eg. heterogeneous multi-core systems), yet technical programming and compilers are far more complicated to be solved. Most of FPGA manufacturers embed a physical core (hard) and a reconfigurable fabric core (soft) into the FPGA silicon. To the popular FPGAs, hard processors such that the Power PC 405/440 cores embed on the Xilinx Virtex 4/5 families and ARM Cortex-A9 embed on the most of Altera FPGAs and Xilinx Zynq-7000 SoC [27]; soft processors such as the Microblaze core is synthesisable on the Xilinx families and Nios I/II or Cortex-M1 on the Altera families. Many companies, such as Gaisler Aeroflex supports embedded soft-cores (LEON3/4) and targeting to most FPGA vendors. Unlike hard-cores, soft-cores offer a wide range of flexible components, and re-programmable, and FPGA device compatibility. For example, the Microblaze processor has multiple options for FPUs, MMU, and configurable caches to name a few; and the clock frequency is around 168 ~ 342MHz [28], depending on the FPGA silicon platforms. In general, the operating frequencies of FPGA soft-cores are much slower compared to ASIC hard-cores. More detail of ASIC and FPGA processors in embedded systems are presented in later in this chapter.

Designs of ASIC&FPGA are generally implemented in a HDL such as Verilog (1986) [29] or VHDL (1987) [30] HDLs appeared to describe the functionality (RTL models of digital logic). Standard-cell ASIC designs are able to perform complex arithmetic faster than FPGA implementations along with lower power consumption and smaller chip size. However ASICs are expensive to make due to high Non-Recurring-Engineering (NRE) and their design logic cannot be easily altered once the devices are fabricated. On the contrary, FPGAs by using

state of the art silicon cells are programmable and allow for reconfiguration and reduced time to market. Furthermore, FPGA vendors provide whole silicon IP ecosystems and developing environments. The comparison of ASIC&FPGA Design flows have shown in **Fig 2.1**; FPGAs has shorter design time due to a sophisticated enumeration of the software for synthesis, timing analysis, and Place-and-Route (PAR). In ASIC design flow, such as Design for Testability to assure high reliability [31] and gate-level simulation in the process within the lowest level of abstraction are essential then the domain will be exceedingly slow and complex. ASIC design tools such as Computer-Aided-Design (CAD) system which is much more complex than general FPGA design tools. In addition, there is a time to wait for the chip manufactures and cannot be changed after submission to fabricate. However, the disadvantages of FPGAs are lower performance, larger sizes used, and more power consumption than ASICs. In summary, FPGAs cost less for small and medium volume target applications (eg. automotives and wireless communications); and when electronic devices come to high density designs for consumer appliances (eg. game consoles, smart phones, desktops and workstations), ASIC devices are cheaper in overall.

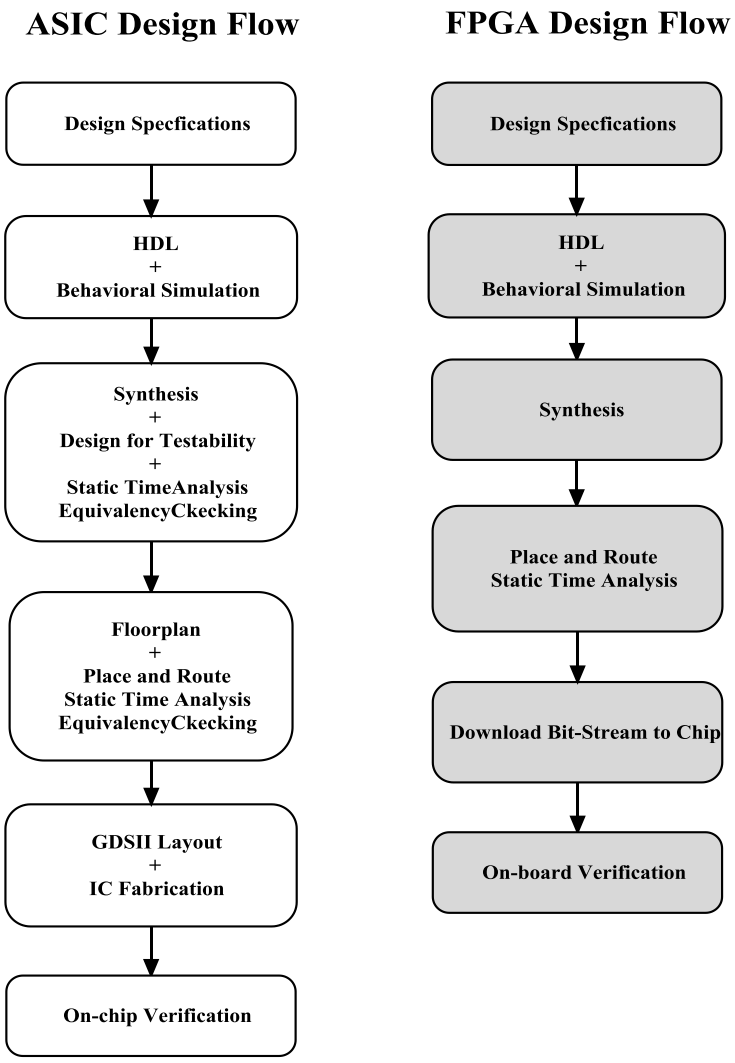

Figure 2.1 ASIC&FPGA devices Design flows.

2.2 Types of Parallelism

There are many types of parallelism for computing architectures. They are typically classified into four categories, Single-Instruction-Single-Data (SISD), Single-Instruction-Multiple-Data (SIMD), Multiple-Instruction-Single-Data (MISD), and Multiple-Instruction-Multiple-Data (MIMD), also known as "*Flynn's Taxonomy*" proposed by Michael J. Flynn [32, 33], shown in **Table 2.1**. The SIMD process is often used DSP resources to be the hardware to execute parallel data streams; the MISD is un-usually model and rarely used. The MIMD is the most common type in parallel programming by using many processors and SPMD and VLIW are other types of MIMD.

	Single Instruction	**Multiple Instruction**
Single Data	***SISD*** 1 CPU - 1 program/data	***MISD*** 1 CPU - 1 program/data
Multiple Data	***SIMD*** 1 CPU - 1 program/distributed data shared/distrusted memory	***MIMD*** Multiple CPUs – tasks/distributed data Multiple CPUs – 1 program/distributed data shared/distrusted memory

Table 2.1 Flynn's Taxonomy.

2.2.1 Instruction-Level-Parallelism (ILP)

Instruction sets are implemented in multi-stages in a processor. The overlapped instructions of an N-stage pipeline can be scheduled by N different instruction streams on N clock cycles. A simple *Reduced-Instruction-Set-Computing (RISC)* processor typically has 5-stage pipeline, I-Fetch, I-Decode, Execute, Memory-Access, and Writeback has shown in **Fig 2.2**. The longer instruction sets to be paralleled is called *Very-Long-Instruction-Word (VLIW)*, for this action, the hardware resource (ALUs and MULs) are highly required because that multiple operations (eg. addition, multiplication, and loads) often execute at the same time. The VLIW implementation is a type of the ILP that said to multiple instructions being executed concurrently per clock cycle rather than a single set.

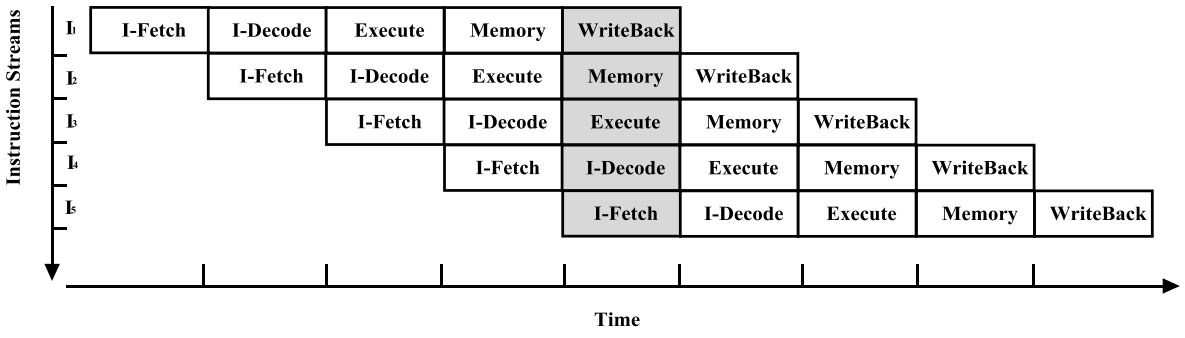

Figure 2.2 A 5-stage instruction pipeline.

2.2.2 Data-Level-Parallelism (DLP)

A multi-core processor system performs the same task on distributed data segments across multiple processors simultaneously, which is *Data-Level-Parallelism (DLP)*, also called "data-parallelism". SIMD processors are the majority type of DLP and a vector machine (Datapath) is widely added to support a main CPU to manipulate massive-parallel arithmetic. There is another way of data parallelism: *Single-Program-Multiple-Data (SPMD)*, disturbing databases are implemented in a multi-core processor system and associated separated data are assigned by a conditional statement across each core. SPMD machines are usually utilized to manage large amount of data streams in parallel processing CPUs. The DLP process is widely used in SMPs, the figures and pseudo codes are shown in **Fig 2.3**.

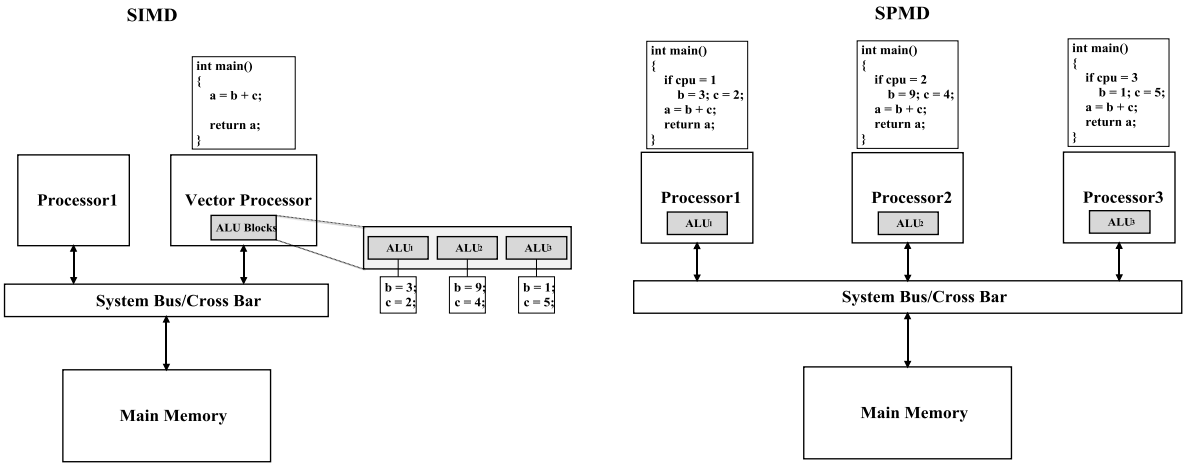

Figure 2.3 Data-parallelsim of SIMD and SPMD.

2.2.3 Thread-Level-Parallelism (TLP)

Thread-Level-Parallelism (TLP) is the most powerful and common model for parallel computing. It is another form of parallelism, comparing to DLP, a program parallelized by functions (or threads) that processes the different task across each core in a multi-processor system simultaneously and each processor executes disturbing process as its instruction is responding (eg. MIMD), which is also called "task-parallelism". Unlike SIMD, the MIMD system is asynchronous paralleling. The associated tasks are also assigned by a conditional statement across each processor, as shown in **Fig 2.4**. To the memory system, MIMD architectures can be greatly improved in distributed memory. Moreover, the SPMD method is a special case of MIMD, and sometimes people named it as TLP process; however, we normally call it "DLP" instead of TLP across whole sections in this thesis work.

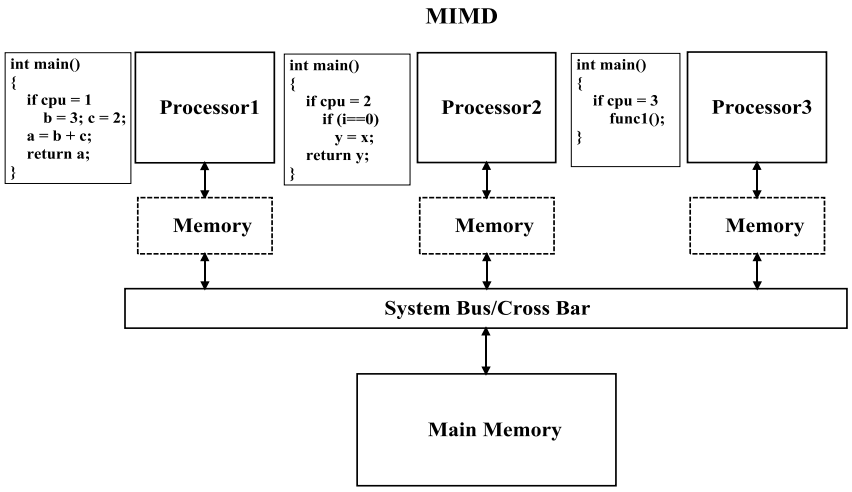

Figure 2.4 Task-parallelism of MIMD.

2.3 State-of-the-Art in Electronic-System-Level (ESL) Design

Electronic-System-Level (ESL) methodologies are implemented in complex systems for SoC or FPGA designs by using high-level language descriptions to control low-level rather than alone with the traditional RTL approach. This allows designers to optimize performance and area by converting high-level computing functions into hardware circuits; and therefore fast the time to market. The digital designs in five abstraction levels are from high to low: 1) System-level (Structures / Modules), 2) Algorithmic-level (Equations / Behavioural), 3) Register-Transfer-Level (RTL), 4) Gate-level (Netlist), and 5) Layout-level (GDSII). The top-to-down ESL design process is shown in **Fig 2.5**.

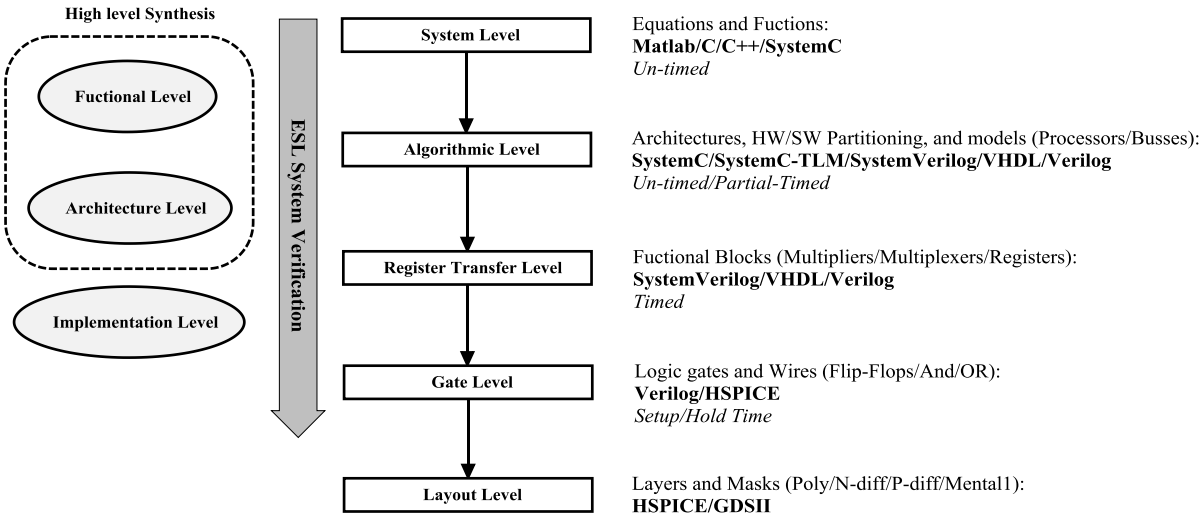

Figure 2.5 The ESL design flow and abstrcation levels.

At higher levels description, software such as C/C++, and Matlab to describe Functional-Level (micro-architecture) of the architecture components (eg. memories, processors, ALUs and buses) for an entire electronic system; and SystemC (2000) [34] or VHDL to describe Architecture-Level (logic) of functional blocks; and Verilog to describe Implementation-Level (circuit) of gates and wires. In ESL synthesis process, the high-level codes define the structure of the system (behavioural synthesis) and transform un-timed or partially timed functional modules into timed (logic synthesis) RTL implementations which are directly used to create a gate-level description.

Many ESL designs are implemented with HW/SW complementary methodologies for custom tradeoffs which a part of C to HDL are compiled and synthesis to generate hardware dataflow; and the rest of the C code is compiled to assembly language as usual in the software. These designs are such as NISC technology (C to Verilog) [35], SPARK (C to VHDL) [36], and LegUp (C to Verilog) [37] toolsets. They are referred to "*High-Level-Synthesis (HLS)*" frameworks and optimize the HW/SW implementations. However, the design process is the lack of a unified HW/SW representations that invites to the difficulties of modelling, validation, and refinement in the entire system. Generally speaking, the most common modelling language currently used for ESL design is SystemC that is easier to communicate through different abstractions.

2.3.1 Hardware/Software (HW/SW) Co-designs

In Co-design systems, hardware accelerators are separate units from the main processor, which to perform complex computations functions in the hardware. The hardware devices often have outstanding execution timing than the software. The chosen functions that most time spent in the program to perform into RTL hardware datapath to increase the total performance.

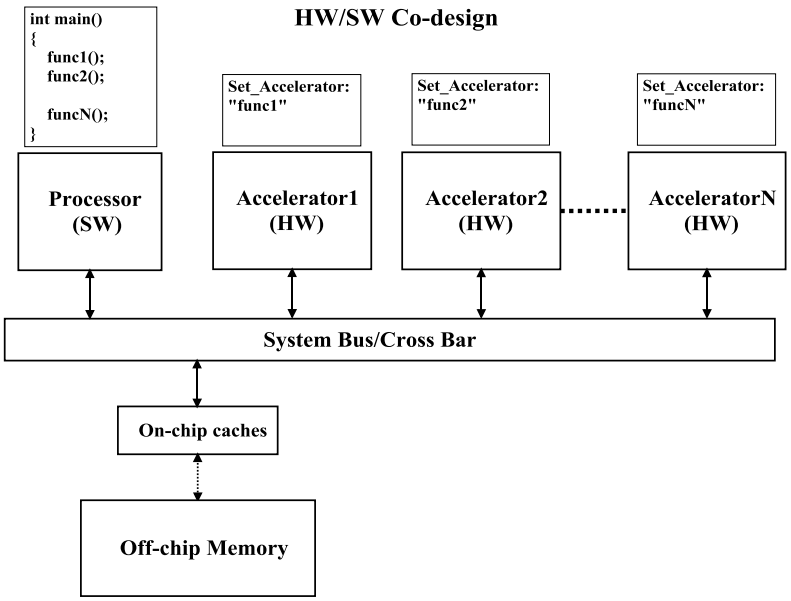

Figure 2.6 The HW/SW Co-design system.

To this computing methodology, the accelerator and a general-purpose processor running the hardware and software portions of the same programs concurrency and their contingents are able to partitioning; and the systems are able to connect multiple accelerators execute in parallel to enhance the performance, a global cache is typically implemented to slave of connected bus and shared data between accelerators and the processor, as shown in **Fig 2.6**.

However, in this thesis work, I regard to the dual-component of a hardware accelerator / a general-purpose CPU only. Specific examples of hardware accelerators are video card or graphic card in a desktop computer. For the embedded system, HW/SW implementation controls are widely used in ESL / HLS designs and built on many FPGAs or SoC chips.

2.4 FPGA Devices Overview

Previously, I have introduced the fundamental pieces of FPGAs, which are CLBs. The logic blocks wired each other in many different ways to form complex combinational FUs. The modern FPGAs combine hard or soft-core microprocessors, interconnections with the number of CLBs, and related IPs to form a “Programmable SoC”. Nowadays, "*Xilinx*" and "*Altera*" are FPGA manufacture leaders and "Lattice" is the third ranked in the world market. To the present high-end FPGA boards, 28 nm layout technology has been utilized on the latest FPGAs version such as Xilinx Virtex-7 and Altera Stratix V with bandwidth up to 2,784 and 28.05 Gb/s respectively; and both of them contain more than 2 million CLBs of equivalent gates [38, 39] that can be fitted to fulfil complex computing and reusable for the designs.

2.4.1 FPGA Fundamental Structures

Imaging a FPGA is a large "memory chip” and software engineers can write some programs to be stored to control it, and let it become a computer. The FPGA hardware architecture consists of a number of, memory blocks, CLBs, DSP units, routing matrix, and I/O pads. The majority hardware device of FPGAs is based on the Static-*Memory (SRAM)* fabricated in CMOS technology and configuration bitstream can be stored. Since, it is a "volatile" memory type; the data in the gates may lose when the power is switch-off.

There are two types of memory within FPGAs: *distributed* and *block* memory. The "distributed memory" is implemented by a number of registers, also named as "CLBs". A number of logic cells form in a CLB. CLBs are the fundamental building blocks of FPGAs that permanently sketch to the hardware target board and depending on different manufactures within different families. The "block memory" is a solid SRAM memory block located on the FPGA floorplan. Furthermore, typically an external FLASH memory for AES encryption stored.

The name of Altera CLBs is different, called *Logic-Array-Blocks (LABs)*. A FPGA CLB/LAB contains a number of logic blocks, is called *Logic-Cell (LC)* in the Xilinx vendor or *Logic-Element (LE)* in the Altera. LC/LE usages determine the resource cost of a system. The example of the FPGA hierarchical structure has shown in **Fig 2.7**, based on the Xilinx vendors. A numbers of an N-input *Look-Up-Table (LUT)* and D-type *Flip-Flops (FFs)* to construct a "*Slice*"; and two slices to form a "CLB". For Xilinx Virtex-5/6, a slice consists of 4 LUTs and 4 FFs (1-bit registers). In earlier Xilinx devices, a slice includes only two 4-input LUTs for a combinational logic and two FFs. However, the number of the input for a LUT; and the number of LUTs and FFs per slice can be increased to obtain higher performance. In high-end FPGAs of Virtex-6 families, the 6-input LUT has been manufactured; and four 6-input LUTs and FFs have been formed in the slice [40].

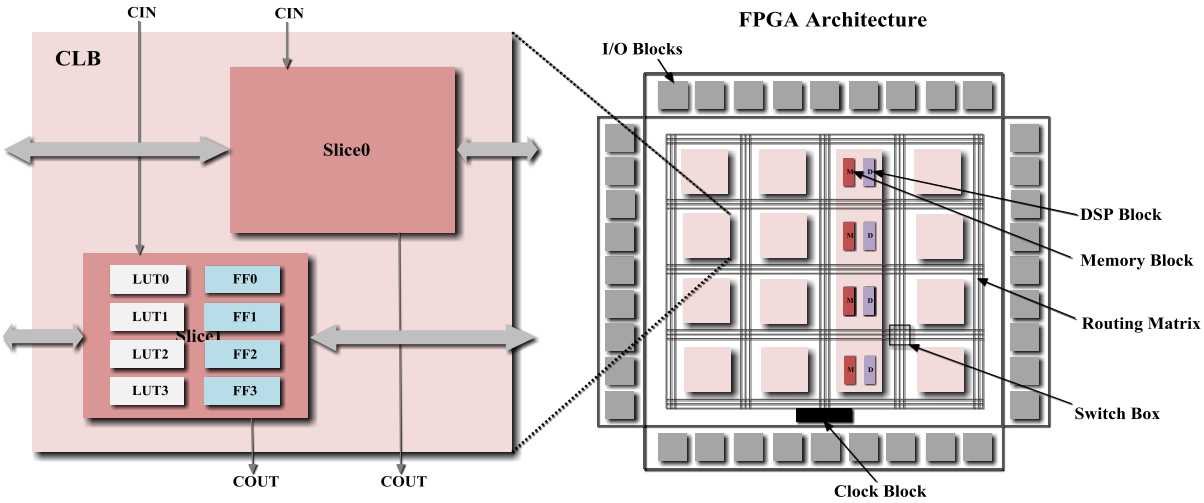

Figure 2.7 FPGA fundamental stuctures.

Moreover, DSP blocks are very dense and special FUs for multiplication arithmetic. Since multiplicand implementations in the LC/LE resource are expensive, use the DSP functionalities efficiently can preserve several hundreds of the logic cells; especially in video / audio data processing applications.

Generally, CLB keeps the same distance each other for the routing and numbers of interconnecting wires are the same in all channels. For *Place-and-Route (PAR)* synthesis, the routing signals are vertical and horizontal parried with switch boxes at interconnections. An I/O block contains input / output registers, MUXs, and clock signals. FPGA I/O blocks are surrounding the array of CLBs and interface to external components. FPGA clocking resources are clock generation and clock distribution. The clock generator is typically controlled by an analog circuit: Phase-Locked-Loop (PLL) with the Voltage-Controlled-Oscillator (VCO) or digital circuit: Delay-Lock-Loop (DLL) to generate a desired clock phase or frequency. The clock network distributes in global clock lines, regional clocks, and IO clocks. The details of the FPGA clocking resource is not in this research.

2.4.2 Advanced FPGA Features

To the higher level view of the FPGA structures, the number of CLBs are reconfigured and interconnected to comprise more compound functionality and several fixed IPs (floorplan) on the silicon, as shown in **Fig 2.8**. Many vendors develop hard / soft processors or special IPs to realize a customizable processor-system. The advanced FPGA components offer the flexible designed products to meet right requirements, faster time to market, and performance / resource tradeoffs. Moreover, a group of I/O blocks to frame an I/O bank. The I/O bank architecture plays a key role when FPGAs interface other external components. The detail of the FPGA I/O standard is not in this research.

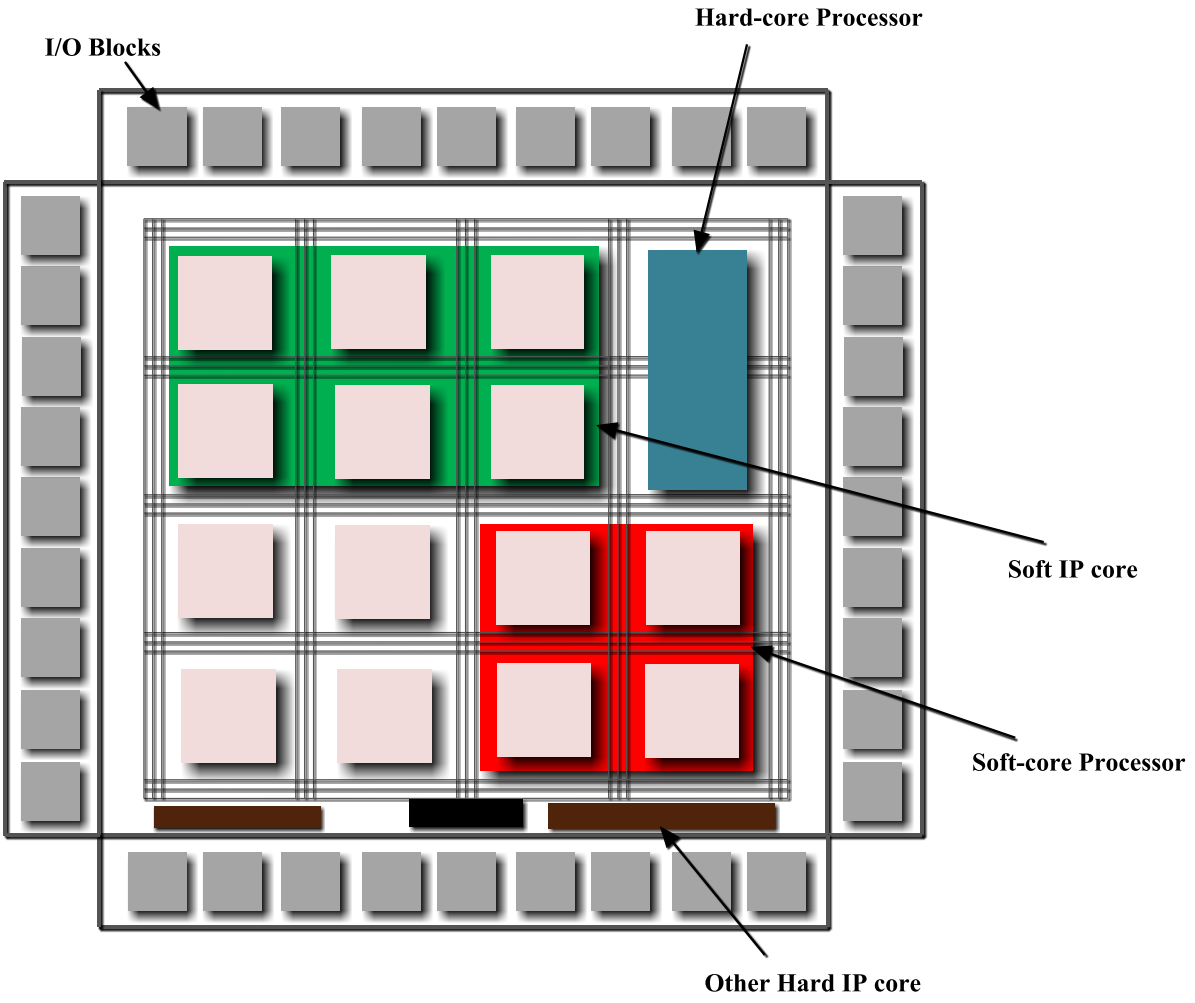

Figure 2.8 FPGA advanced stuctures.

Each size of the soft processors / IPs breakdown of FPGA fabric is ordinary pre-estimated to avoid out of resource utilization. As a general rule, the higher resource usage results, the longer routing distance and timing delays between CLBs, and the harder to process the design (synthesis and PAR); these will slower the maximum

system clock. It can be solved by placing registers along the datapath to balance delays between registers, however, it will may increase the size of the design still and most of the soft IP cores design tool do not provide this technique. In **Fig 2.9**, it is shown that the longest wiring (routing) delay determines the maximum frequency on an FPGA.

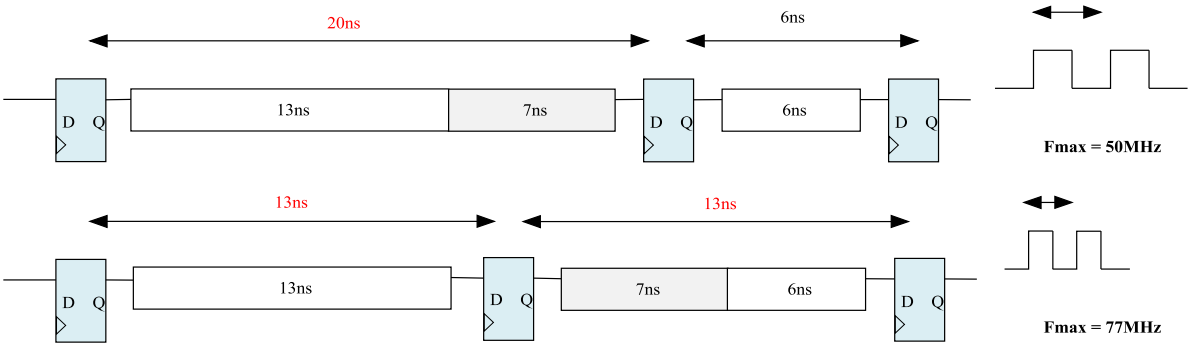

Figure 2.9 The routing delays and maximum frequency within FPGAs.

In general, for FPGA resource estimation, it is recommended to overestimate the total resource usage may be mapped onto FPGA boards. Typically said < 50% FPGA utilization of the initial estimation, because the prototyping will may change and some diagnostics logic may will be added in further. Moreover, each CLB may difficult to wire together in the PAR and degrade the execution time of the programs when devices are too full of the logic cells, and said < 75% FPGA utilization of the final design [41].

2.5 Embedded Processors on FPGAs

There are two types of the embedded processors on the FPGA-based designs: *hard* and *soft* cores. To compare soft vs hard-core on FPGA devices, the advantage of the soft-processors are utilized standard mass-produced, the lower FPGA devices cost, and the numbers of cores are customizable. However, the disadvantages are lower processor performance, higher power consumption, and larger area, because in the most part that there are breakdown most of the LC/LE resource (eg. MicroBlaze occupies around 1,200-5,000 LUTs on Xilinx FPGAs and PowerPC is a fixed gate-level IP) within the FPGA.

Through the embedded processors modern today, there are increasing interest for the "more than one core" system designs in one chip die in order to accomplish superior performance. There are three sorts of combinations to attain more than one core system: "Multi-core Processor *(Homogeneous)*", "Co-design Processor (*Hardware/Software*)", and "Multi-core Processor *(Heterogeneous)*". The ideal example of those architectures and prototypes are represented in **Fig 2.10**.

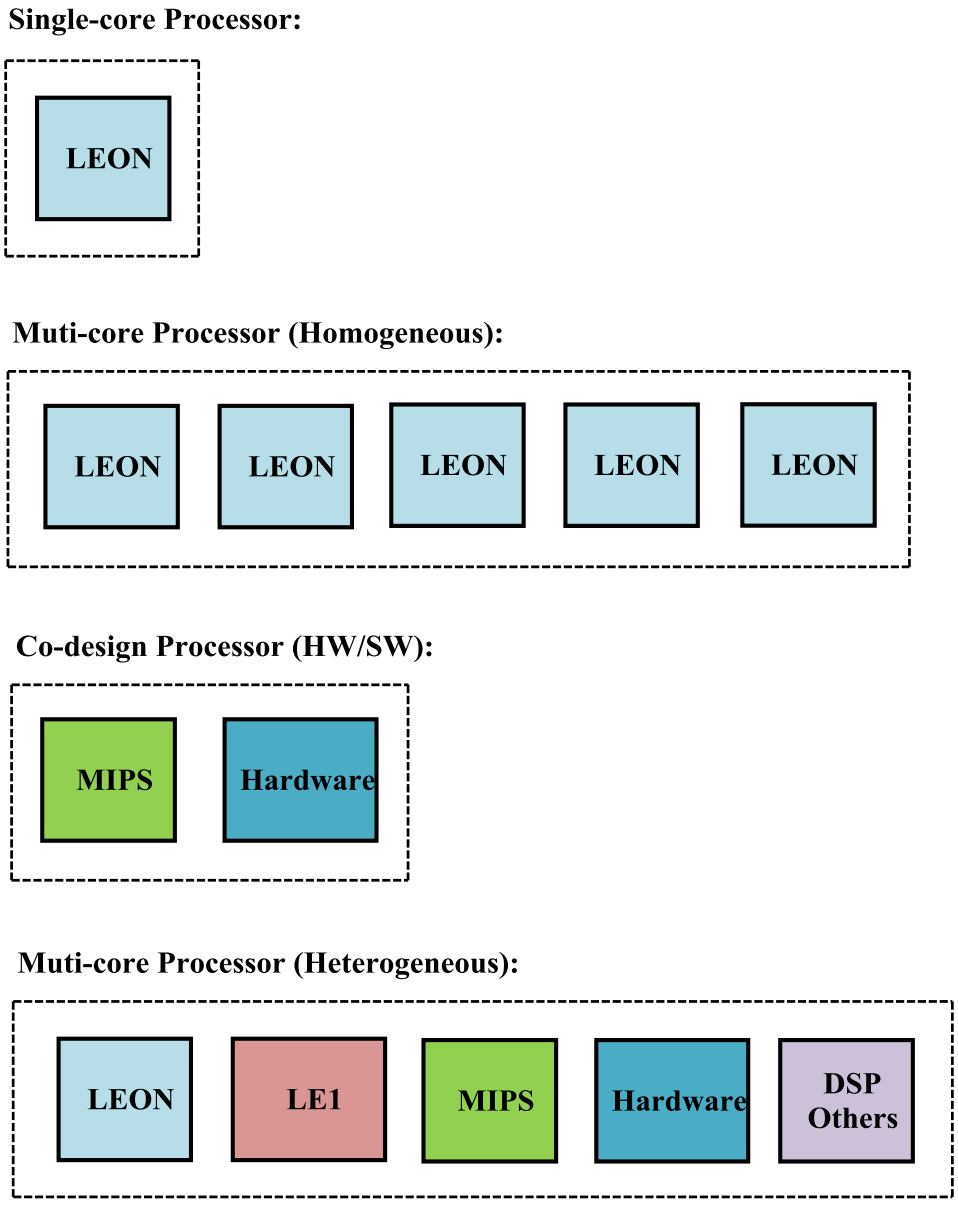

Figure 2.10 Embedded processor-systems.

To increase computational purposes by the hardware accelerator to perform function-complex programming (HLS), the Co-design system is appropriate to be used. The most common system of the multi-core processor is homogeneous; the processors are from the same manufactured and easily to be constructed. Homogeneous multi-processor system are favourable to be utilized in "data-parallelism". The most progressive system is the heterogeneous multi-processor, because each of the processors can match different applications to that core is best suited to be executed; thereupon to meet ultimate performance [42]. Heterogeneous multi-core processor are facilitated in "task-parallelism". However, they are hardly to be built up, due to very complex design tools of the software compatibility of the mixture IP components. To this thesis work, I only concern the homogeneous case in the study.

2.5.1 Hard and Soft Processors

The hard-core also refers to ASIC processors, such as Power PC 440 and ARM Cortex-A9 are embed "physical" core and dedicated part of the integrated circuit (floorplan) into the FPGA silicon. The popular fabricated hard-core CPUs on FPGAs are presented in **Table 2.2**.

Processor	Developer	Architecture	Bits	Pipeline Stages	L1/L2 Cache(KB)	MMU	FPU	Clock rate/MHz	Area/mm^2
PowerPC 440	AMCC	Power Architecture	32	7	32/256	✓	✓	667	6.0
PowerPC 460	AMCC	Power Architecture	32	7	32/256	✓	✓	600~1000	1.23
Cortex-A7	ARM	ARMv7-A	32	8	8~64/1000	✓	✓	>1000	0.45
Cortex-A9	ARM	ARMv7-A	32	8	?/8000	✓	✓	830	1.5
4KEc	MIPS	MIPS32	32	5	Instruction/ Data 8/8	✓	✓	233	2.5
4Kc	MIPS	MIPS32	32	5	Instruction/ Data 8/8	✓	✓	190	3.42

Table 2.2 Embedded hard-cores on FPGAs [43 - 46].

The soft-core also refers to FPGA processors, such as MicroBlaze, NIOS II, and LEON3/4 are built in “reconfigurable" and "synthesizable" cores then fit into the FPGA logic "fabric". They are frequently open-source / proprietary with IUs, and optional MMU and FPUs [47]. In **Table 2.3**, outline the well known manufactures soft-core CPUs suitable on the FPGAs.

Processor	Developer	Architecture	Bits	Bus Support	Pipeline Stages	MUL	FPU	Cache	MMU	Area (LUTs)
MicroBlaze	Xilinx	MicroBlaze	32	PLB, OPB, FSL, LMB,	3, 5	opt	opt	✓	opt	1,200-5,000
PicoBlaze	Xilinx	PicoBlaze	8		no	✗	✗	✗	✗	190
Nios II/f	Altera	Nios II	32	Avalon	6	✓	opt	✓	✓	1,800
LEON2	ESA	SPARC-V8	32	AMBA2	5	✓	ext	✓	✓	5,000
LEON3/4	Aeroflex Gaisler	SPARC-V8	32	AMBA2	7	✓	✓	✓	✓	3,500-6,000
LatticeMico32	Lattice	LatticeMico32	32	Wishbone	6	opt	✗	✓	✗	~2,400
Cortex-M1	ARM	ARMv6	32	AMBA2	3	✓	✗	✗	✗	2,600

Table 2.3 Embedded soft-cores on FPGAs [48 - 51].

Soft-cores are usually implemented with a HDL such as VHDL and Verilog. The benefit of using soft IPs includes configurability to optimize for suitability targeted applications, high-level of reusable design, reduced obsolescence risk, and simplified design modifications [52]. The probable configurability of soft-processors to be implemented in the following guidelines:

- **Instruction Architectures**
- **Register Windows / Clusters with Functional Units**
- **Instruction and Data Caches Support**
- **Memory Mapping I/O (Virtual Memory)**
- **Memory Type or Size and Peripheral Bus**
- **Hardware Accelerator / Coprocessor to Perform RTL**
- **Branch Predictions**

These parametrizable features in soft-cores are instantiated custom executed FUs (eg. IU & FPU with ALUs) [53]. However, different manufactured soft-processors may have different customizable and specifications that depends on the design purpose of the processors.

2.5.2 Memory Architectures of the Multi-core Processor Systems

The memory architectures in the processor-system is organized in a hierarchy system. There are two types of management: "caches" and "banks". The *cache system* is a block of temporary memory in the microprocessor that the access time is faster than the main memory, but slower than registers in the CPU. Since the 1980s, performance of the processors has been highly grown and the memories could not follow the CPU clocking speed [54], due to the data was likely to be used again, and it should have a small storage block that closed to the CPU; thus the data transfer time would be quicker and the performance had been improved. Some processor-systems have several memory divisions in a local RAM that is called *memory bank*. It is a uniform slot for storage in that memory and determined by a memory controller to be accessed.

In multi-core processor system, each core typically connects together with a *system bus* or *crossbar switches* with a main memory. There is two types of physical memory to be accessed: *shared-memory* and *distributed-memory*, as shown in **Fig 2.11**. *Uniform-Memory-Access (UMA)* shared-memory machine is the most popular way to form a *Symmetric-Multi-Processor (SMP)* system. The earliest SMP machine was introduced by Burroughs' MIMD D825 processor in 1962 [55]. In an embedded system, a *Synchronous-Dynamic-Random-Access-Memory (SDRAM)* is widely used to form the main memory in the processor-systems, because it synchronized with the system bus, advanced generations are DDR SDRAM and the DDR2 and DDR3. In a shared-memory system, multiple cores where are connected with the bus interface and all cores can access a common memory symmetrically via load&store operations. In distributed-memory, each CPU employs a local memory and communicates through a network.

The advantages of shared-memory are easier to implement in a single address space and equal speed of the data flow between each processor when programs to be loaded and stored. However, the disadvantage is adding more cores, which can significantly increase traffic of the data access between each processor and the main memory that is called "*data synchronization confliction*" or "*data overhand*". This phenomenon usually occurs in load&store instructions of the CPUs in a single main memory system. The advantages of the distributed-memory can access its own memory immediately and employ a large number of processors in the system will not cause data overhand. However, the disadvantage is difficult to implement with data communication between processors. In this study case, we consider shared-memory rather than distributed.

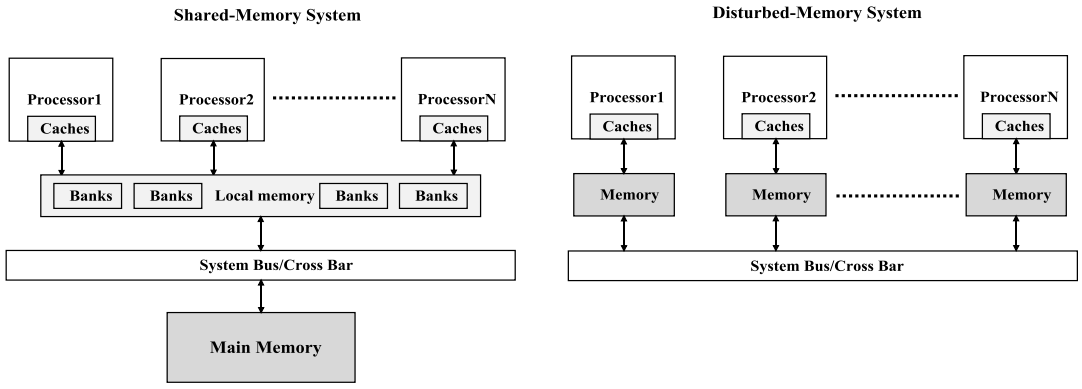

Figure 2.11 The stucture of shared / distributed memory of the multi-core processor systems.

2.6 Summary

This introduction chapter briefly describes the history of computing technology that summarized in a milestone of **Fig 2.12** with many significant breakthroughs and important emerged for embedded technologies. DLP programming of the SPMD implementation and domain of HLS designs from high-level algorithms to RTL synthesis that will be used in this study are illustrated. Furthermore, outlines the many advantages / disadvantages of FPGAs compared to ASICs and hard / soft processors in the example. In addition, the fundamental building blocks such as CLBs / memory blocks / DSPs and reconfigurable IPs alike soft-cores (logic cells) / hard cores (floorplan) on FPGAs are introduced. Moreover, the possible multi-processor systems that can be established are "homogenous multi-core processor" system to develop a data-parallelism solution or "Co-design processor" system to specialize a hardware accelerator to solve complexity function issues. These two computing methodologies generalize the aspirations of this thesis and their designed system may on different FPGA vendors. However, to evaluate their Speed&Area tradeoffs, a scheme definition and formulation are essential. More detail of them will be discussed in the next chapter.

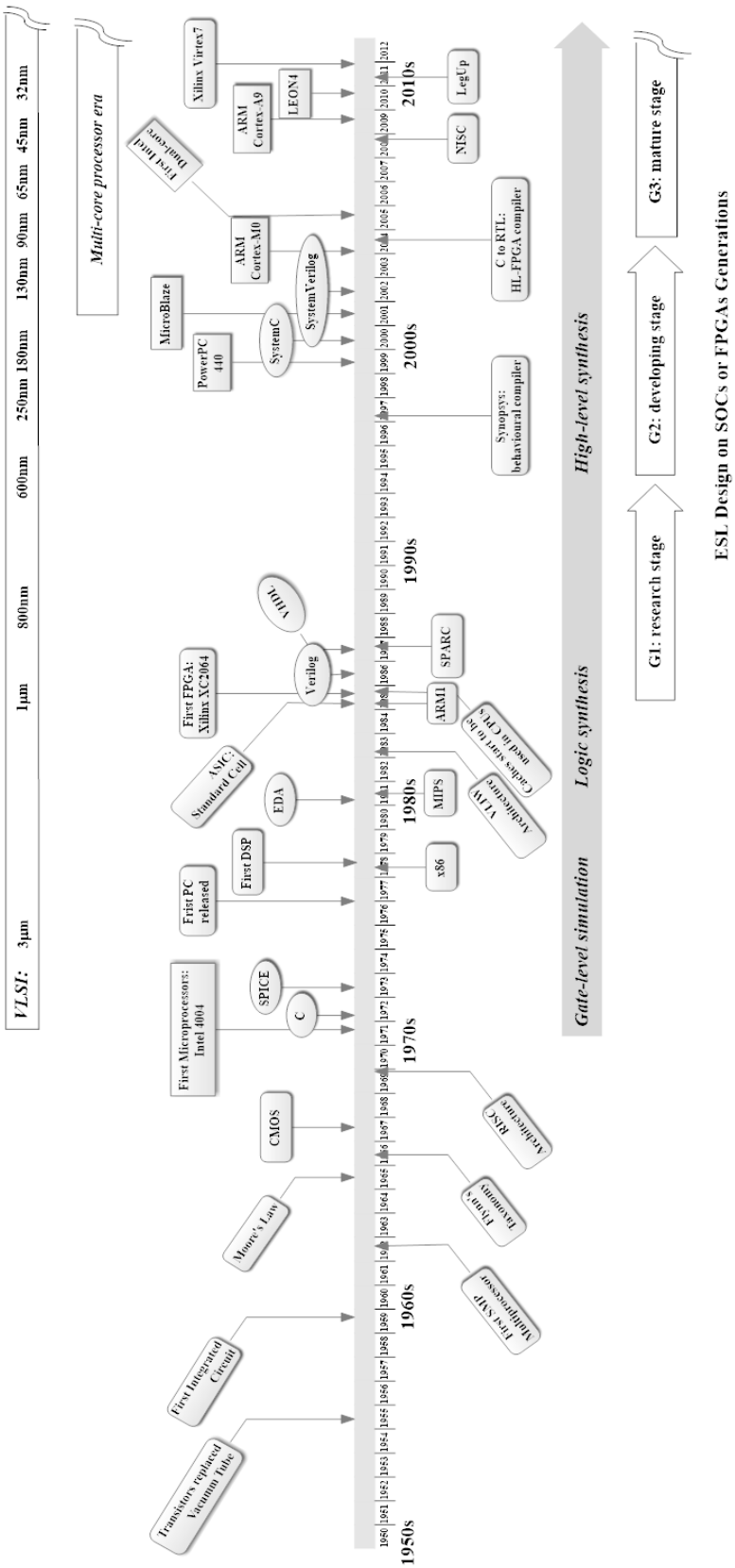

Figure 2.12 The time-line of the computer technology.

Chapter 3

Background of Methodologies and Implementations

3.1 Introduction

This chapter describes the details of DLP / HLS computing methodologies in the domain of the fulfilments of the computational subjects. To follow up the embedded processor-system structures, we should begin to look at different implementations that are working up for the specific purpose of the methodologies. The "DLP methodology" is for multi-processing that was originated to analysis of scientific research in massive numerical information and a classical solution by supercomputers of multi-core processor architectures. However, the "HLS methodology" is for creating digital hardware by high-level programming, which software engineers intended to validate and verify the hardware circuits based on software implementations; and lead to HW/SW Co-design architectures. Applications for embedded systems in DLP&HLS methodologies are video / image / audio / lossless compression of multimedia processing.

3.2 Implementations of Data-Level-Parallelism (DLP)

The *data-intensive* processing is a type of parallel computing that approach a huge amounts of data. This class of computing processing requiring multi-core general-purpose processors that divides a collection data into multiple segments across different computing nodes (cores) independently uses the same executable program in parallel [56]. The greater the aggregated data distribution, the larger number of cores, the more speed-ups are benefited.

Applications of this computational process are usually called "Database" or "Big Data", typically TBs or PBs in size from the memory. Such applications for workstations or servers are World Wide Web search-engines, biological systems, and weather forecast. However, the size of embedded SoC or FPGA applications are relative small, they are purely data compression or encoding algorithmic (eg. Huffman coding, Convolution, Sampling, and Lempel-Ziv) are typically < 1MB in size. For this algorithm proposal, I apply the SPMD (also see the definition on page 27) implementation of the DLP methodology. SPMD machines are widely implemented in distributed and also shared memory.

```
if CPUID = "1"
   lower_limit := 0
   upper_limit := Num_Data/3

else if CPUID = "2"
   lower_limit := Num_Data/3
   upper_limit := 2(Num_Data)/3

else CPUID = "3"
   lower_limit := 2(Num_Data)/3
   upper_limit := Num_Data/3

for i from lower_limit to upper_limit by 0
   array [Num_Data]
```

Figure 3.1 Pseudo C codes for an 3-core of SPMD multi-processor system.

An example of SPMD processing that is executing programs in a 3 multi-core processor system (CPUs 1, 2, and 3) as shown in **Fig 3.1**. The program representing a pseudocode that operates the arbitrary algorithm with a *for* loop in separated elements of the array *Num_Data* concurrency in each processors. The "*lower_limit*" and "*upper_limit*" are the ranges in the number of data [Num_Data] and makes to each CPU of its own copy. Now, each CPU executes array [Num_Data] for its own value of the limit by reading the *Processor ID (CPUID)* in a *if else* statement, they operate in different parts of array simultaneously, thereby distributing the data among themselves. In this experimental case study, assuming that all spit-data are fully parallelizable and the fraction of the programs are directly proportional to the number of the cores; thus, the speed-up of a program does not affected by the "Amdahl's Law". Additionally, configurations of each core are the same micro-architects such that the same resource models and performance.

3.3 Implementations of High-Level-Synthesis (HLS)

The HLS methodology is designed for *function-intensive* processing, because of the process often synthesizes the specified "function" in C to the hardware. This class of computing processing requires HW/SW compounds of acceleration units and general-purpose processors on a SoC or FPGA platform that the software is well for features and flexibility, while hardware is good for performance (also see the definition on page 30).

Those applications are algorithm complexity by function-levels in mixtures of digital compression or encode / decode algorithmic for multimedia software codes, such as JPEG (including decode, Huffman coding, and DCT) and ADPCM (including encoding and decoding).

HLS process (see **Fig 2.5**) automatically translates a behavioural description (set of operations and data dependencies) at the algorithmic level of to a structural description (FUs, memory elements, MUXs, and buses) at the RTL level under constraints. Behavioural descriptions are usually expressed using HDLs such as *Verilog*; an input specification is transferred by a compiler into a formal model that represented with a *Data-Flow-Graph (DFG)*, where operations for data dependencies are easily identified. There are three main steps in HLS process: *Scheduling*, *Allocation*, and *Binding* [57], showing in **Fig 3.2**.

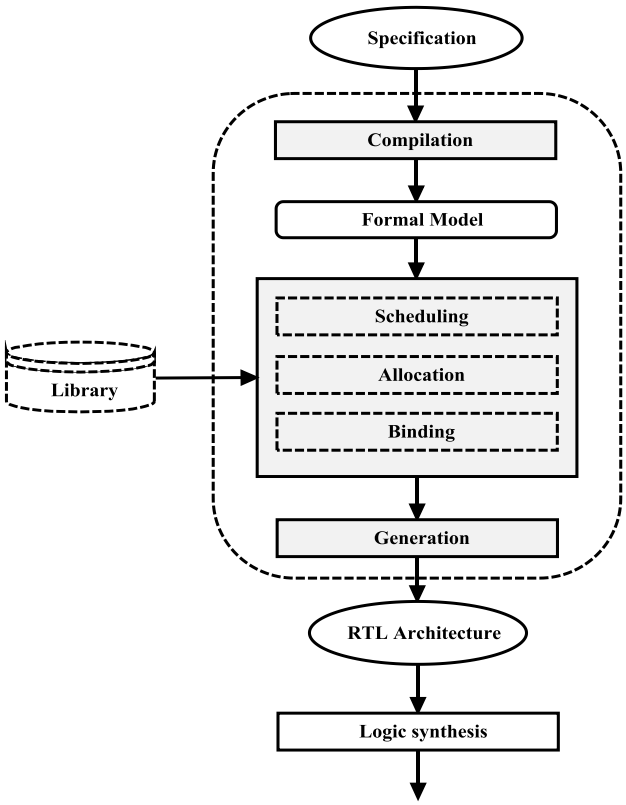

Figure 3.2 The HLS design flow.

The scheduler determines the instants at which the executions of the DFG operations start. The scheduler optimizes the design execution time or its FUs utilization such that the implementation cost is minimized. Datapath synthesis allocates operations to functional modules, variable to registers and connects the functional modules using MUXs. The aim of datapath synthesis is to minimize the amount of registers and MUXs. The results of scheduling and datapath synthesis are a structural description at the RTL level. This description is finally translated to gate-level using logic synthesis.

- **Scheduling**

  Behavioral description does not include any information about resource and timing requirements and to provide such information, scheduling algorithms are employed. The basic idea of scheduling is to

distribute the algorithm operations such as additions and multiplications across DFG in such way that the given Resource-Constraints (RC) or Timing-Constraints (TC) is met. The fundamental concepts of un-constrained scheduling are *As-Soon-As-Possible (ASAP)*, each operation scheduled into the earliest possible control step, and *As-Late-As-Possible (ALAP)*, each operation scheduled into the latest possible control step that showing in **Fig 3.3**. In HLS, duration of FUs express in multiple of the system clock period that is called a *control-step* (*c-step*) for the scheduling length.

- **Allocation**

  For datapath synthesis, generating structural datapath realization from scheduled DFG are the last steps of allocation. The allocation chooses FUs and registers or busses from the component library selecting by HLS tool the one that best matches the design constraints, depending on the synthesis tradeoffs.

- **Binding**

  For datapath synthesis, generating structural datapath realization from scheduled DFG are the last steps of binding. Binding assigns operations to FUs, variable to registers, and data transfer to bus instance. Optimization with the aim of the cost of registers and connecting FUs such that the cost of interconnect like number of MUXs can be minimized (MUXs are expensive to implement in FPGAs using LUTs). Register binding more demanding because the set of values must be assigned to the minimal number of registers.

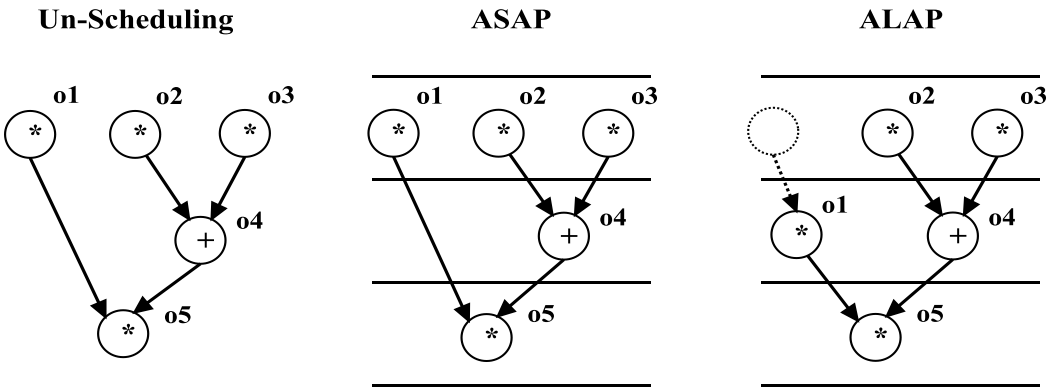

Figure 3.3 ASAP and ALAP scheduling.

Generally, ALAP scheduling produces less hardware and more execution time than ASAP scheduling. For constrained scheduling, one wish to synthesize a design subject to silicon area constraint (fixed number of FUs) and time constraint (fixed number of c-steps). RC scheduling is aiming to operate to c-steps such that the execution time is minimised for a given number of FUs. This requires optimal utilization of available hardware resources. Moreover, TC scheduling is real-time applications require system functions to be executed in a fixed, pre-defined time and the aim of synthesis is to minimise the number of FUs.

In this thesis work, a variety of powerful scheduling tool has been explored and created by J. Cong and Z. Zhang, *System-of-Difference-Constraints (SDC)*. The advantage of SDC-based scheduler is a versatile scheduling style in different types of constraints including resource, timing, frequency, and latency under a mathematical Linear-Program (LP) formulation framework [58]. This scheduling algorithm is promising to be used in LegUp design tools.

3.4 Design-Space-Exploration (DSE) on FPGA-Based

Any embedded processor-system may only to be designed on a FPGA vendor. However, to evaluate and compare its resource-level on different vendors are critical. I formulate an equation to represent the resource usages comparison between Xilinx and Altera that being used in this thesis work. The *Design-Space-Exploration (DSE)* solutions of the area increased by programs speed-up for the global FPGA-based MPSoC design are introduced. Moreover, many factors such as the number of ALUs / MULs, memory size, bus structures, and versions of the design or synthesis tools will effectively result different area breakdown on silicon and the execution times.

3.4.1 Xilinx and Altera Resource Usage Conversion

Generally, the geographical features and interconnectivity types of Xilinx&Altera FPGAs are not the same. Also, the LC/LE structures between Xilinx and Altera are rarely the same, so basically it's difficult to compare these two objectives. However, there is a common regulation, it is usual to compare the number of LUTs and not the number of registers between different FPGA vendors. For the Xilinx Virtex-5/6 devices, a LUT can be configured as one 6-input LUT with one output, and with a register (some of them to be two 5-input LUTs with separated outputs) [59] is called "*LUT Flip-Flop Pairs*". The common figure is generalized in **Fig 3.4**. The carry logic (a Full-adder) is utilized to compute the programming arithmetic and the selection bit “S” is connected through the MUXs to be programmable. Note: 1 CLB = 4 LUT-FF Pairs.

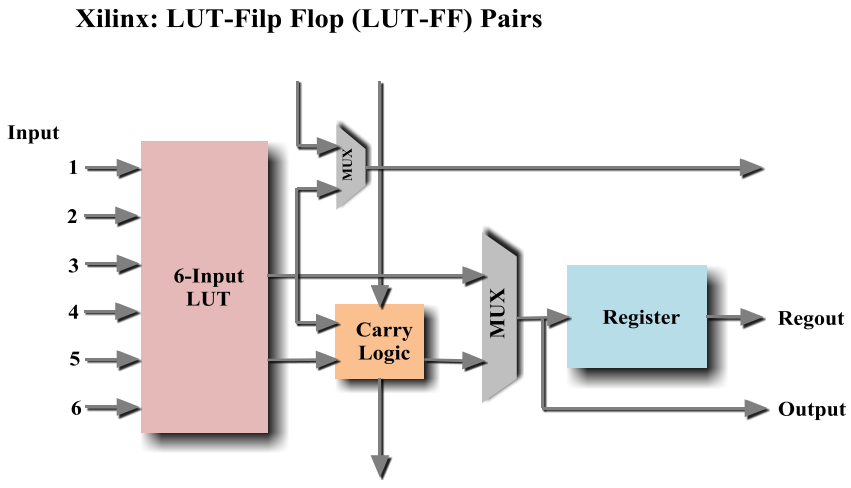

Figure 3.4 A Virtex-5 LUT-FF Pairs [60].

For the Altera Stratix III/IV series, the portion is called "*Adaptive-Logic-Module (ALM)*" and that is more area effect than a LUT-FF Pairs of the Xilinx resource. It consists of an 8-input combinational logic, two registers, and two adders, as shown in **Fig 3.5**. The all-in-one combinational cell is divided between two Adaptive-LUTs (ALUTs) and consisting of a various combination (two independent 4-input LUTs or a 5-input and a 3-input LUTs) [60]. Note: 1 ALM = 2.5 LEs and 1 LAB = 10 ALMs.

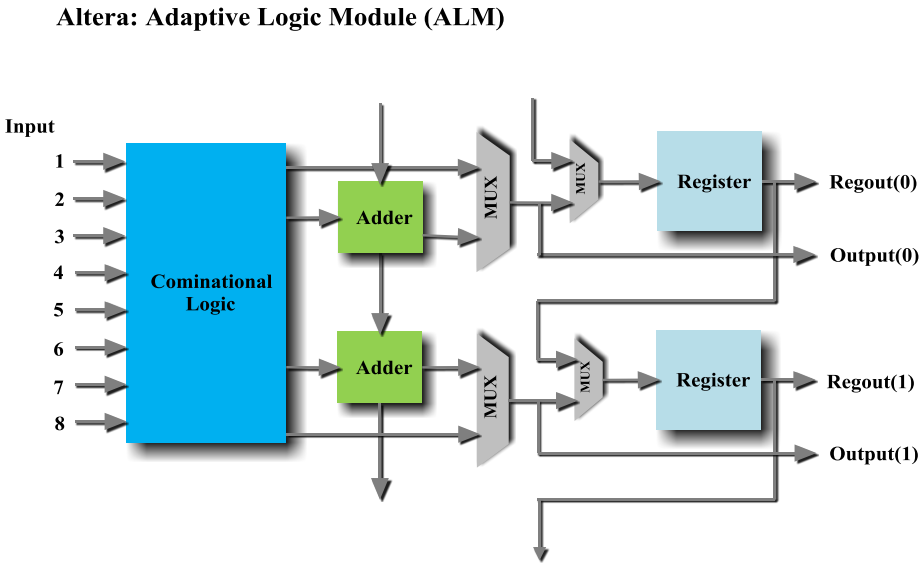

Figure 3.5 A Stratix III ALMs [60].

By advance, according to benchmark comparison results of the logic efficiency between Xilinx LUT-FF Pairs & Altera ALMs in normalized relative capacity (65nm post-fabricated chips), and the routing connectivity of the Altera LABs have greater efficiency than Xilinx CLBs in a given building block; on average, the Stratix III logic density is "1.8 times" advantage over the Virtex-5 [61]. In addition, device architectures of Virtex-6 and Stratix IV (45nm) are not changing, comparing the older version. To follow up this generalization, the equation to be represented the number of logic resource from synthesis data between Xilinx&Altera FPGAs, which are possible and prevailed:

$$\textit{Stratix IV DATA ALMs} \times 1.8 = \textit{Estimated in Virtex-6 DATA LUT-FF pairs} \qquad (3.1$$

3.4.2 DSE of Multi-core Processor System

In this evaluation case, LEON3 and LE1 homogenous multi-core processor systems are for the MPSoC platform. Assume that each of the cores is equal in size (at the same configurations); thus the number of resource utilizations is directly proportional to the number of cores have increased. The targeted platform is the *Xilinx Virtex-6* FPGA. The following equations below are used of DSE evaluations in this platform:

$$Program\ Speed\text{-}up = \frac{Execution\ \ Time\ of\ Uniprocessor}{Execution\ \ Time\ of\ Multiprocessors} \quad (3.2$$

$$Area\ Cost = \frac{Area\ of\ Multipro\ essors}{Area\ of\ Uniprocessor} \quad (3.3$$

$$Speed\&Area\ Efficiency = \frac{Program\ \ Speed-up}{Area\ of\ Cost} \quad (3.4$$

The "Program Speed-up" represents to less time spent with the number of processors have increased (Parallel-SW), the "Area Cost" associated with the single-core area with the multi-core area, and the "Speed&Area Efficiency" is associated the area cost with the program speed-up of the design.

3.4.3 DSE of Co-design System

In this evaluation case, the LegUp Co-design (Accelerators / Tiger-MIPS processor) system is for the MPSoC platform. Where the area depends on input applications (algorithm types and size of codes) and how many functions are accelerating (size of HW/SW partitioned in the hardware). The targeted platforms are *Altera Cyclone II FPGAs*. The following equations below are used of DSE evaluations in this platform:

$$Program\ Speed\text{-}up = \frac{Execution\ \ Time\ of\ SW}{Execution\ \ Time\ of\ HW/Hybrids} \quad (3.5$$

$$Area\ Cost = \frac{Area\ of\ HW/Hybrids}{Area\ of\ SW} \quad (3.6$$

$$Speed\&Area\ Efficiency = \frac{Program\ \ Speed-up}{Area\ of\ Cost} \quad (3.7$$

The "Program Speed-up" represents to less time spent of the function accelerated (HW or Hybrid) with the time spent in the single-core (SW), the "Area Cost" associated with the single-core area with the HW or Hybrids area, and the "Speed&Area Efficiency" is associated the area cost with the program speed-up of the design.

3.5 Summary

This chapter introduces DLP&HLS methodologies, implementations, and available applications. To summarize these architectures, SPMD is a form of DLP methodology using multi-core processor systems that across each core to handle massive-data implementations. A software technique of reading CPUIDs from each processor to run each data segment is introduced. HLS process is part of ESL design, which focuses on High-level to RTL-level synthesis for generating a piece of hardware on silicon to increase performance. The main streams of HLS are scheduling, allocation, and binding. Moreover, the LC/LE conversion of Xilinx&Altera that have been presented in order to estimate resource cost between them. Furthermore, to evaluate the Speed&Area Efficiency of FPGA-based MPSoCs, I have introduced a series of equations in multi-core processors and Co-design systems. However, it is not yet to start the evaluation of DLP / HLS designs. To the next chapter, the SW flow benchmarks of LEON3 / LE1 / Tiger-MIPS single-core processors for each of these designs will be evaluated elementary.

Chapter 4

Evaluation in Single-core of Soft Embedded Processors

4.1 Introduction

These embedded experimental approaches are "SW" flows that run the entire C-based programs to the general-purpose µPs. LEON3 and LE1 cores are DLP implementations in homogenous multi-core processors and the Tiger-MIPS core is HLS implementations in LegUp Co-design system. Firstly, I represent the configurations of LEON3, LE1, and Tiger-MIPS respectively. Secondly, I execute a wide-range of digital multimedia codes in each processor so as to collect a large volume of benchmark suites. Thirdly, I average those benchmark results; and therefore to evaluate the performance of each processor. To evaluate the area on FPGA-based, LEON3 and LE1 cores are being synthesized on the Xilinx Virtex-6; however the Tiger-MIPS core of the LegUp-system can only be synthesized on Altera FPGAs and on the Altera Stratix IV in this proposal.

4.2 Background of LEON3, LE1, and Tiger-MIPS Processors

LEON and MIPS processors are RISC architectures that instruction formats are fixed (typically 32-bit in embedded µPs), and the LE1 processor is a VLIW architecture, which the instruction length is variable (64-bit in this case study). More details about difference between RISC and VIW architectures are discussed in **Section 5.2.1**.

To summary these uni-processors trait, the LEON3 processor is highly configurable, especially for cache-system and on-chip debug units, and available to be synthesized on most of manufactured FPGA boards; a LE1 VLIW processor is highly internal configurable to perform complexity computing and able to be synthesized on Xilinx Virtex families; both of these processors are established to minimal configuration to save FPGA resource; the Tiger-MIPS processor is a fixed soft-core by the LegUp designers that only able to be synthesized on the Altera Development-and-Education *DE2* (Cyclone II) or *DE4* board (Stratix IV) [62, 63] at present.

4.2.1 The LEON3 Processor

The *LEON3* soft-core is a 32-bit RISC highly configurable through "VHDL" model, compliant with the IEEE-I754 SPARC V8 architecture in embedded systems. The configurable LEON3 core is located in *Gaisler-Research-Library (GRLIB)*. GRLIB is a collection of reusable IP cores, which is based on the AMBA AHB/APB on-chip buses. Each of the IP components has a particular vendor that specified by a VHDL package in each library [50], and can be easily created by "Xconfig" GUI tool. Fig 4.1 shows that a LEON3 core and its important configurability and available interface. The main LEON3 core functionalities are following:

- **FPUs and Coprocessor**
- **Number of Register Windows**
- **Instruction and Data Caches**
- **Local I&D RAM**
- **MMU**
- **DSU**
- **AMBA Interface**
- **Multi-processor Support (LEON3 MP)**

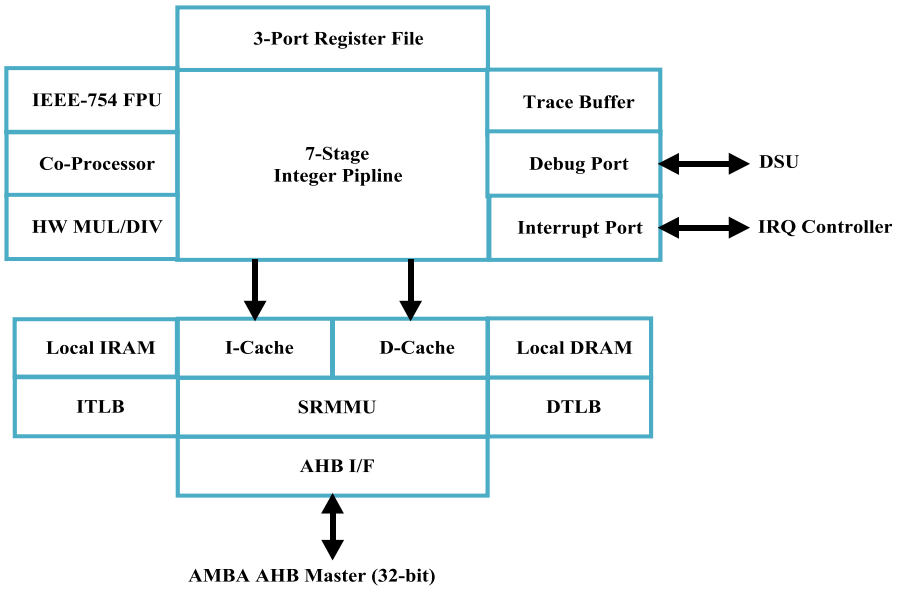

Figure 4.1 LEON3 processor core block diagram [64].

The LEON3 core is able to interface for the FPUs and custom Coprocessors, these two cases are not used in this study. The 3-port register file has (2 read, 1 write) ports with separated address and data bus, which includes the *register windows*. It is a group of general purpose "*r*" registers. The IU contains total *r* registers of 8 global plus circular stack from 2 to 32 sets of 16 registers [65, 66]. *Multiplier and Divider (MUL/DIV)* operations are supported by SPARC integer multiply and divide instructions. For instance, to perform a 32x32-bit integer multiplication, will result 64-bit and 64 divide by 32 bits, result a 32-bit; and 2-cycle latency is the lowest for the best performance. Static branch prediction reduces the penalty for branches exceed by instruction that changes the condition codes, normally improves the performance with 10 - 20% on most control-type software. *Single-Vector-Trapping (SVT)* is also from SPARC V8 architecture to reduce code size for embedded applications.

The LEON3 core is interfaced to the AMBA AHB bus master to load/store data to/from the cache system. Both of *Instruction and Data (I&D)* caches are configured to implement a "direct-map" cache or a "multi-set" cache with set associativity 2 - 4, the set size is the range of 1 - 256 KB, divided into cache lines with 16 or 32 KB data; for multi-set caches, four replacement polices options can be selected: *Least-Recently-Used (LRU)*, *Least-Recently-Replaced (LRR)*, *Random*, and *Direct*. The set size of both local I&D RAMs are configured up to 256KB. *Memory-Management-Unit (MMU)* of the LEON3 core is the full SPARC V8 MMU specification (mapping between multiple 32-bit virtual address spaces and 36-bit physical memory), however MMU is not used in this study. *Debug-Support-Unit (DSU)* is implemented to interface for the LEON3 core of debug mode and provide up to four watchpoint registers; behaves as an AHB slave accessed by the AHB CPUs master, and also AHB trace buffer can monitor and store executed or read out instructions for debug interface. Moreover, multi-processor methods are feasible on the LEON3 SoC template [64].

The LEON3 IU is 7-stage pipeline with Harvard architecture, shown in **Fig 4.2**. This shows the structure of the internal pipeline stages in the core. I-cache is fetched at Instruction-Fetch if that is enabling and there is an instruction in the cache; instructions are decoded at Instruction-Decode and the CALL / Branch target address are executed; operands are read from the register files at Register-Access; executes operations such as ALUs, shifts, and MUL/DIVs; D-cache is fetched or written at Memory-Access; interrupts are through at Exception; and any results of operands or caches operations are written back to register files.

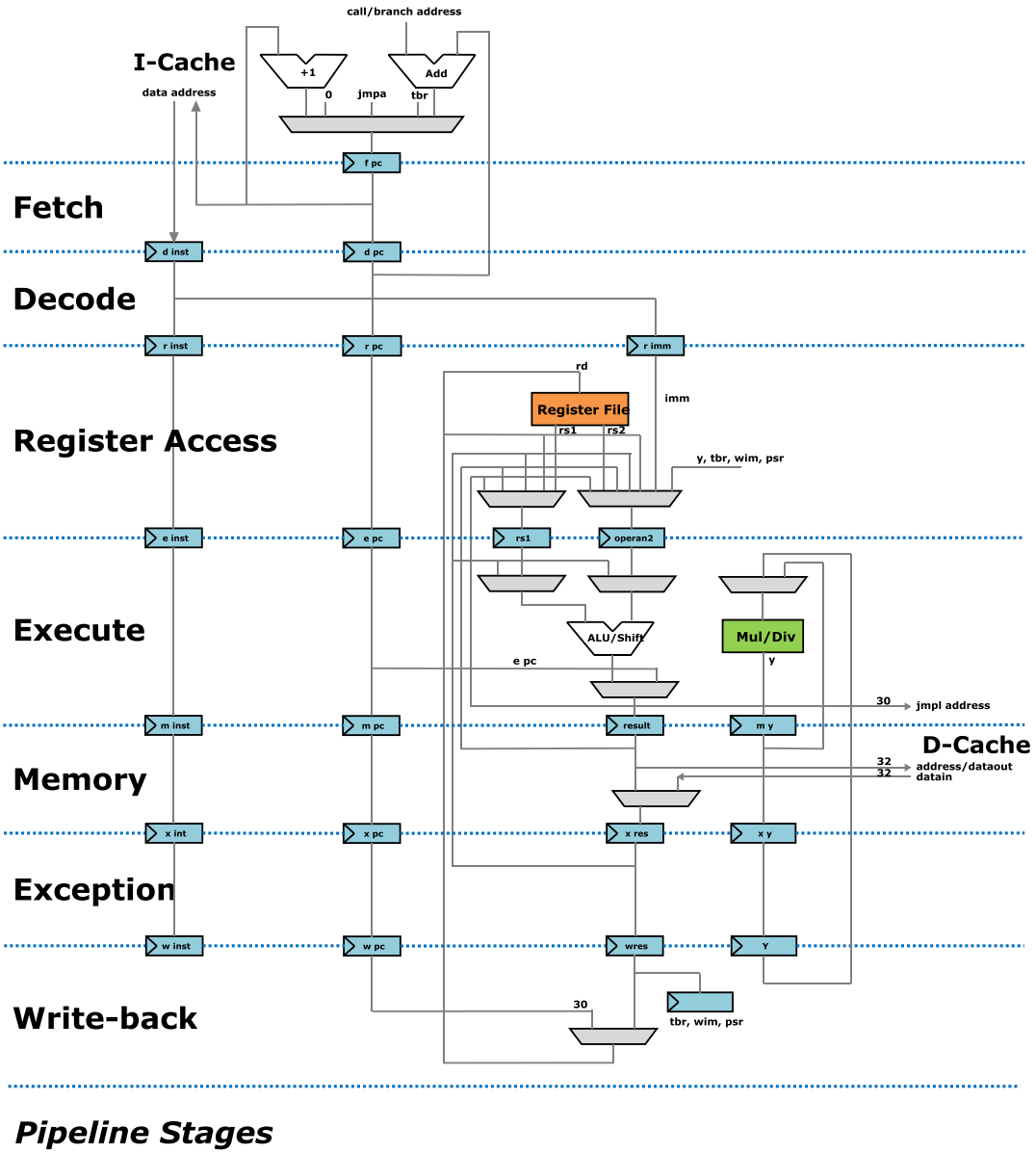

Figure 4.2 LEON3 CPU core IU datapath [64].

For a single-core of LEON3 configuration: ***8 register windows*** of default setting, enabling ***MUL/DIV instructions*** and ***2-cycles latency*** of ***32 x 32 pipelined hardware MULs***, no FPU since only IU value are calculated, ***16KB I&D-cache / 32 Byte per line*** of the ***"Direct-mapped"*** associativity, ***64KB I-RAM*** and ***256KB D-RAM***, there is no need MMU of virtual memory in this study, ***4KB instruction / AHB trace buffer*** and ***2 IU watchpoints*** of the ***DSU***, ***one pipelined load delay*** to obtain best performance, ***static branch prediction***, and ***SVT*** is enabled. For information about AMBA bus configurations and how to use the GRMON debug of the terminal on the computer screen will be shown in **Section 5.2.3** of LEON3 MP configurability.

4.2.2 The LE1 Processor

The *LE1* (32-bit for 1-width) VLIW [67] is highly parametrizable in both architectural and micro-architectural views and "VHDL" model implementation. It presents many architectural parameters to the programmer to fully customise the hardware being produced. The main microarchitectural parameters of the LE1 core are following:

- **ISSUE_WIDTH**
- **CLUSTERS**
- **IALUs**
- **IMULTs**
- **IALU_LAT**
- **IMULT_LAT**
- **IRAM_SIZE**
- **DRAM_SIZE**
- **LSU CHANNELS**
- **DRAM_BANKS**
- **LE1_PROCs (LE1 CMP)**

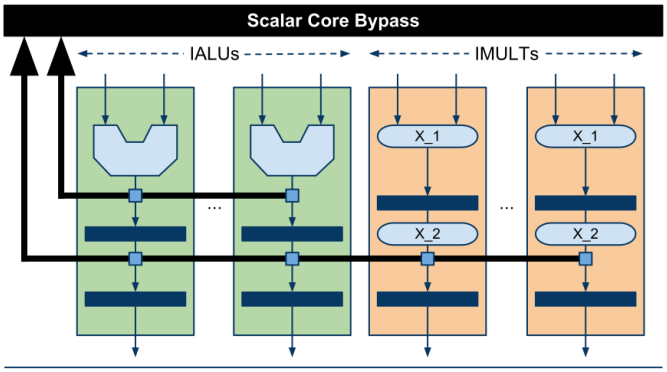

Figure 4.3 IALU/IMULT and IALU LAT/IMULT LAT for the LE1 core [67].

The *architectural width (ISSUE_WIDTH)* of the LE1 processor can be optimized in a number of RISC instructions. There are up to *CLUSTERS* (group of ALUs, MULTs, and Register Files) clusters, each with its own register set, Integer (SCore) and Floating Point (FPCore) datapaths.

The configuration of the SCore is dependent on the *Integer-ALUs (IALUs)* and *Integer-MULTs (IMULTs)* parameters which define the number of FUs available for executing integer based arithmetic operations. The latencies of the IALUs and IMULTs are depicted in **Fig 4.3**. These values are modifiable, lower latencies and more FUs result in extra complexity within the silicon and thus a higher number of 32-bit result busses to the Score bypass logic to have the results from the SCore available at an earlier time. This ultimately proves to be the critical path in the processor. The *LE1_PROCs* is configuration of the number of contexts for multi-processor methods.

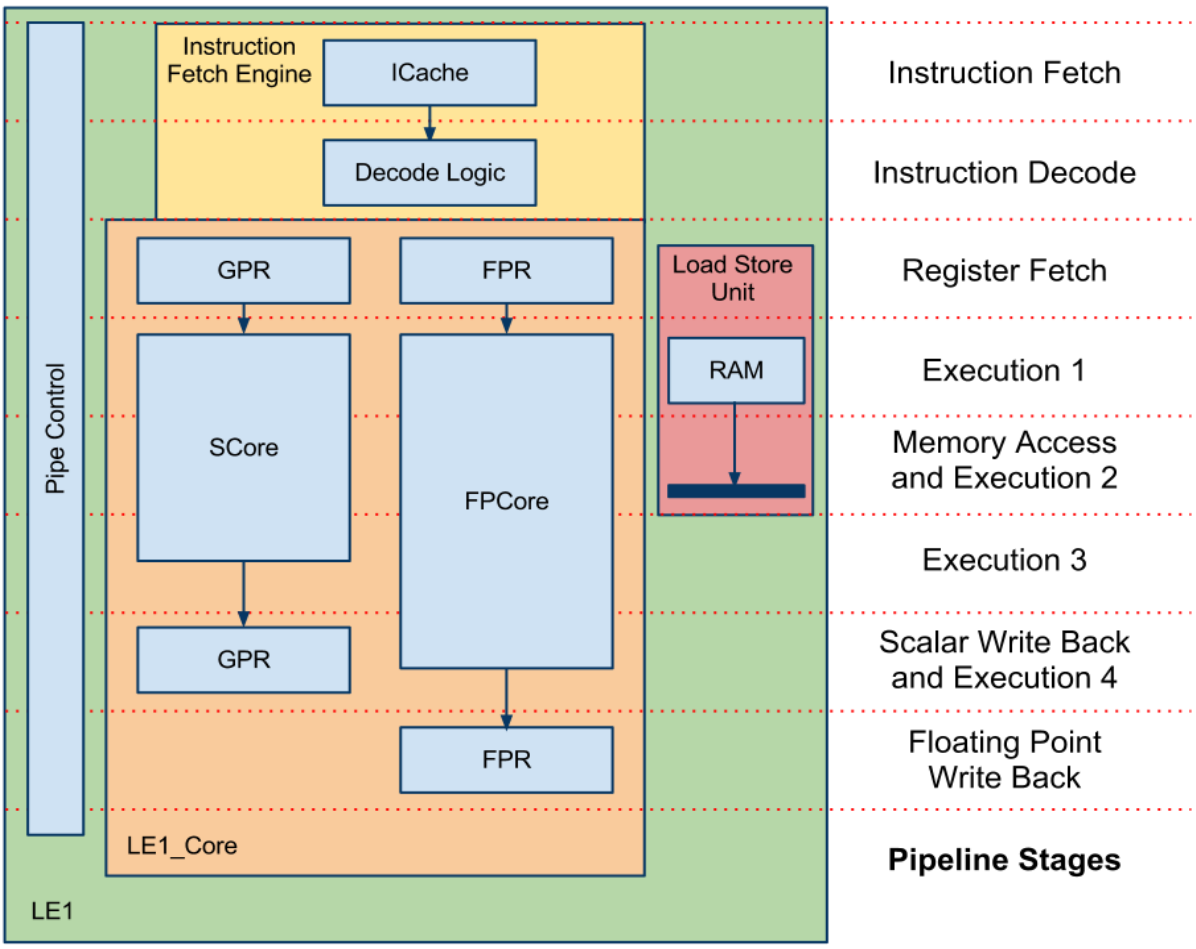

Figure 4.4 LE1 CPU core schematic [68].

The LE1 has a 7-stage pipeline (ignoring the FPCore stage), shown in Fig 4.4. This shows the internal structure and a breakdown of the internal pipeline stages of the LE1. The Pipe Control block depicts the primary control mechanism which schedules the full flow of instruction fetch and decode to data execution. It is responsible for initialising internal registers and memory sections through a debug mechanism from a host machine as well as maintaining the control space of the LE1 through a collection of state machines which schedule the overall system LE1 execution. As well as the Pipe Control, the CPU is composed of an *Instruction-Fetch-Engine (IFE)*, *Load-Store-Unit (LSU)* and the main execution core (LE1 CORE). The IFE maintains the I-cache and associated state machine. Each long instruction word can be up to two times ISSUE_WIDTH operations wide due to the inclusion of 32-bit immediate for large integers and addresses which results in the IFE controlling interlocks to retrieve all required instructions when they span more than one I-RAM location. The banked, shared memory is accessed from the LE1 CORE through the LSU. The numbers of channels (*LSU CHANNELS*) to the memory along with the banking system (*D-RAM BANKS*) of the memory are both modifiable within the LE1 configurations. Finally, the LE1 CORE includes the main execution data paths of the CPU.

For a single-core of LE1 configuration: ***2 ISSUE WIDTH***, ***2 CLUSTERS***, ***2 IALUs***, ***2 IMULTs***, ***64 KB IRAM_SIZE***, ***256 KB DRAM_SIZE***, ***2 LSUs***, and ***2 DRAM_BANKS*** respectively.

4.2.3 The Tiger-MIPS Processor

The *Tiger-MIPS* processor is a RISC 32-bit "fixed-core" based on the original *MIPS (Microprocessor-without-Interlocked-Pipeline-Stages)* 32 architecture that through "Verilog" HDL and designed by Ben Roberts and Gregory Chadwick to be used with Altera's Avalon bus [69]; selected for HLS LegUp-system, documentation, and mature development ecosystem [63].

The MIPS processor has a 5-stage pipeline based on the common RISC subset that shown in **Fig 4.5**, and are very similar with LEON; however, Decode / Register Fetch are at the same stage and without the Exception stage. A typical MIPS core has 32-bit GPRs for *Integer-Registers (IRs)*: Register *r0* always holds 0 and *r1* is assembler temporary, *r31* is usually used for *Jump-and-Link (JAL)* instructions, *HI* (Higher result) and *LO* (Lower result) registers to access the results of integer MUL/DIVs, and multiply-accumulate operations, and a *Program-Counter (PC)* Register. Most MIPS cores have caches, but they are not implemented in the same configuration, with write-back or write through I&D-caches [70].

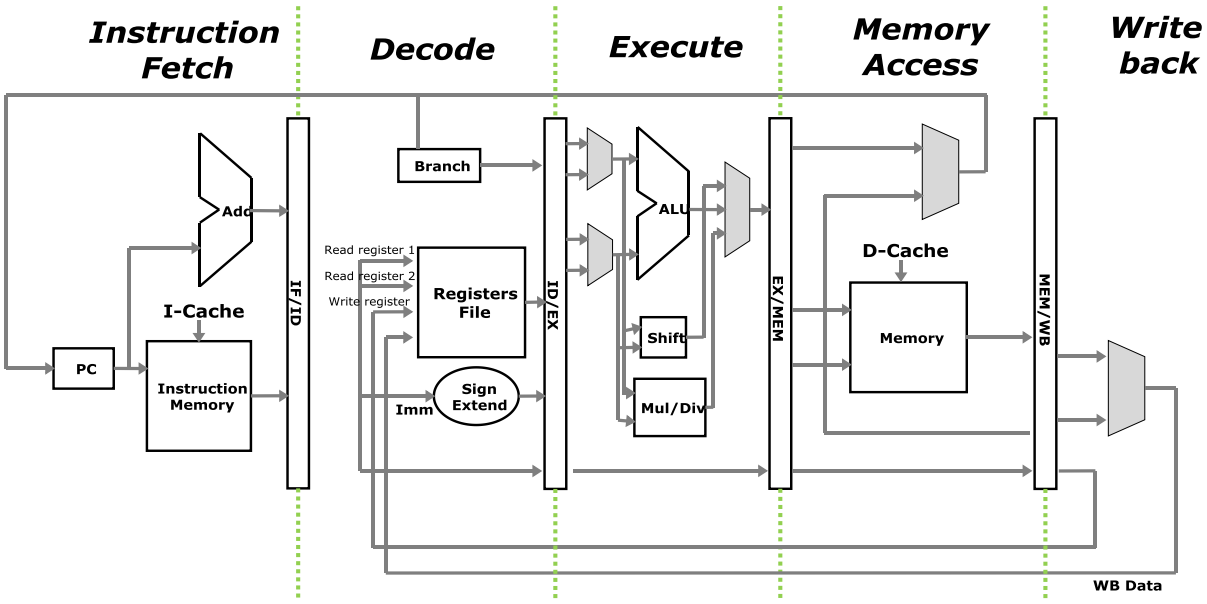

Figure 4.5 MIPS CPU core datapath [71].

To the default Tiger-MIPS core, ***32 32-bit Integer GPRs***, and the number of blocks in caches are 2^9 = 512 lines (bytes) = ***16KB*** for ***I&D-caches*** size of the "***Direct-mapped***" associativity is included.

4.3 The Software (SW) Flow

The compiler transforms C source code into the "computer assembly language" that often forms a binary also known "object code" to be executable. Different computer architectures have different computer languages; therefore compilers for different processors are typically unique.

There are identical compilers for these three processors: Kernels for SPARC V8 are used with LEON3, Aeroflex Gaisler provides *Bare-C Cross (BCC)* compiler system, also known as the "SPARC-GCC" compiler. It consists of the GNU GCC C/C++ compiler and the Newlib standalone C-library [72]. Hewlett-Packard (HP) provides *VLIW-Example (VEX)* Toolchain, which contains compilation-simulator to target VLIW processors [73]. The VEX assembler takes as input the textual assembly output from the compiler, flattens the code, resolves branch / jump targets and produces two output files, the instruction binary and the initialized data section. The front-end of *Clang* compiler is C/C++ to generate the software binary for the soft processor. However, alternative compilers may affect application computational speed of the model simulations as well as the optimization flags used during compilation.

4.3.1 Experimental Implementations

For the stimulations of the LEON3, the following C programs were tested on the target FPGA board by the LEON3 *GRMON* debug monitor. Assigning a function of "clock()" to the end of the programs, it would return a time spent of the "main()" in microseconds (μs). For the LE1 stimulations, *INSIZZLE* is a cycle-accurate simulator and returns the clock cycle count, calling the single data memory block and the results displayed in the final simulator C-source files [68]. For the Tiger-MIPS stimulations, the *SPIM* simulator, the counter variable gave the total number of cycles for a complete execution and returned the total time spent on the programs in picoseconds (*ps*). For the FPGA synthesis of LEON3 and LE1 cores, I used LEON3 / GRLIB IP Library design tool and LE1 Toolchain to target the Virtex-6 LX240T (ML605) FPGA (see **Section 5.2.3** and **5.2.4**); and the Tiger-MIPS core, I used *LegUp-3.0* and *Quartus II 12.0* to target the Stratix IV EP4 FPGA and the clock period constraint would set to the default value of "5 ns" on Stratix IV series.

4.4 Benchmark Collections

This section represented the overall benchmark suits that execute in LEON3&LE1&Tiger-MIPS processors to this experimental work. These benchmark collections were chosen from a wide-range of C-based algorithms and would be used in the following case studies of performance evaluations in this chapter and further in chapter 5 and 6. The following codes were separated into three divisions of "data-intensive", "function-intensive", and "other" programs. However, the LE1 processor does not support "SoftFloat" functions and "little endian"; therefore, I compared all programs and without "double-precision FPU algorithmic" and the "Motion" codes. Furthermore, all of the assembled programs were optimized to the highest performance level to reduce the cost of compilation.

4.4.1 Data-intensive Programs

These C/C++ programs include two examples of the DLP methodology by CM and LZW. They are data-intensive processing of large data volumes and suitable for multi-core processor systems. CM is a popular program in video / image processing and LZW is a universal lossless data compression algorithm that widely used in PDF files in Adobe Acrobat. The detail of software implementations for CM and LZW is presented in **Section 5.4**, and simplified description of programs is listed in **Table 4.1**.

Program	Design Description	Source
CM60x60	Convolution Matrix with 60x60 output data	
LZW45K	Lempel Ziv Welch with size of 45KB input data	LZ78 [74]

Table 4.1 Outline of the DLP benchmark suits.

4.4.2 Function-intensive Programs

The following C/C++ *CHStone* programs are for the HLS methodology and represented in **Table 4.2**. They are function-intensive processing and available from LegUp example directory that facilitates C to HDL for Co-design partitions. However, these programs are "not" suitable for parallel computing in multi-core processor implementations. It consists of 12 programs which are included four arithmetic programs, four media applications, three cryptography programs, and one processor. More details of software implementations for CHStone is presented in **Section 6.4**. The simplified information for each program is as follows below:

Program	Design Description	Source
MIPS	Simplified MIPS processor	CHStone group
DFADD	Double-precision floating-point addition	SoftFloat [75]
DFDIV	Double-precision floating-point division	SoftFloat [75]
DFMUL	Double-precision floating-point multiplication	SoftFloat [75]
DFSIN	Sine function for double-precision floating-point numbers	CHStone group, SoftFloat [75]
ADPCM	Adaptive differential pulse code modulation decoder and encoder	SNU [76]
GSM	Linear predictive coding analysis of global system for mobile communications	MediaBench [77]
JPEG	JPEG image decompression	The Portable VideoResearch Group, CHStone group [78]
MOTION	Motion vector decoding of the MPEG-2	MediaBench [77]
AES	Advanced encryption standard	AILab [79]
BLOWFISH	Data encryption standard	MiBench [80]
SHA	Secure hash algorithm	MiBench[80]

Table 4.2 Outline of the CHStone HLS benchmark suits.

4.4.3 Other Programs

The following several C/C++ small test programs, which given by LegUp example C source codes are represented in **Table 4.3**. However, larger volume applications such as follows: The "*OGG*" is the largest codes of this section that contains format only. "Containers" may include streams encoded with multiple codes of video and audio data and can be in various formats (eg. MPEG-4 and MP3). The "*FFT*" is a well known algorithm to compute the DFT rapidly in audio parts. The "*FIR*" is a moving average filter that widely used in signal processing. The information about each program is as follows below:

Program	Design Description	Source
ARRAY	Multi-dimensional arrays	LegUp [37]
DHRYSTONE	Tests a system's integer performance, and no operating system call	LegUp [37]
DIVCONST	Test division algorithms	LegUp [37]
FFT	Fixed point 16-bit input-output in place Fast Fourier Transform	LegUp [37]
FIR	Finite-Impulse-Response (FIR) filter stream with 32 inputs	LegUp [37]
FUNCTION POINTER	Demonstrate a function pointer	LegUp [37]
FUNCTIONS	Testing multiple functions	LegUp [37]
HIERARCHY TEST	Testing addition algorithms in functions	LegUp [37]
LLIST	Testing a linked list node struct with pointers	LegUp [37]
LOADSTORE	Print out “Hello world”	LegUp [37]
LOOP	Simple loop with an array	LegUp [37]
LOOPBUG	Test Double-precision floating-point addition in for loop	LegUp [37]
MALLOC	Allocate a block of size bytes of memory and return a pointer to the beginning of the block	LegUp [37]
MEMORY ACCESS TEST	Performs read/write memory accesses from/to local and global memory	LegUp [37]
MEMSET	Testing the conversion of llvm.memset and llvm.memcpy intrinsics into aligned calls of our own functions	LegUp [37]
OGG	Is designed to provide for efficient streaming and manipulation of high quality digital multimedia.	LegUp [37]
SELECT	Testing ? : statement	LegUp [37]
SHIFT	Left-shifting an integer by more than 32 bits	LegUp [37]
SIGNDDIV	Testing division operands for sign numbers	LegUp [37]
STRUCT	Testing structs with nested arrays and structs. Functionality includes loads, stores, global and zero initialization.	LegUp [37]
UNALIGNED	Testing unaligned memory access.	LegUp [37]
SWITCHES	Testing switch-case statement	LegUp [37]
SRA	Algorithm of square-root approximate to integer	LegUp [37]
TIGER SRA	Algorithm of square-root approximate to integer using function to define equations.	LegUp [37]

Table 4.3 Outline of the other benchmarks.

4.5 Evaluations of Speed&Area in Single-core of Soft-processors

The total of ***"33" benchmarks*** were executed in each of examining processors. The executed cycle counts as well as the execution times from each µPs were displayed in **Table I.A**, **Appendix I**; and area breakdown and comparisons of LEON3&LE1 processors on the *Virtex-6 LX240T (ML605)* FPGA and the Tiger-MIPS processor on the *Stratix IV EP4* FPGA would be discussed afterwards.

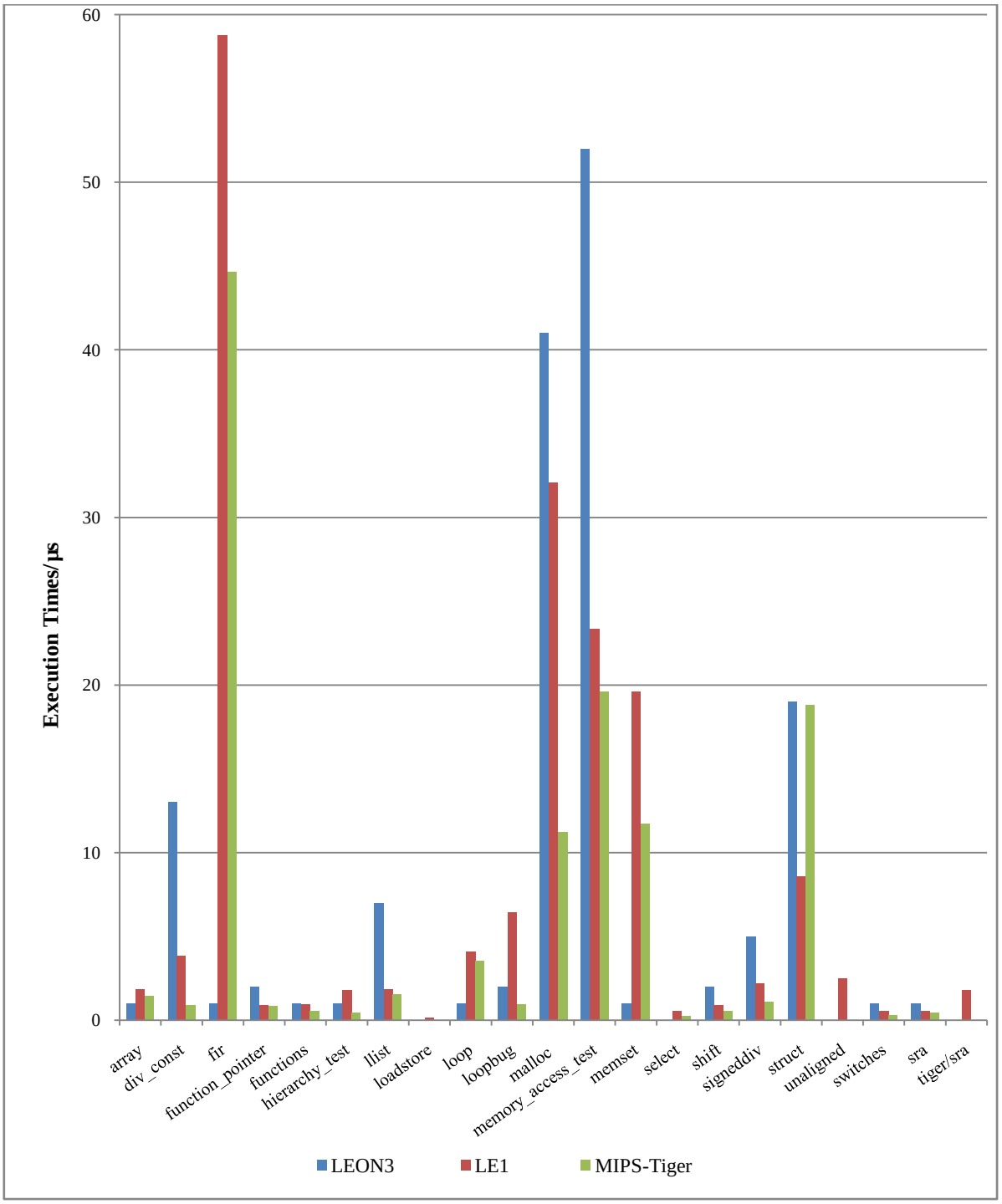

Figure 4.6 Comparsion in execution times at Low-end Benchmarks.

The system frequency on FPGAs of the LEON3, LE1, and Tiger-MIPS cores were clocking at *75*, *75*, and *74.26* MHz respectively to obtain the duplicated performance. The 33 benchmark results were separated into four execution time domains in order to observe them in a histogram distinctly. There were the first benchmark group of under 800 µs at *Lower-end Benchmarks* (Fig 4.6), the second of 800 ~ 8,000 µs at *Mid-end Benchmarks* (Fig 4.7), the third of 8,000 ~ 120,000 µs at *High-end Benchmarks* (Fig 4.8), and the fourth of over 120,000 µs at *Very High-end Benchmarks* (Fig 4.9) correspond to their own time periods.

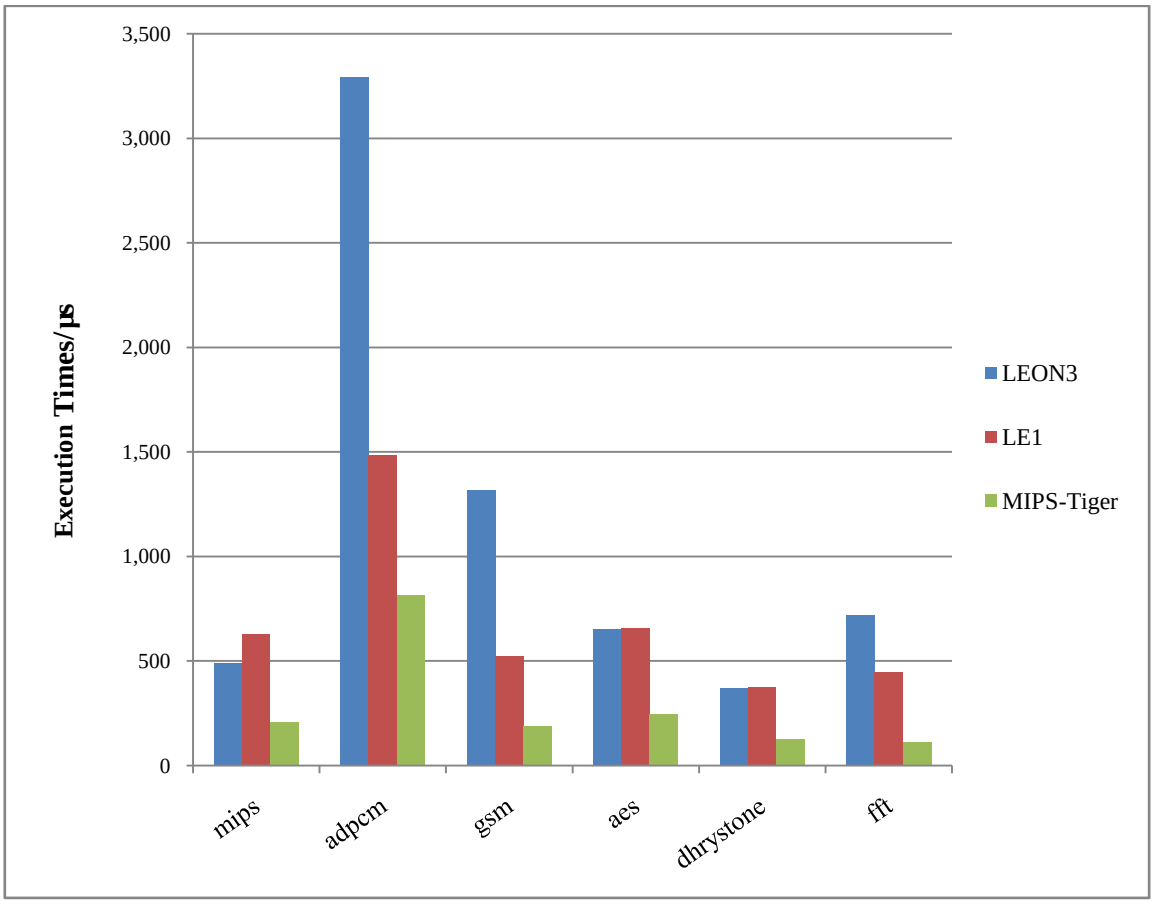

Figure 4.7 Comparsion in execution times at Mid-end Benchmarks.

As the results shown, at Lower-end Benchmarks for basic C algorithms, the Tiger-MIPS processor was powerful all around, comparing with LEON3&LE1. In addition, the LEON3 appeared to be much disadvantaged at MUL/DIV operations such as *DIV_CONST*, and *SIGNEDDIV* benchmarks; and main memory access through load/store from local to global memory such as the *MEMORY_ACCESS_TEST*. Besides, it seemed to be a little disadvantaged at the pointer syntax of the memory location such as *FUNCTION_POINTER* and *LLIST*; the shift operation and "switch-case" statement such as *SHIFT* and *SWITCHES*. However, it was well at addition operations such as *ARRAY*, *HIERARCHY_TEST*, and *LOOPBUG* and more advantageous at dealing with data movement between registers or local memories than Tiger-MIPS&LE1 processors, such as *MEMSET* and *LOOP*. Moreover, the performance of the "structure" statement in the LE1 processor apparent to be fine, comparing with LEON3&Tiger-MIPS, such as *STRUCT*.

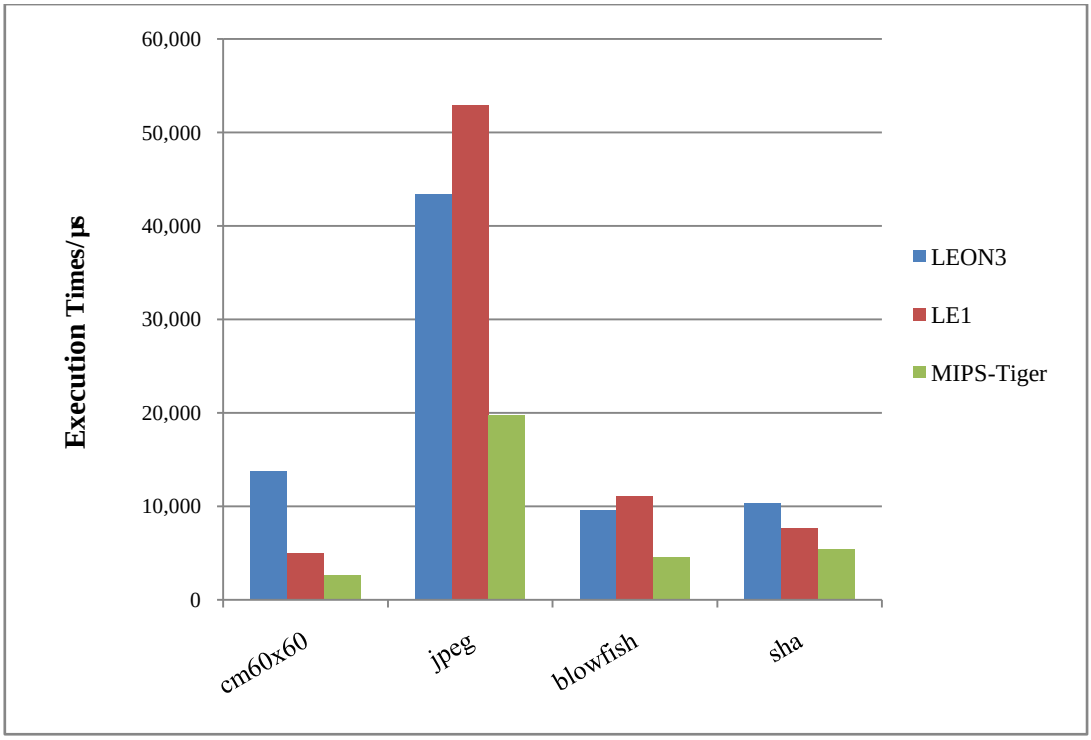

Figure 4.8 Comparsion in execution times at High-end Benchmarks.

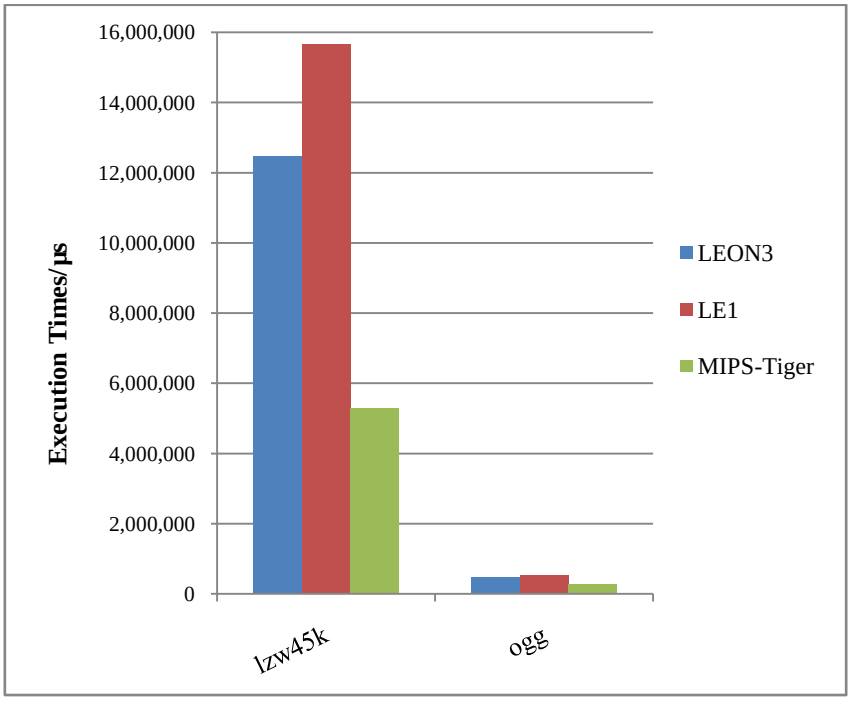

Figure 4.9 Comparsion in execution times at Very High-end Benchmarks.

At Mid-end and High-end Benchmarks for larger codes (see page 121, **Figure III.A**, the percentage of algorithmic types of benchmarks), the lacking performance at MUL/DIV operations of the LEON3 processor have also shown on *ADPCM*, *GSM*, *FFT*, and *CM60x60* benchmarks. The same aspect were also being discovered on *CM60x60* results of DLP implementations (see **Section 5.5.1**). Moreover, for the *LZW45* benchmark, although the LEON3 has greater performance of movivg data in registers for the C string functions (see page 73), however there was many execution times penalty in data accessing from the main memory than the Tiger-MIPS.

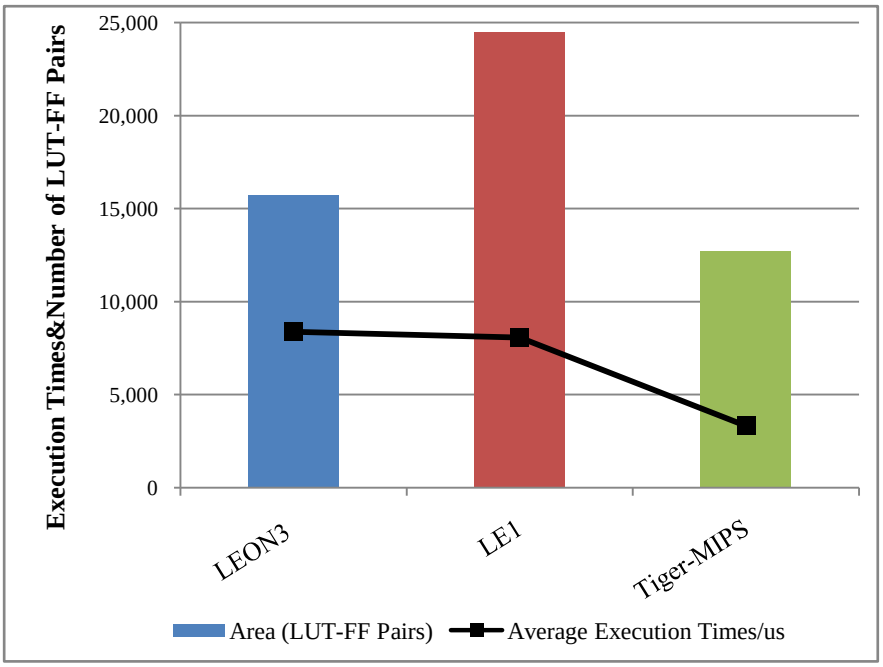

Figure 4.10 The SPEED and area results of LEON3&LE1&MIPS SW flow.

By the LEON3 and LE1 (Xilinx); and the Tiger-MIPS (Altera) processors were breakdown on different FPGA vendors, I applied the resource usage conversion between Xilinx&Altera of the **Equation 3.1**. It was important that running benchmarks on the "Altera DE4" rather than DE2 boards, because that Xilinx Virtex-6 and Stratix IV are the same scale of fabrication process and logic density, therefore it was more precise converting area between them (also see page 45). The Tiger-MIPS processor estimated area breakdown of the number of "LUT-FF Pairs" on the Xilinx Virtex-6 FPGA was calculated out. Corresponding to the cell transformations (see page 45), Altera: 1 ALM = 2 ALUTs and there was 14,135 ALUTs, hence the number of ALMs was 7,068 on the Stratix IV FPGA and *12,722* LUT-FF Pairs on the Xilinx Virtex-6.

Additionally, the average of aggregate execution times in benchmarks gathering was also computed. The wide-spread of the Speed and Area results in the SW flow of LEON3, LE1, and Tiger-MIPS processors were plotted in **Fig 4.10** (Note: Speed - averaging the execution times between Mid and High-end benchmarks). The left

vertical axis have shown geometric mean execution times (10 times of a millisecond) and the right axis shown the area (number of LUT-FF Pairs on Xilinx vendors).

To evaluate the Speed&Area issue between LEON3&LE1&MIPS processors on FPGA-based, I made use of the DSE formulas that have advocated in **Section 3.4.1** and **3.4.2**. Whereas, the equations of the "Program Speed-up" was represented as Average Execution Times of LE1 or Tiger-MIPS divided by Average Execution Times of LEON3; the "Area Cost" was represented as the number of LUT-FF Pairs of LE1 or Tiger-MIPS on Xilinx divided by the number of LUT-FF Pairs of the LEON3 on Xilinx; the "Speed&Area Efficiency" was represented as Program Speed-up of LE1 or Tiger-MIPS divided by Area Cost of the LEON3. While, the LEON3 was the baseline processor to be compared to LE1&Tiger-MIPS; and their program speed-up, area cost, and the Speed&Area Efficiency were always 1. I expressed all the experimental values in two decimal places. The total evaluations have shown in **Table 4.4**.

	LEON3	**LE1**	**Tiger-MIPS**
Average Execution Times/μs	79,017	73,833	35,736
Area (Number of LUT-FF Pairs)	15,728	24,454	~12,722
Program Speed-Up	**1**	**1.04**	**2.52**
Area Cost	**1**	**1.55**	**0.81**
Speed&Area Efficiency	***1***	***0.67***	***3.11***

Table 4.4 The Speed&Area Efficiency of LEON3&LE1&MIPS SW flow.

For the evaluations of LEON3&LE1&MIPS processors: along with the LE1 core, although the execution time were merely *4%* increased, the core has led to the highest area penalty *155%* of a LEON3 core, and the Speed&Area Efficiency of the LE1 core has had much deceased (*0.67*). As far as concerned with the Tiger-MIPS core, the average benchmarks have increased twice and half as much *252%* and was almost *19%* resource usage reduced, and the Speed&Area Efficiency of Tiger-MIPS core has been increased (*3.11*).

To evaluate the "speed" results, it was probably due to the Tiger-MIPS core has had shorter instruction pipeline depths (5-stage), comparing to LEON3 and LE1 (7-stage). In other words, the longer instruction pipeline of CPUs, the longer clock cycle needed to complete a sequential pipelining, and the longer time needed to compute a program. However, increasing CPU instruction a depth would allow higher frequencies for the designs, which mean LEON3 and LE1 processors are ideally enable to be clocked at faster rates. Moreover, the LE1 processor has double instructions implementations within the CPU to form as a superscalar architecture which allows double independent instructions to be executed in parallel per cycle. In this process, CPU hardware would have to check the available resource (register, memory, and FUs) for multiple instructions running simultaneously. However, the speed-up of a superscalar CPU would not be directly proportional to the numbers of the instruction throughput. The average benchmark result was disappointed and approximately or exactly a little bit

better than the LEON3 core. It was due to parallel executions of superscalar pipelined would have limited by resource conflicts and data dependencies (also see **Section 5.2.1**). These could be improved by increasing the numbers of IALU&IMULTs and CLUSTERs within LE1 microarchitectural parameters.

To evaluate the "area" results, the size of a LEON3 core was very close to the Tiger-MIPS. It was however with debug units and some communication ports which more configurations than the Tiger-MIPS. A LE1 core (nearly the Tiger-MIPS core size and *80%* size of a LEON3) needed to be operated with a service MicroBlaze SoC (11,880 LUT-FF Pairs). Moreover, increasing the number of CLUSTERs to meet higher ISSUE_WIDTH or increasing the number of IALU&IMULT to lower the latency, would have increased the resource significantly.

4.6 Summary

This chapter demonstrates a wide-range of ANSI C-based programs running in different configuration of LEON3, LE1, and Tiger-MIPS soft processors and they are synthesized on different target FPGAs. Fortunately, a formula for converting the FPGA resource between Altera ALMs and Xilinx LUT-FF Pairs that provides Xilinx&Altera device logic comparison approximately. The LEON3 processor is the baseline comparison for another two of the "Speed&Area Efficiency". To summarize the results, it shows that the Tiger-MIPS processor has the fastest execution time and the smallest area cost; and has won the whole tradeoffs than LEON3 and LE1 processors. However, it does not mean that LEON3 and LE1 processors are worse than the Tiger-MIPS at entire prerequisite. The merits of the LEON3 and LE1 cores such that are highly configurable in customization. Moreover, the design tools are able to support multi-core processor system for parallel computing and very flexible on target FPGA boards. To the next chapter, I will examine and compare the Parallel-SW flow benchmarks in the case of the DLP methodology on LEON3&LE1 multi-core processor systems.

Chapter 5

Evaluation in Data-level-parallelism of LEON3 MP and LE1 CMP on FPGA-based SMPs

5.1 Introduction

A number of homogenous multi-core processors can be implemented on the FPGA-based platform and perform a parallelism computation. This chapter presents the "Parallel-SW" flows of CM image processing convolution and LZW compression / decompression algorithm executing in LEON3 MP and LE1 CMP system of the DLP methodology (also see the description in **Section 3.2**). In contrast with the single-core, the multi-core system provides higher performance at lower frequency in such parallel programming within FPGAs. In addition, for the set up of the number of CPUs in this study, the available processors in the system varied from 1 up to 6 that synthesized on the Virtex-6 LX240T (ML605) that contains 150,720 LUTs-Flip Flop pairs and maximum block RAMs of 3,600 KB that can be fitted and reused for the designs [81]. Furthermore, it is necessary to evaluate and compare the Speed&Area tradeoffs of these SMPs onto the same FPGA device.

5.2 LEON3 and LE1 Multi-core Processor Methods

The conventional RISC instruction in which a single-core is an example of SISD stream and in multi-core system can realize MIMD process. However, a VLIW processor with longer instructions (ILP implementations) in which a uni-processor and connect multiple cores in a system for parallel programming (TLP or DLP) is also an example of MIMD stream (also see page 27). In these processor-system implementations, computing types are all identical to SPMD which is the sub-type of the DLP process for multiple core systems of LEON3 (RISC) and LE1 (VLIW) processors.

The SMP system is the most popular framework of the multi-core processors for parallel computing that can be classified into task or data level parallelism. For the novel comparison, LEON3 and LE1 soft-cores have very similar instruction pipeline depths. However, a LE1 core designed has the advantage of ILP optimizations and highly paramertisable architectural FUs, such as ALU / MUL units to meet the critical computation for the performance, and the LEON3 MP has the advantage of wealthy reusable IPs (GRLIB IP library) suitable for complex MPSoC designs and a versatile cache-system to be implemented.

5.2.1 RISC vs VLIW

RISC instructions are simple operations based on load/store architectures (register-to-memory), and it is the only operation that affects the memory. Arithmetic operations between registers (register-to-register) and requires a number of GPRs then to execute in ALU / MUL units. VLIW operations, which concerned with multiple issued of RISCs, a powerful compiler is needed to schedule instruction packets in parallel and decide when they are compiled [82], therefore, the numbers of GPRs and FUs of VLIWs are larger in order to satisfy multiple operations that speed up programs. An ideal case of VLIW operations without any resource and data or control dependency conflicts compared with RISCs as shown in **Fig 5.1**, "double semicolon" means the clock cycle and multiple instructions are executed concurrently. The blank slots (group of four bytes) are ALU / Branch or NOP instructions.

RISC

```
Br    r4 <- 0
ld    r5 <- 4[r1]
ld    r6 <- 8[r2]
ld    r7 <- 12[r3]
add   r5 <- r6 + r7
st    4[r1] <- r5
```

```
if(i==0){
a = b + c;
}
```

Br	r4	0	
ld	r5	r1	4
ld	r6	r2	8
ld	r7	r3	12
add	r5	r6	r7
st	r5	r1	4

VLIW (2-Width)

```
Br    r4 <- 0
ld    r5 <- 4[r1]
;;
ld    r6 <- 8[r2]
ld    r7 <- 12[r3]
;;
add   r5 <- r6 + r7
;;
st    4[r1] <- r5
```

Br	r4	0		ld	r5	r1	4
ld	r6	r2	8	ld	r7	r3	12
add	r5	r6	r7	-	-	-	-
st	r5	r1	4	-	-	-	-

Figure 5.1 Example of RISC and VLIW instructions.

5.2.2 Symmetric-Multi-Processors (SMPs) with Data-Level-Parallelism (DLP)

The diagram of the SMP system is shown in **Fig 5.2**, a number of homogenous cores are connected by a system bus or crossbar switches, peripherals, and where all processors access a common memory symmetrically (also see page 37). Processors are identified by a unique ID and ready for software implementations. However, accessing to the main memory through a shared bus does not fully scale with the number of processors and result "data overhand" (also see page 38).

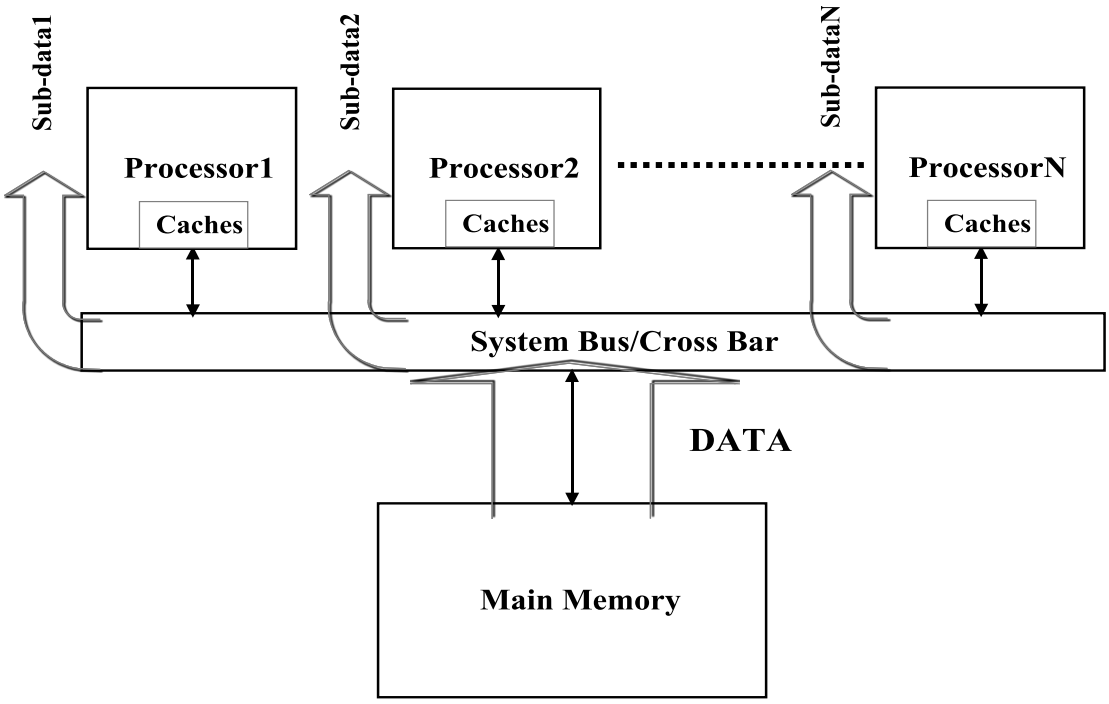

Figure 5.2 SMPs with DLP.

Typically, the disadvantage of data synchronization effect can be lowered by employing local caches in each processor to consistency load/store data from the core; thus to speed-up data access from the main memory and also reduce the bus traffic of the SMP system when the program in parallelism. That is known as the "*Cache Coherence*" implementation. Moreover, to generalize, there are three main factors affected the scalability of SMPs: cache coherency pipelining, time spend in spin lock (number of cycles) by programming, and memory accessed conflict. Most of the SMP computers typically have ten or fewer cores. For embedded processor-systems designed, the best performance of the number of processors in the UMA SMP is limited to 8 cores [83, 84].

5.2.3 LEON3 MP and IPs Configuration

To minimize the LEON3 MP System suitable for this design, many of the IPs (eg. Ethernet MAC consumes more than 10,000 LUTs) were disabled to save FPGA resources. Each LEON3 core configuration of the LEON3 MP is the same as in the single-core benchmarks (Section 4.2.1). For the local memory configuration, the ***I-RAM*** size is ***64KB*** and ***D-RAM*** is ***256***, ***128***, ***64***, ***64***, ***64***, and ***32KB*** respectively from 1 to 6 cores, in order to align the shared D-RAM of the LE1 CMP. To summarize the LEON3 MP components, **Table 5.1** shows the key features of LEON3 configurable blocks that are important in this MPSoC platform.

Functionality Blocks	Description
IU	Full SPARC V8 integer unit, 7 stages pipeline interfaced with I&D caches sub-system, including MUL/DIV instructions, and the numbers of register windows is configurable.
Cache System	Various setting for separated I&D caches with 1-4 sets, 1-256kbyte/set, 16 or 32 bytes per line (*Total cache size = sets*set size*). The D caches uses *write-though* policy interfaced with AMBA/AHB bus and also perform *bus-*
On-chip DSU	It is a type of non-intrusive debugger to target FPGA.
Interrupt	Interrupt interface (up to 15 asynchronous interrupts).
AMBA	ARM AMBA-2.0 standard, separated with AHB and APB bus, CPUs execute load /store data to/from the caches.
LEON3 Cores	Number of LEON3 cores. Up to 16 cores (masters) can be build.

Table 5.1 LEON3 blocks CONFIGURABILITY [64].

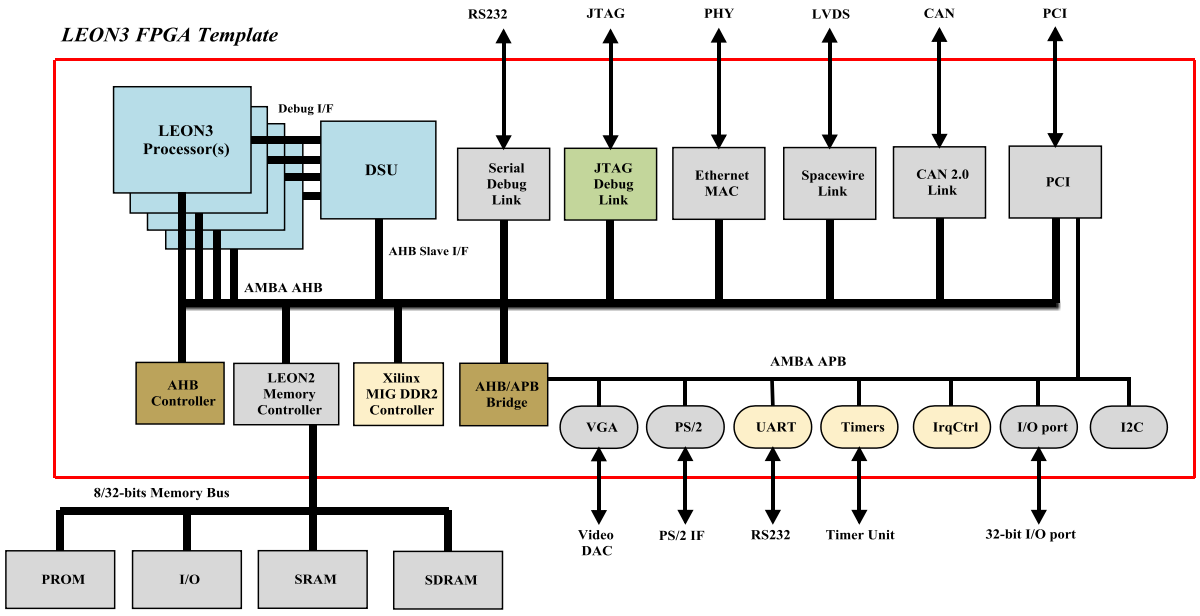

Figure 5.3 LEON3 multi-core with GRLIB IP for the SoC [85].

The gray colour items show the disabled IPs in the LEON3 FPGA template, as showing in **Fig 5.3,** based on the *Advanced-Microcontroller-Bus-Architecture (AMBA)* on-chip buses, a bridge link between *Advanced-High performance-Bus (AHB)* and *Advanced-Peripheral-Bus (APB)*. The *Xilinx MIG DDR2* is the global memory controller of the system, and it is the same for the LE1 CMP on the FPGA.

For ***AMBA-2.0*** configuration, all the LEON3 processors and many of the IPs such as ***JTAG Debug Link*** and PCI are master connected to AHB bus. The ***round-robin*** policy is supported; and JTAG interfaced for the GRMON debug monitor that is connected to the AHB. In addition, an APB bus is slave to the master AHB bus, and interconnects to the APB, ***Timers*** and 8 ***bytes (FIFOs)*** for ***UART*** console in order to return fast printout results on the terminal of the computer screen. The ***interrupt controller*** is designed to be used in LEON3 multi-processor systems and explained in **Section 5.3.1**.

5.2.4 LE1 CMP and Configurability

Table 5.2 presents the parameters of the LE1 which are important for this experiment and more information about frontal definitions of the single-core (see page 51 - 54). The LE1 CMP customizations are available in order to exploit TLP (named "DLP" instead in this case study) while being able to closely match the performance and area requirements of target applications and architectures. As well as the LEON3 core, LE1 CPU configurations of its CMP are organized to previous setup in **Section 4.2.2**.

Architectural	Description
ISSUE WIDTH	Architectural width (VLIW) of the processor. This is the number of RISC operations (syllables) dispatched every on clock.
CLUSTERS	Number of clusters of the LE1 CPU. The LE1 is a multi-cluster architecture with each cluster having its own set of architectural registers and execution resources.
IALUs	Number of integer ALUs per processor.
IMULTs	Number of integer multipliers per processor.
I-RAM SIZE	Size of closely-coupled instruction (code) RAMs. The application code is loaded in this memory prior to execution.
D-RAM SIZE	Size of closely-coupled data RAMs. The Initialised data segment is loaded in this memory prior to execution. Serves as the stack area for all active hardware threads.
D-RAM BANKS	Number of banks of the Data RAM. Accesses to disjoint banks incur no cycle penalty.
LSU CHANNELS	Number of channels to the Data RAM per processor.
LE1 PROCS	Number of LE1 contexts (cores) in the multi-processor (CMP) configuration.

Table 5.2 LE1 microarchitectural configurability [68].

A shared-memory CMP (2-core), as shown in Fig 5.4, can be instantiated. These are achieved by altering the ***LE1 PROCs*** parameter shown in Table 5.2, populate multiple contexts of the LE1 CORE and are enabled to fit up to 6 cores into the target FPGA device. The figure depicts a dual-LE1 system with separate instruction RAM and a common, banked D-RAM. The instantiated ***local I-RAM*** and a ***global D-RAM*** size were established to ***64KB*** and ***256KB*** respectively. In this study, I will investigate TLP by altering the number of instances of the LE1. Both of these parameters are modified within a top level XML machine model and used throughout the LE1 tool flow to specify these micro-architectural configurations.

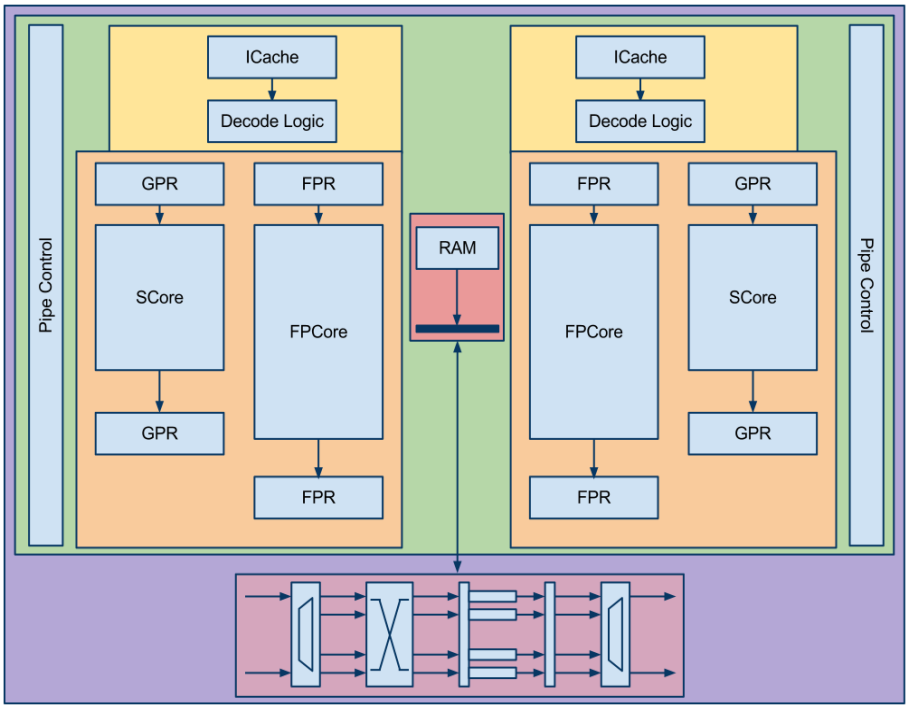

Figure 5.4 Two-way multi-processor consisting of two instances of a 4-wide, single-cluster LE1 core, the common data memory and the thread control unit [69].

5.3 The Parallel-SW Flow

The ideals of the Parallel-SW flow in this experimental used are defined in **Section 3.2** (see page 40). For this DLP methodology, both of the LEON3 MP and LE1 CMP are determined to SPMD multiprocessing. Each executed segments of the program is acquainted with the value of "CPUID" by each processor. In addition, the overall system clock of the Parallel-SW flow within FPGA is significant impact on the performance, since processors of the multi-core would occupy large numbers of the FPGA fabric and all soft-core design tools on FPGAs are automatically RAR the fabric each other. Therefore, it may cause the longer wiring delay and slow down the maximum frequency.

5.3.1 LEON3 CPUs Identification

In the multi-core processor system, there is a unique identity for each processor. The processor configuration register (%ASR17) supplies a unique index for each LEON3 core and also provides various other configuration options. This register can be accessed from application through an inline assembly instruction, which is read and masked to retrieve its unique CPUID. Initially, CPU0 is active only, this is then required to start all other processors in the system. This is achieved by writing to the *Interrupt-Controller-Register (IRQ)* to enable other processors. An example of this code is also shown in **Fig 5.5**. To implement this method, the stack pointer also needs to be modified for multiple program executing in a multi-core system by creating an 1MB stack offset in terms of the number of CPUs [MemorySize - (*CPUID* * - 1M)]. This allows each processor a 1MB stack to perform computation. Another aspect which required implementing was fixed using the CPUID to only setup global memory if CPUID = 0.

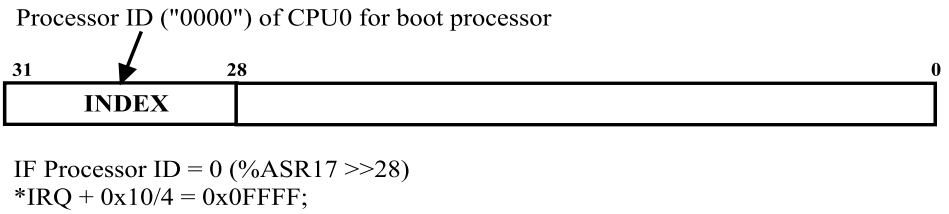

Figure 5.5 LEON configuration register (%ASR17) [65].

5.3.2 LE1 Contexts Indentify

Similar to the LEON3 MP, the LE1 CMP includes a custom instruction which can be called from the application level to return internal registers within the system. In this study the custom instruction returns a unique identifier of the current LE1 context (core). This is used to allow each available context to execute a common instruction RAM and use its unique identifier to perform certain tasks which are specified by the application being run.

5.3.3 Structures of LEON3 MP & LE1 CMP and System Clock on the FPGA

The LEON3 MP is a substantial multi-core processor system and interfaced with a 32-bit AMBA AHB /APB bus. The AMBA clock is synchronized with the system clock. An AHB/APB bridge is used to connect two buses clocked by synchronous clocks with the same frequency ratio and the rest of the configuration IPs. **Fig 5.6** shows the block diagram of the LEON3 MP. The options of the system clock frequency of the LEON3 MP in the Xconfig GUI design tool are 60, 75, 80, 100, and 120MHz.

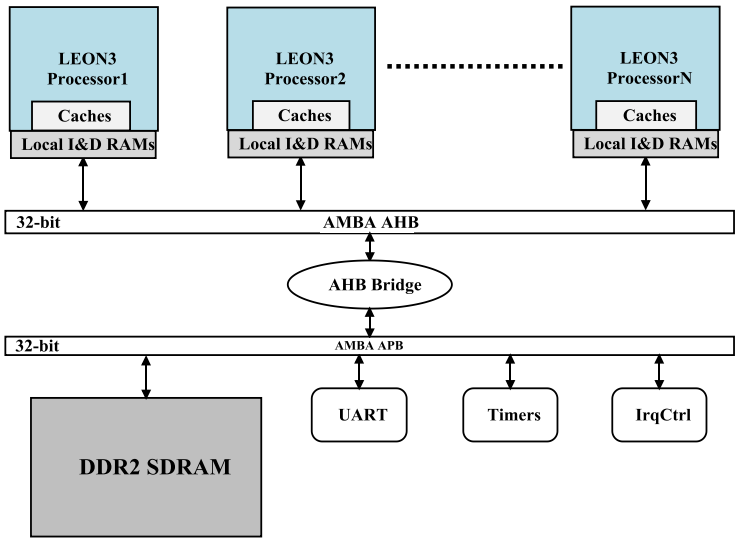

Figure 5.6 LEON3 MP blocks diagram.

However, the LE1 CMP (Fig 5.7) works as a coprocessor with the main CPU (MicroBlaze soft-core). The main processor is utilized to instruct the sub-processors to execute the actual computing instructions which the programmed are desired. The secondary PLB of the LE1 CMP is interfaced with the primary PLB of the MicroBlaze processor with a PLBV46 bridge [86]. The clock frequency ratio between primary PLB and secondary PLB of the LE1 CMP system are 2:1. Thereby, in this experimental implementation, the MicroBlaze is clocking at 100MHz and the LE1 cores are at 50MHz. To obtain the approximate performance comparison with the LEON3 MP by selecting the processor clock at 60MHz in Xconfig tool.

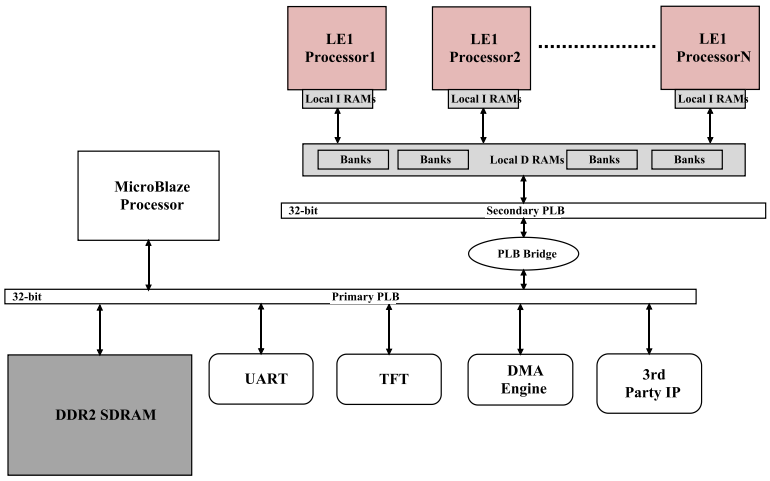

Figure 5.7 LE1 CMP blocks diagram.

5.4 Benchmarks of Data-Level-Parallelism (DLP)

Two applications were created to exploit the multi-core processor systems which are CM and LZW multimedia programs. The SPMD processing was chosen where each CPU/Contexts execute the same process in separate data items. Both of the LEON3 MP and LE1 CMP systems being targeted are bare-bone, they have no operating systems / task managers performing housekeeping. However, CPUs / Contexts have the ability to retrieve a unique identifier which was used to target specific data sections. Moreover, both of the programs were compiled to the best performance level to reduce code size and execution time.

5.4.1 Convolution-Matrix (CM)

In this DLP example, I introduced a method for image processing. Among the frequently algorithms domain, the convolution matrix is the most common way to make an image processing filter work. For a 2-D convolution example, it has a 3x3 image of filter and a 5x5 image of the input [87], as shown in **Fig 5.8**. Take a 3x3 block of source pixels (*Kernel*) and turn it into a single output pixel by multiplying each of its corresponded matrix elements and finally sum them up. Consequently, the rest of the 3x3 output pixels were achieved by the kernel scanning from left to right (rows) and top to bottom (columns).

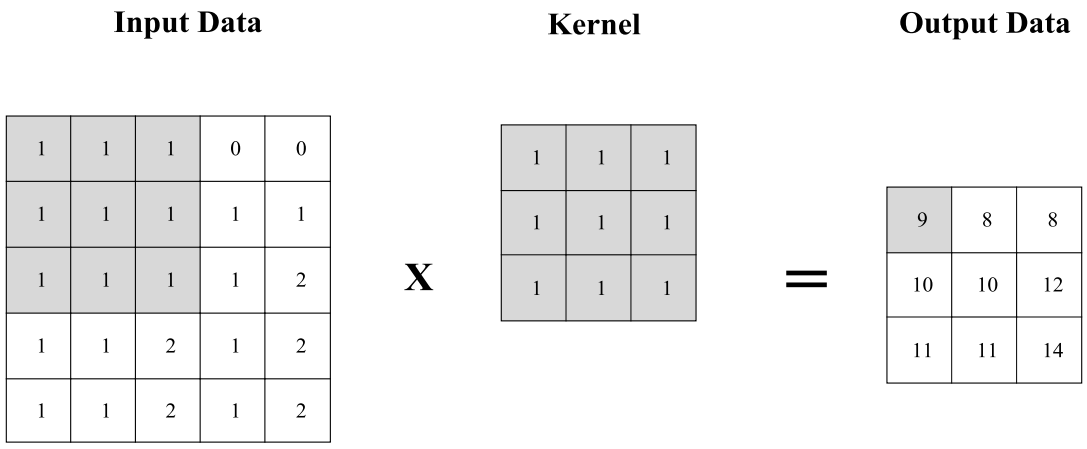

Figure 5.8 An example of convolution matrix.

To create the standard C properly for this destination pixel, I fixed a kernel size of *N* x *N* pixels (N = kernel width) and a variable rectangular *M* x *M* input image (M = input width) to obtain an *(M-2)* x *(M-2)* output image, as shown in **Fig 5.9**. In this study case, I performed a convolution of 3x3 kernel (multiplied the corresponding value of each of them from left to right and top to bottom) at a single item in an input by doubling “for" loops in a function called "*compute*". The first loop across the “*x*” dimension (rows) and the second loop across the “*y*” dimension (columns) to do the multiplicand; then added them together eventually. In the "main()", the same computational method to generate an output by calling the "compute" function to perform other two “for" loops

through the rows and columns of the output pixels. An input pixels set of 62x62 was applied and across the 3x3 kernel and resulted a 60x60 output pixels.

This method of parallelising the data to be processed is crude and requires the number of processes computing the algorithm to be a factor of the total size. However, in terms of multi-processing this method performs fine without any methods of mutual exclusions or conditional variables which will require extra libraries or operating systems which in terms of impose an overhead in execution time. The output pixels of this case were factored by the numbers of cores of 1 – 6 cores, results the whole data set being executed with no overlaps. Simply separate data segments by dividing the "for" loop of "y" columns for each processor and execute each segment that recognized by the CPUID (also see **Fig 3.1**).

```
/* main programs */
int main(void)
for (x = 1; x < M - 1; x++){ /* loop through (output) rows */
   for (y = 1; y < M - 1; y++) { /* loop through (output) columns*/
      compute;
   }
}

 /* compute function */
int compute(x, y)
 for (x = 1; x < N; x++){ /* loop through (kernel) rows */
   for (y = 1; y < N; y++) { /* loop through (kernel) columns*/
      Sum the 3 x 3 kernel array and result to a single pixel;
   }
}
```

Figure 5.9 Example of C code that an MxM input through the NxN kernel of the CM.

5.4.2 Lempel-Ziv-Welch (LZW)

The LZW [88] compression algorithm was chosen as an application which could be easily split over all available CPU/Contexts. The efficiency of compression when splitting the LZW algorithm across multiple threads is studied in [89]. However, it was not a study to concern the compression technique. An implementation of the LZW algorithm was developed in C where all data items required for performing the compression (such as dictionary and the size) are stored in volatile memory on the stack and within registers. The LZW programs contain many C string functions such as *strlen()*, *strcpy()*, *strcat()*, *strncat()*, and *strcmp()* to initialize and encode/decode the dictionary characters. A global input and output data section were generated for the separate CPU/Contexts to read and write. Each CPU/Contexts compressed a contiguous data area which was dependent on the number of other active CPU/Contexts in any given system and its output was stored in the

global data of confirmation that the algorithm executed correctly. All data items were statically defined at compile time so that there was no issue with memory within the stack for each core being overwritten.

```
/* global variables */
 char *input ;  /* global input */
 int size ; /* size of data set */
 int numCPU; /* CPUs in system */
 short *compr ; /* global compressed output */
 char *output ; /* global output */

 /* local variables */
 int cpuid ; /* CPU identifier */
 int start = (CPUID * (size / numCPU)) ;
 int end = ((CPUID + 1) * ( size / numCPU)) ;

 /* call compressing function */
 compress (input, start, end, compr) ;
 /* call  decompressing  function */
 decompress (compr,  start, end, output) ;
```

Figure 5.10 Example of C code which splits global input array over each active CPU for the LZW.

This sectioning of the data was done using a process shown in **Fig 5.10**. This figure shows both system wide variables as well as those local to each processor. Using the current unique ID (*CPUID*), the size of the input data section (*size*) and the number of processors in the system (*numCPU*) the start and end variables are calculated. These are then passed to the compress function along with pointers to the global input and output arrays. Using these passed variables each processor compresses the data from *(input + start) to *(input + end) and places the result in the *compr* array. A decompression of the data was then performed and placed results in the global output array. Validation of the process was performed by comparing the input and output arrays. Using only compression the resulting would be different in each configuration, this is the result of the compression algorithm and how the dictionary was produced during execution.

As well as the CM data organization, an input data set of 45KB was produced and kept constant across all execution run. This size was chosen as the numbers of cores used in this study (1 – 6) are all factors with no overlaps. Both compression and decompression were checked running on both single process and multiple process modes. The output of compression on the input test data was saved and used to compare the results from running in both systems to confirm correct execution was performed in all cases.

5.5 Evaluation and Comparison of LEON3 MP and LE1 CMP

Those applications introduced previously was executed in both of multi-core processor systems. The numbers of CPU/Contexts available in the system were altered from 1 up to 6 (the numbers of instructions were altered from 1 up to 12). The results (***"2" benchmarks***) of execution cycles and execution times of Convolution-Matrix 60x60 input dataset (*CM60x60*) and Lempel-Ziv-Welch size of 45KB input data (*LZW45K*); and resource breakdown of LEON3 MP & LE1 CMP along with the percentage on the whole *Xilinx-Virtex 6 ML605* FPGA fabric (150,720 LUT-FF Pairs and 14,976KB Total Block RAMs) were reported.

5.5.1 Analysis of Performance and Area

The total results of Parallel-SW flow in the LEON3 MP and the LE1 CMP were presented in **Appendix II**. The executed cycles and execution times of the *CM60x60* and *LZW45K* from the LEON3 MP and the LE1 CMP were displayed in **Table II.A**. As well as the execution times, the total resource distributions of the whole FPGA in terms of "LUT-FF Pairs (Area)" and "Block RAMs" have shown in **Table II.C**. However, to produce a fair benchmark comparison of this case study, instruction-levels were the base units, the line results in execution times of *CM60x60* and *LZW45K* decreased when the number of 32-bit instruction sets (issue widths) increased in those SMPs as represented in **Fig 5.11** and **Fig 5.12**. As the result presented, execution times of the *CM60x60* in both SMPs were very linearly decreased in the same path when the number of issue widths increased and the LE1 CMP was a little better than the LEON3 MP; however, execution times of the *LZW45K* were not reduced in the same path, the LEON3 MP was approximately *3* times faster than the LE1 CMP along with the number of instruction sets increased.

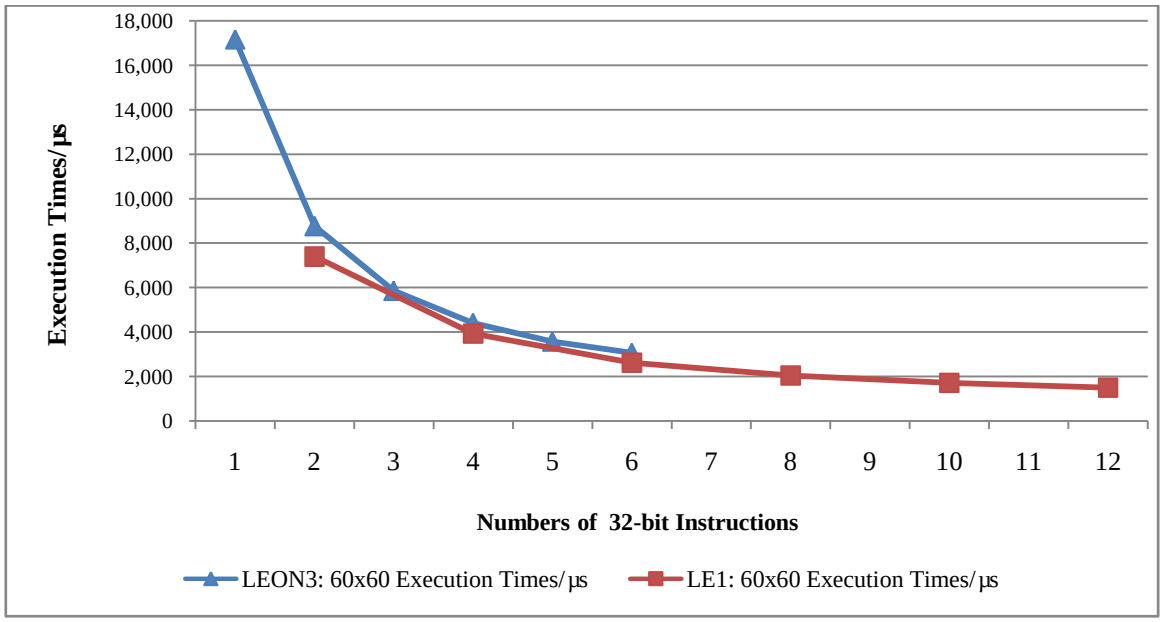

Figure 5.11 Speed-up results of **CM60x60**.

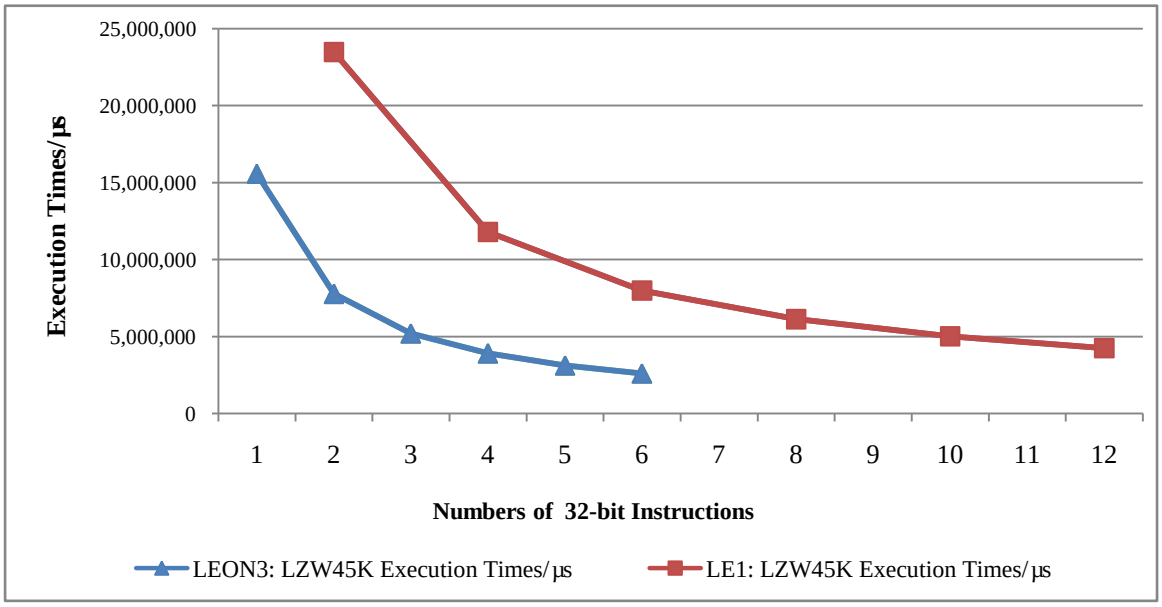

Figure 5.12 Speed-up results of **LZW45K**.

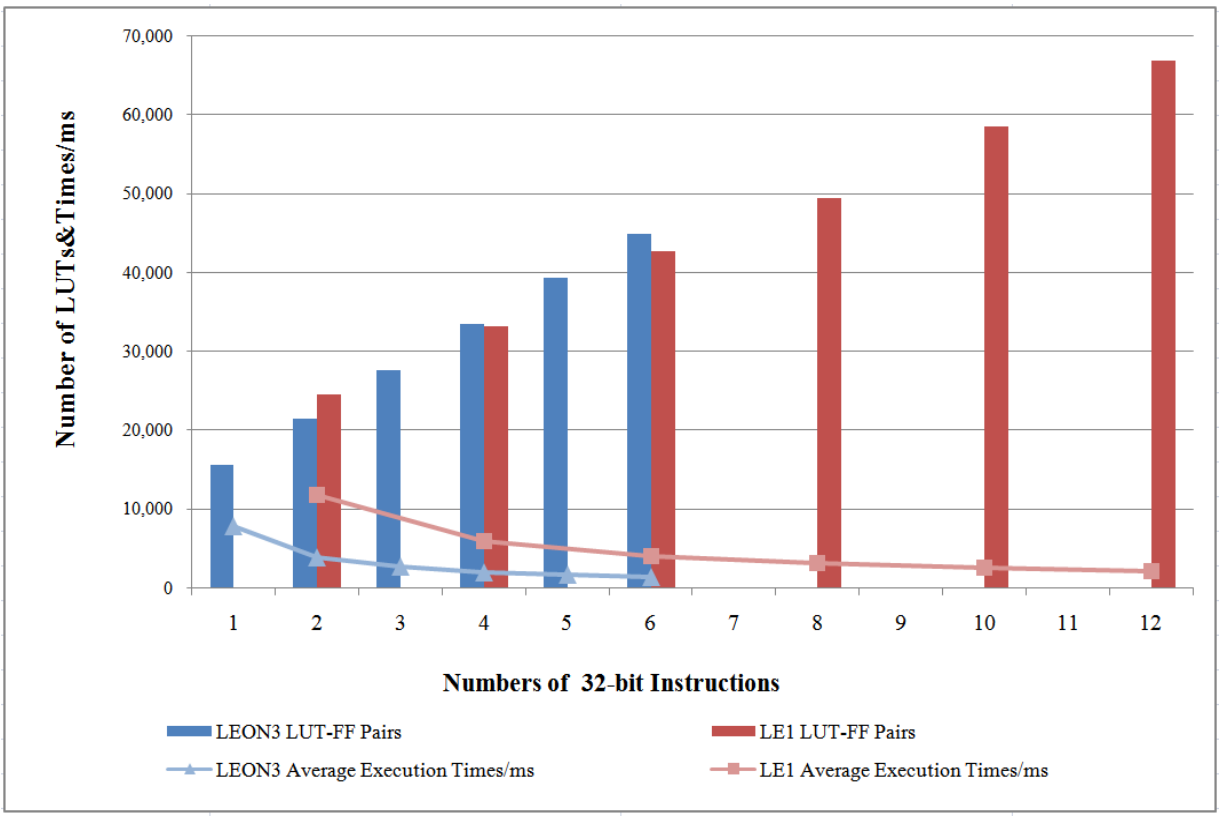

Figure 5.13 The overall performance and area results in LEON3 MP and LE1 CMP.

To summarize the average analysis, the wide-spread of the Speed and Area results of the Parallel-SW flow in LEON3 MP & LE1 CMP were plotted in **Fig 5.13**. In this case of soft-cores evaluations on the FPGA, I concerned only about LUTs and block RAMs for the area. Since, resource utilization on DSP48E1 could be ignored; both of SMPs were less 4% breakdown on DSP units (see **Table II.B**). As the result following by the number of the instructions (comparing the range of 2 - 6 issue widths); in terms of resource utilizations (LUT-FF Pairs), the average FPGA fabric usage of the LEON3 MP was approximately to the LE1 CMP along with the number of issue widths increased; in terms of program speed-up by the number of the instructions/CPUs, it presented that average execution times of the LEON3 MP were averagely *3.03* times faster than the LE1 CMP along with the number of issue widths increased.

Moreover, the number of resource utilizations might affect the system clocking on the FPGA. The maximum frequency of the 6-core LE1 CMP was limited to *50* MHz, due to larger resource of the LE1 CMP breakdown on the FPGA, in order to satisfy a longer connection of PAR, and to meet longer timing. In fact, at the same lower frequency (50MHz), the LEON3 MP execution times might be much slower. However, the LEON3 MP was able to clock at a maximum of *80* MHz (smaller area led to shorter timing delay) and average execution times would have been much faster than the LE1 CMP at 50MHz on the Xilinx-Virtex 6 ML605 board.

5.5.2 Distribution of Block RAMs on the FPGA

The resource distributions of RAMB36/18E1 were displayed in **Table II.B**. Both of the LEON3 MP and LE1 CMP have sub-memory system to decrease the time to access memory, but in different architectures. Each LEON3 core has had the separated instruction and data caches (cache coherency) and all the LE1 processors shared a D-RAM memory bank (paralleled memory accesses).

To implement the cache memory in the LEON3 MP, the size of the caches was critical impact of the overall performance; the smaller the cache blocks, the poor execution times it would be acquired, owing to using fewer cache size could reduce the number of data valid / invalidations in the cache line. However, there was the maximum optimization of the total cache size to maintain the best performance (16KB of I&D-cache respectively in this case study). Multi-bank memory system could significantly reduce the probability of conflicts in LE1 CMP. However, these issues were not in this experimental research.

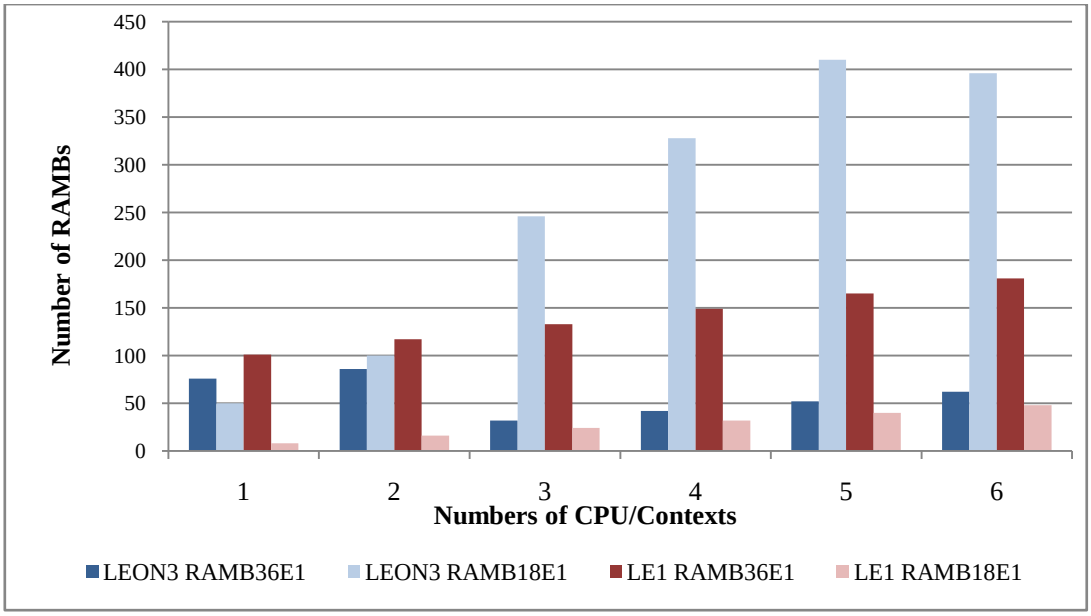

Figure 5.14 The total number of block RAMs in LEON3 MP and LE1 CMP.

At theoretical fulfilment, increasing the memory cache / bank sets, several hundred RAMs will be consumed. The actual number of RAMBs breakdown on the FPGA for both SMP systems has shown in **Fig 5.14**. The basic block RAMs on Virtex-6 FPGAs can be configured as a number of 36KB block RAMs. Each 36KB block RAM (numbers of 416) contains two independently 18KB block RAMs (numbers of 832) [90].

For the observation of memory resource, there has been found that in the figure, showing larger numbers of RAMB18E1 have been used by the cache-system in the LEON3 MP and were increased by adding more cores. However, the number of RAMB36E1 have been significantly decreased at the 3-core and continuously increased up to 6-core. Moreover, both RAMB36/18E1 have increased constantly in the LE1 CMP; on the other hand, the LE1 system was not implemented any caches at all and has had fewer numbers of RAMB18E1 utilizations. For this speculation, perhaps, the cache elements of soft-processors were likely to be implemented on small blocks of RAMB18E1; and local RAMs were chiefly on large blocks of RAMB36E1 and a little amount on RAMB18E1 within Xilinx Virtex-6 FPGAs.

5.5.3 Synchronized Confliction and Speed&Area Tradeoffs

Both of the SMPs have shown the potential disadvantage in contention when multiple core access memory concurrently (data synchronization confliction). The result has shown that as more CPU/Contexts were added, the execution time decreased, this decrease levels off eventually due to memory and bus bandwidths conflicts. These circumstances happened as the program speed-up in multi-core processor system, restricting the parallelism could be achieved that the time needed for the sequential fraction of the program.

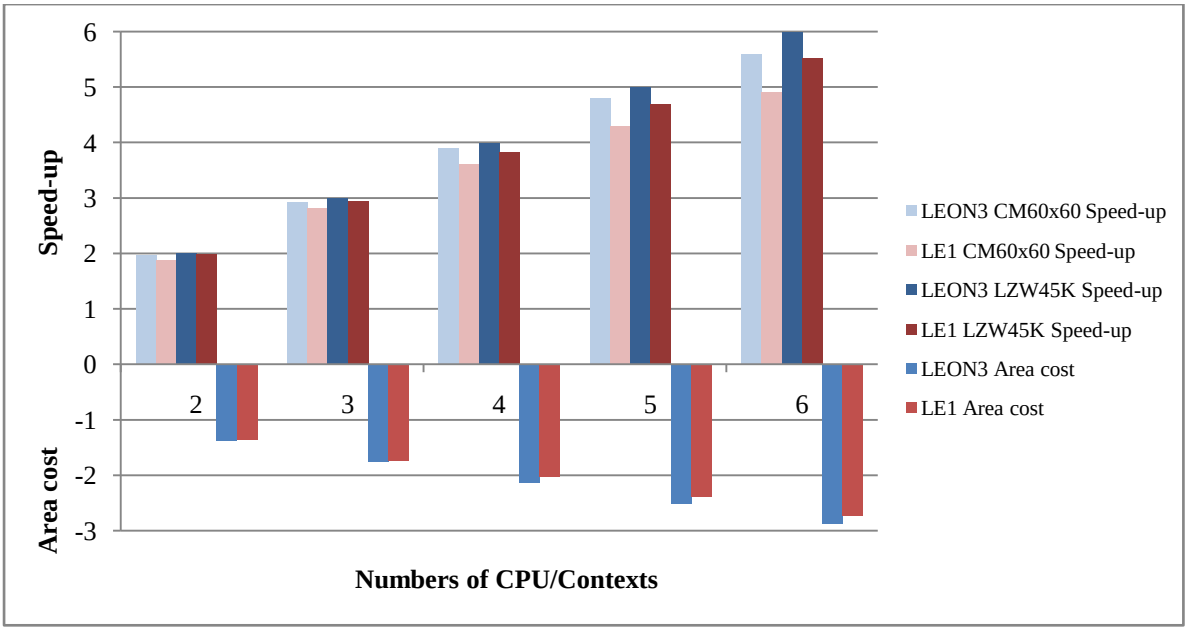

Figure 5.15 LEON3 MP & LE1 CMP Program speed-up vs Area cost.

To investigate the speed-up against the number of processors and area in the these systems, I employed the DSE **Equation 3.2 - 3.4**. The results of program speed-up versus area penalty (the number of LUT-FF Paris) as the number of processors increased and the Speed&Area Efficiency for both of SMP systems were displayed in **Table II.D**, and the histograms have shown in **Fig 5.15** and **Fig 5.16**.

As the result represented, the *CM60x60* speed-up of LEON3&LE1 SMPs were not very linear, especially for the LE1 CMP, the LE1 CMP speed-up ration was dropped more significantly than the LEON3 MP when the number of the cores have increased. However, the *LZW45K* speed-up ration of the LEON3 MP was very linear; in contrast, the LE1 CMP was dropped constantly upon 2-core. In overall, the LEON3 MP system was less effect on data synchronization conflict than LE1 CMP, and program speed-up of the LEON3 MP was always more beneficial than LE1 CMP. Morever, unite with the area cost to the Speed&Area Efficiency, the results of LEON3&LE1 SMPs have shown better tradeoffs when the number of processors increased, and LEON3 MP was advantageous then the LE1 CMP.

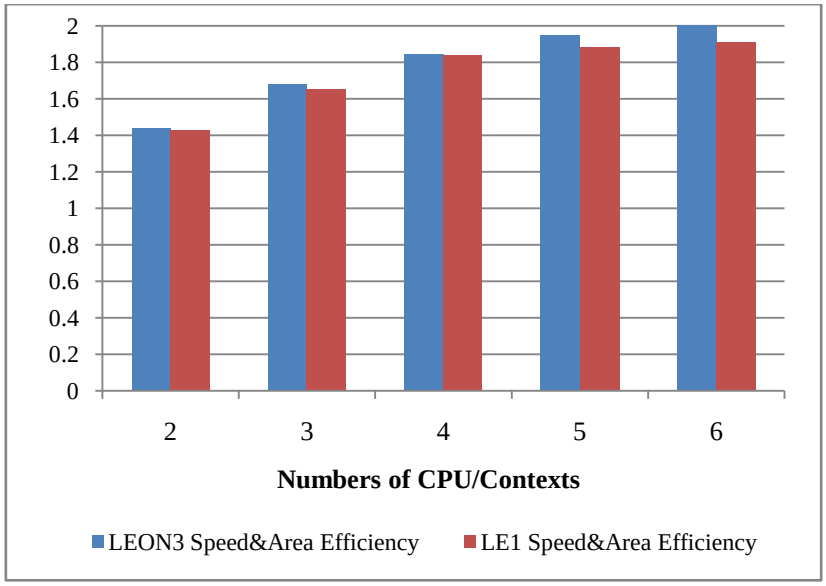

Figure 5.16 LEON3 MP & LE1 CMP Speed&Area Efficiency.

5.6 Summary

Through the "data-intensive" benchmark shown, the LEON3 MP is faster than LE1 CMP in compression&decompression codes and slower in image processing, due to weakness in MUL operations. However, the speed-up efficiency of the LE1 CMP is lower than the LEON3 MP, due to the virtue of the LEON3 MP cache coherence activity. In summary of the average Speed&Area in these SMP systems on the same FPGA, results present that the LEON3 MP was superior in speed and also area tradeoffs. The LE1 CMP is however required more logic gates within the FPGA silicon, and lower the whole system clock; and thus will benefit from implementing it on a larger FPGA board that contains more logic cells to exert its "speed" superiority. Moreover, the number of memories banked in the LE1 CMP shall increase to reduce the data overhead effect than higher the speed-up efficiency. To the next chapter, I will examine and compare HW/SW flows benchmarks in the case of HLS methodology on the LegUp accelerator / Tiger-MIPS processor Co-design system.

Chapter 6

Evaluation in High-level-synthesis of LegUp Co-design System on FPGA-based Processor

6.1 Introduction

This chapter presents "HW/SW" flows of the CHStone benchmark suits executing in LegUp accelerator / Tiger-MIPS processor system of the HLS methodology (also see the description in **Section 3.3**). In general, the hardware design is a lot more difficult to write complex code in HDL and become harder to debug, thus the process time consuming for many applications. On the other hand, software design is always matured and easier to be debugged by analysis tools. The LegUp Co-design system allows researchers to compile selected C segments (the most functions called) to Verilog and synthesized to the hardware accelerator and rest of the segments are executed in the Tiger-MIPS soft processor on the FPGA-based platform. The software profiling is usually implemented by the designer to understand the statistics of the codes, such as time spent in each subroutine before the HW/SW partitioning implementations. The goal of the following experiments is to evaluate the Speed&Area tradeoffs in LegUp Co-design systems, and compare tradeoffs between HW, Hybrid, and SW flows on the FPGAs.

6.2 LegUp Co-design System and Program Profiling

The main LegUp flow consist of running a C program in the Tiger-MIPS processor through a software binary executable (and also provides self-profiling); a portion of the program was being selected by the profiling data and generating the hardware accelerator then scheduling the instructions into a number of c-steps for the DFG through HLS operations of the compiler and; finally the accelerator / processor system is executed and synthesized on the FPGA [14].

In the structures of LegUp, the Tiger-MIPS processor connects to a custom hardware accelerator through the *Altera Avalon* on-chip bus that is generated by Altera SOPC Builder on the FPGA system [91], as shown in **Fig 6.1**. An on-chip data caches with "write-through" policy are supported to the DDR2 SDRAM off-chip memory via a "memory controller" using to the accelerator / processor memory-shared system. In order to maintain data coherency, all global variables that are not constants stored

in main memory [92]. Each hardware accelerator has its own local memory which is stored constant variables and not shared with accelerator / processor system. This allows one cycle memory access in the local RAM.

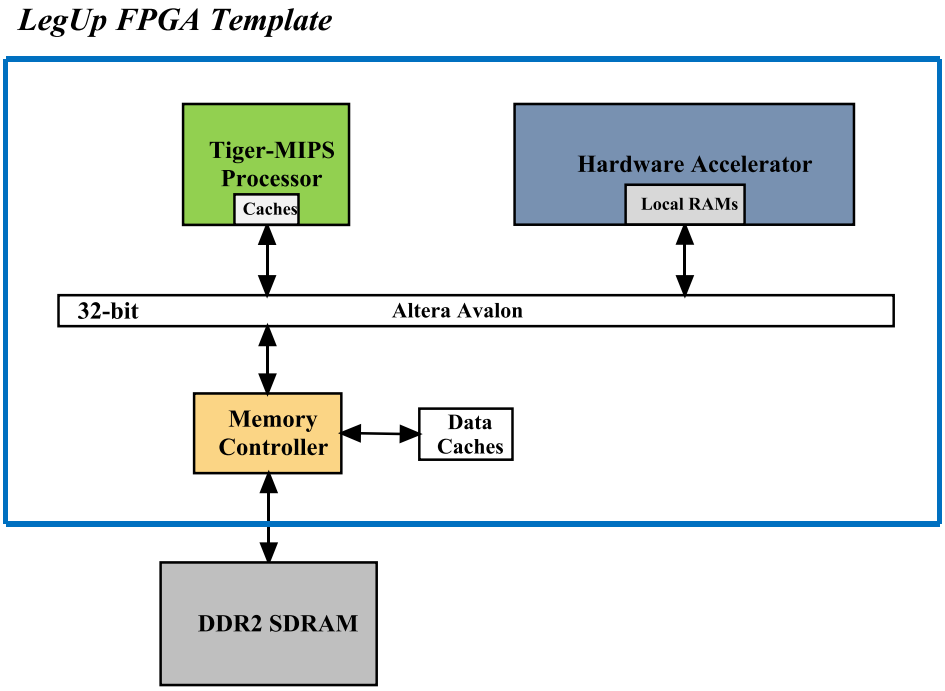

Figure 6.1 LegUp's FPGA target system [14].

6.2.1 The Call-graph Profiler (Kcachegrind)

The data of code profiling is represented in two different formats. There are *flat* and *call-graph* profiling. The flat profile consists of a list of execution times of a program in decreasing order. However, a call graph that represents calling relationships between subroutines. An example of call graph profile listing is shown in **Fig 6.2**. The entry is as "EXAMPLE" that has "Caller" routines as its parent; "Callees" are as its children [93].

Although the original LegUp-system offers a self-profiling tool of "LEAP" (flat profile), I intended to use another code-analyst profiler instead in order to inspect call-graphs directly, and it is called "*Kcachegrind*". The Kcachegrind is a code profiler by using runtime instrumentation, part of Valgrind and displays the call-graph by browsing the performance results. The profiling data is generated by cachegrind and calltree. The tool uses the processor emulation of Valgrind to run the executable then catches memory accesses for the trace; and the program does not need to recompiled [94].

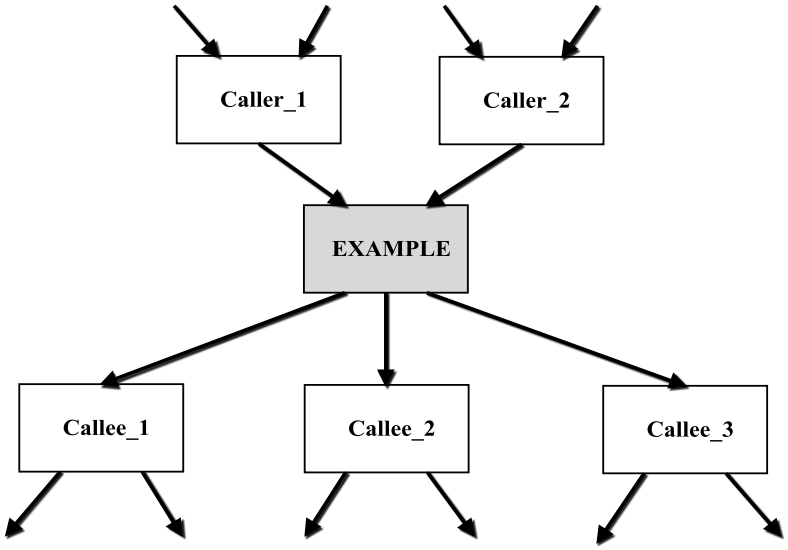

Figure 6.2 Fragment of the call-graph.

6.3 LegUp Architecture and Designed Flows

The LegUp Co-design systems have two different synthesis flows: 1) HW (Hardware Accelerators): Synthesizes the whole *.c files into the hardware accelerator. 2) Hybrid (HW/SW) System: Compile selected C functions into hardware accelerators while executing the remaining segments in the soft-processor (via front-end of the Clang). For C-to-Hardware compiler, back-end of the Clang is the "*Low-Level-Virtual-Machine (LLVM)*" compiler [95] for generating the hardware. To all design flows, I use *Quartus II version 10.1 service pack 1 (10.1sp1)* to target the "Altera DE2" (Cyclone II) in this section study.

Every test program in LegUp has a main function which serves as a "Testbench" with an *if else* statement # RESULT: PASS or FAIL in the end to check the test case passed. *LegUp-2.0* tools support Integer types, Functions, Arrays, Structs, Global Variables, Pointer Arithmetic, and Bitwise. However, there are some software techniques, such as "Dynamic Memory", "FPUs" and "Recursion" is unsupported [92]. Both of systems are able to simulate in the *ModelSim* Stimulator to verify the output and synthesized onto an Altera FPGA. At the beginning, the global setting for targeting Altera FPGA family the designer wish to synthesis and clock period constraints:

- **FAMILY**
  Choose a target FPGA. The synthesized circuits have been verified in hardware using the Altera DE2 (Cyclone II FPGAs) or the Altera DE4 (Stratix IV FPGAs).

- **LEGUP_SDC_PERIOD**
  *15*ns for the Cyclone II or *5* ns for the Stratix IV.

6.3.1 The Hardware (HW) Flow

In LegUp hardware accelerators (*LegUp-HW*), each C function conforms to a Verilog "module". For examples, in a module, inputs of two parameters in the function are provided by a 32-bit (Data) integer "*a*" and a 32-bit (Address Size) pointer "*b*". The "signal ports" that synthesized on the circuits are represented in **Fig 6.3**. The start / reset signals to set to "1" by the initial state of the state machine. The "finish" signal is kept "0" during loops until the last state when "waitrequest" is "1", then 32-bit value of "return_val" is output. The hierarchical of the modules is followed on the call-graph of the C functions. The memory controller is to share data between these modules. Each module communicates its sub-module through "MUXs" to reach the memory controller. However, MUXs area breakdown on the FPGA is large, and LegUp uses the "Bipartite weighted matching" for the binding solution to minimize the number of interconnections that sharing an FU [14, 96].

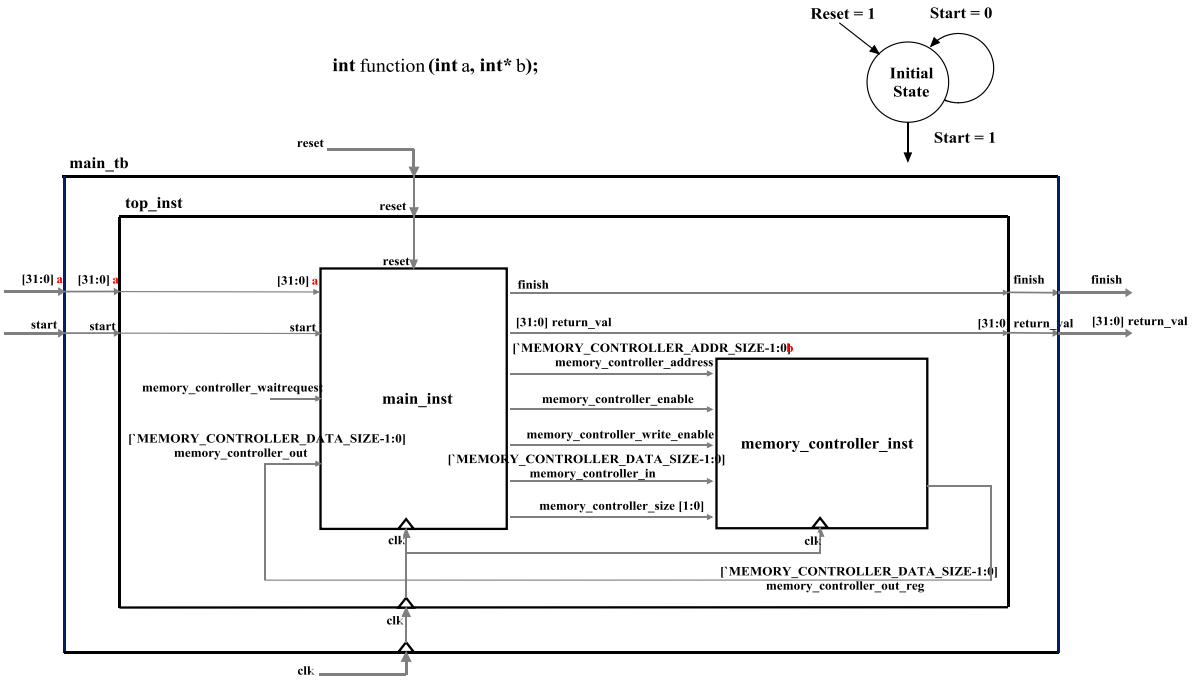

Figure 6.3 The signal port of LegUp in Verilog hardware [92].

Clang compiles the C source into LLVM byte code files (*.bc) and LLVM-compiler reads the *legup.tcl* file that contains a number of "parameters" then produces the Verilog files (*.v). The following parameters control LegUp synthesis operation. The outlines of the important parameters in LegUp to be customized are guided:

- **SDC_ALAP**
  Enable As-Late-As-Possible (ALAP) scheduling instead of As-Soon-As-Possible (ASAP).

- **SDC_NO_CHAINING**
  Disable chaining of operation in clock cycle. This will perform the maximum amount of pipelining. The SDC_PERIOD parameter is useless when this is set.

- **MUL to DSP**
  Enable the binding restricts multiplier to only the number of DSP units available on the target FPGA in the experimental study.

- **SHARE_DIV**
  If set to 1, the *divider* will be shared with any required mux width.

- **SHARE_REM**
  If set to 1, the *remainder* will be shared with any required mux width.

- **MAX_SIZE**
  The maximum chain size to be considered, setting to 0 to disable pattern sharing and shares only dividers and remainders.

- **SHARE_ADD**
  If set to 1, the *addition* will be shared with 2-to-1 muxing.

- **SHARE_SUB**
  If set to 1, the *subtraction* will be shared with 2-to-1 muxing.

- **SHARE_BITOPS**
  If set to 1, the *bit-wise operation* will be shared with 2-to-1 muxing.

- **SHARE_SHIFT**
  If set to 1, the *shift* will be shared with 2-to-1 muxing.

For the "*scheduling*", LegUp performs ASAP (see the definition on page 42) scheduler with operator chaining and pipeline FUs and targets a 66 MHz clock period constraint on the Altera DE2 which assigns each operation into the earliest possible c-step. For the "*allocation*", LLVM instructions hold the unique names of each FUs / registers and generate RTL modules from the scheduled DFG. For the "*binding*", binding operations can be set to force to generate the RTL data structure to DSP (Embedded MULs on the Cyclone II) units on the FPGA. The following parameters in pattern sharing for FUs are enabled by values of SHARE_DIV, SHARE_REM, MAX_SIZE, SHARE_ADD, SHARE_SUB, SHARE_BITOPS, and SHARE_SHIFT, which are set to "1" and MAX_SIZE is set to "10" [92], with those settings, LegUp will achieve the maximum binding for the best result.

6.3.2 The Hardware/Software (Hybrid) Flow

The LegUp HW/SW (*LegUp-Hybrid*) implementation is a Co-design system, where implementing the most intensive computing functions into the hardware accelerator and executing the rest of the program in the Tiger-MIPS soft processor; and the number of functions that input to the hardware are customizable and lead to HW/SW partitioning. Once the profiling data was listed in the Kcachegrind tool, and named manually the accelerated function in the "*config.tcl*" file. The LegUp design flow with other CodeProfiler (Kcachegrind) is shown in **Fig 6.4**.

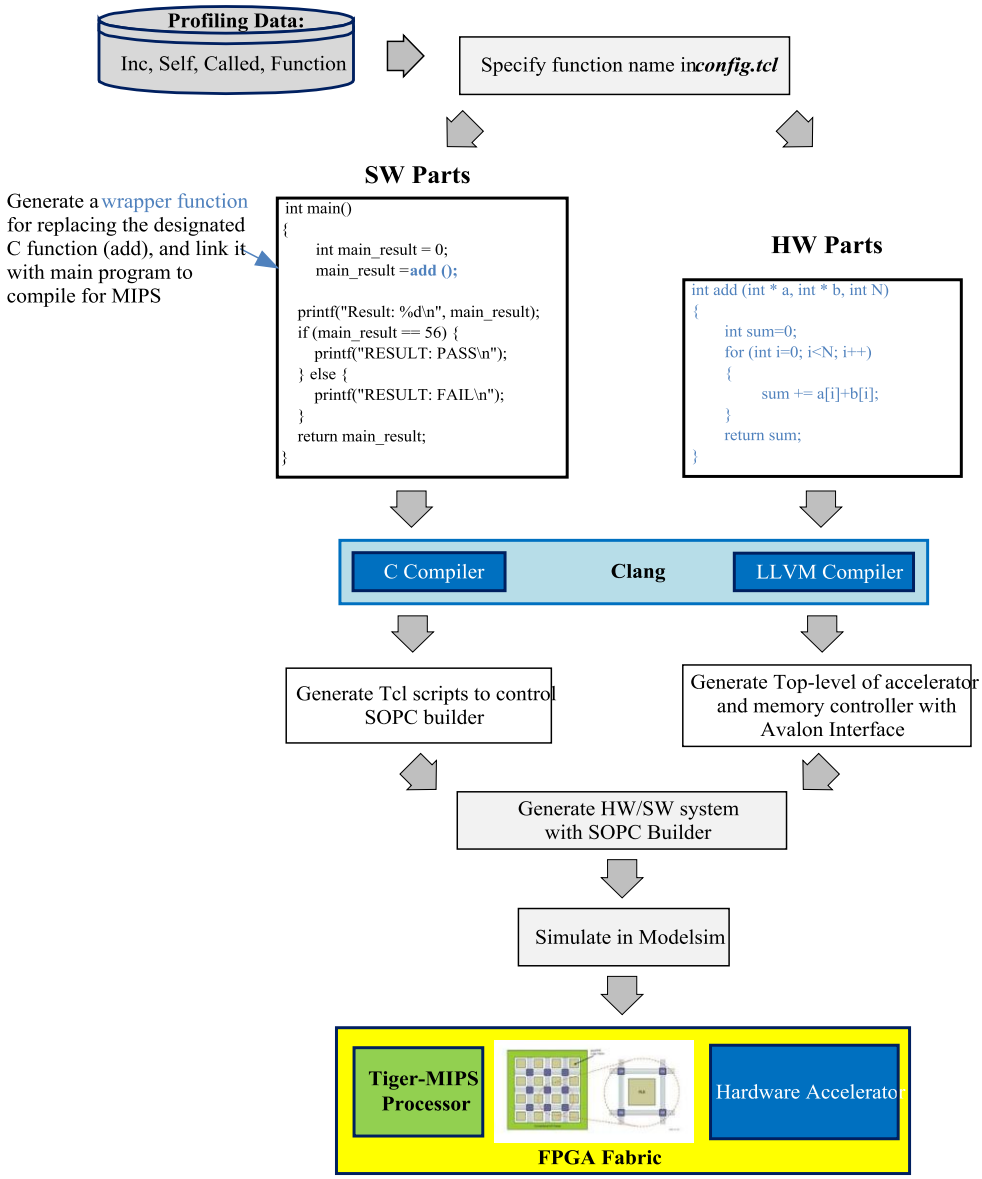

Figure 6.4 LegUp HW/SW Co-design flows.

The processes are divided into HW and SW flows. The SW flow generates a "wrapper" file according to the "config.tcl" file, replacing the wrapper function for the HW portion from the original C function; the "wrapper function" allows accelerator / processor communication without affecting the rest of the software; finally the rest of C program generates the SW binary and also creates "tcl scripts" which control the Altera SOPC Builder. At the same time, the HW flow generates the accelerator by compiling the indicated functions to synthesize RTL data into hardware (the same HLS process in **Section 6.3.1**) as well as creating a top-level Avalon interface to communicate with the soft-processor and the D-caches. Once both SW and HW flows are complete, the tcl script will run the SOPC Builder to put on the accelerator to the system. In the end, ModelSim simulates the accelerator / processor system to display total execution cycles and times. To specify a function to be accelerated in LegUp-Hybrid flow of the config.tcl file:

- **SET_ACCELERATOR_FUNCTION "NAME"**
  To place the name of the C function to accelerate in the config.tcl file for Hybrid-systems.

6.4 Benchmarks of High-Level-Synthesis (HLS)

The input data ("Test data vectors") of the CHStone benchmarks are either "Integer" or "Hexadecimal" values (pointer arithmetic for address space), which all declared in "array" elements of the C standard. All programs were "un-optimized", in order to maintain best average results (Some of them were even upper in non-optimization benchmarks). The important information about those programs is following below sections:

6.4.1 Microprocessor

- **MIPS**
  This is a simplified MIPS processor which has 32 registers and 30 instructions in a *switch-case* statement. A program produced by CHStone is served as a test vector of one input / output data in 8 array elements.

6.4.2 Double Precision Floating-point Arithmetic

The following programs are implemented with IEC/IEEE-standard double-precision floating-point addition package using 64-bit integer numbers. The C source file is part of the SoftFloat and all operations are performed according to this package, Release 2b [75]. There are DFADD, DFMUL, DFDIV, and DFSIN benchmarks:

- **DFADD**
  This is a C program of "*Addition / Subtraction*" arithmetic for 64-bit double-precision floating-point as a test vector of two input data and one output in 46 array elements. Two sub-functions (add/subFloatSign) have been called and there are a number of the control statements such as *if* and *goto* in the loop.

- **DFMUL**

  This is a C program of "*Multiplication*" arithmetic for 64-bit double-precision floating-point as a test vector of two input data and one output in 20 array elements. One sub-function (float64_mul) has been called and there are a number of the control statements such as *if* in the loop.

- **DFDIV**

  This is a C program of "*Division*" arithmetic for 64-bit double-precision floating-point as a test vector of two input data and one output in 22 array elements. One sub-function (float64_div) has been called and there are a number of the control statements such as *if* in the loop.

- **DFSIN**

  This is a C program of "*Sine*" arithmetic for 64-bit double-precision floating-point as a test vector of two input data and one output in 36 array elements. One sub-function (float64_sin) has been called and *do-while* statement enclose a float64_div and a float64_add in the loop.

6.4.3 Media Processing

The following programs are relative to audio / image / video signal processing. There are ADPCM, GSM, JPEG, and MOTION benchmarks:

- **ADPCM**

  This is a C program of "*Audio*" signal processing in telecommunication for the Adaptive-Differential-Pulse-Code-Modulation (ADPCM), which varies the size of the quantization in G.722 CCITT standard (7 kHz wideband audio code operating at 48, 56 and 64 kbit/s) [97]. There are two main sub-functions (encode and decode). Test Vectors: test data input / compression for encoding and result for decoding in 100 array elements.

- **GSM**

  This is another C program of "*Audio*" signal processing for the Linear-Predictive-Coding (LPC) widely used in speech analysis techniques at low bit rate under the (Global-System-for-Mobile-Commutations) GSM that is a communication protocol for mobile phones. There is a main sub-function of Gsm_LPC_Analysis. Test Vectors: test word input / output data in 160 array elements in a *for* loop.

- **JPEG**

  This is a C program of "*Image*" signal processing for the Joint-Photographic-Experts-Group (JPEG) widely method in lossy compression algorithm based on the Discrete-Consine-Transform (DCT) for digital photography. The most important sub-function is (decode_block) that composed of three main parts: Decode Huffman, Inverse Quantization, and Inverse DCT. Important information: Image Height = 59, Image Width = 90, and Sampling Factor 4:1:1. Test Vectors: 7506 array elements.

- **MOTION**
  This is another C program of "*Video*" signal processing for MOTION vector decoding motion estimation process used in picture reference in MPEG-2 standard. There are two main sub-functions (Initialize Buffer and motion_vectors). Test Vectors: test buffer frame input / output in 2048 array elements.

6.4.4 Security

The following programs are relative to encryption is the process of encoding codes for protective data that hackers cannot read it, but the authorized ones can. There are AES, BLOWFISH, and SHA benchmarks:

- **AES**
  This is C program of "*cryptography*" processing for the Advanced-Encryption-Standard (AES) and has been worldwide used in both hardware and software. The methodology of the algorithm is "Symmetric-Key-Algorithm" for both encrypting and decrypting the data. There are two main sub-functions (encrypt and decrypt). Test Vectors: test input data: Statements and Keys are all in 16 array elements respectively; and expect output data for encrypt and decrypt are hexadecimal in 16 array elements. There are a number of the control statements such as a *switch-case* in the loop.

- **BLOWFISH**
  This is another C program of "*cryptography*" processing for Data-Encryption-Standard (DES) in "Symmetric-Block-Cipher" implementation. The program has only the encryption algorithm. There is one main sub-function (BF_encrypt). Test Vectors: test keys of input / output in 5200 array elements.

- **SHA**
  This is another C program of "*cryptography*" processing for Secure-Hash-Algorithm (SHA) of cryptographic hash functions. In hash function process, the input data can be encoded to a message digest. There is one main sub-function (sha_transform) to do the transform. Test Vectors: test input in 2*8192 in 2-Dimensional array elements and expected output in 5 array elements.

6.5 Experimental Methodology and Results

This section work aims to analyze the standard "C" code profiling of *CHStone Suits* HLS and *CM60x60* / *LZW45K* DLP programs (total of ***"14"*** ***benchmarks***) by Kcachegrind tools; and programs were executed in LegUp Co-design systems. For the experimental platform, I expected to continue to use the Altera DE4 board to obtain the benchmarks; regrettably, LegUp-3.0 Hybrid flows were not compatible with the DE4. Therefore, the DE2 maturely collaborates with LegUp-2.0 was selected for this case study (Note: the SW flow performance on the DE2 should have been one third of the DE4, due to the clock period constraints).

The results were represented as the execution cycles / times and resource distribution of each program in following of "LegUp-HW" (whole programs to the accelerator), "Hybrid1" (the most functions called to accelerator), and "Hybrid2" (the second most functions called to the accelerator) that synthesized on *Altera Cyclone II FPGAs* fabric (the number of LEs, Memory-bits, Embedded MULs, and clock frequency on the feasible FPGA) were reported. These results compared against the baseline system of the "MIPS-SW" (same as the Tiger-MIPS SW flow in chapter 4, but was synthesized on the Altera DE2) benchmarks.

Moreover, some of the benchmarks were not able to acquire, there were *MIPS*: Hybrid flows, due to there are only the "main" function in the program; *CM60x60*: Hybrid1 flow, due to there is a technical problem of LegUp wrapper function generation; and *LZW45K*: Execution time of HW flow, due to the generating accelerator on the memory blocks are out of the FPGA resource.

6.5.1 Analysis of Profiling Data

The visualization of profiling data was represented in **Fig 6.5**. The "Total Instruction Fetch Cost" corresponds to the CPU execution time which has shown on the bottom of the figure. The "*Incl*" column is the inclusive cost, represented as time spent in percentage of the function (the blue bar); the "*Self*" is the function called itself; the "*Called*" is the number of the called by that function; and also the "*Function*" is the function name. The "*Location*" tab, the function location of the C files in the host computer can be viewed.

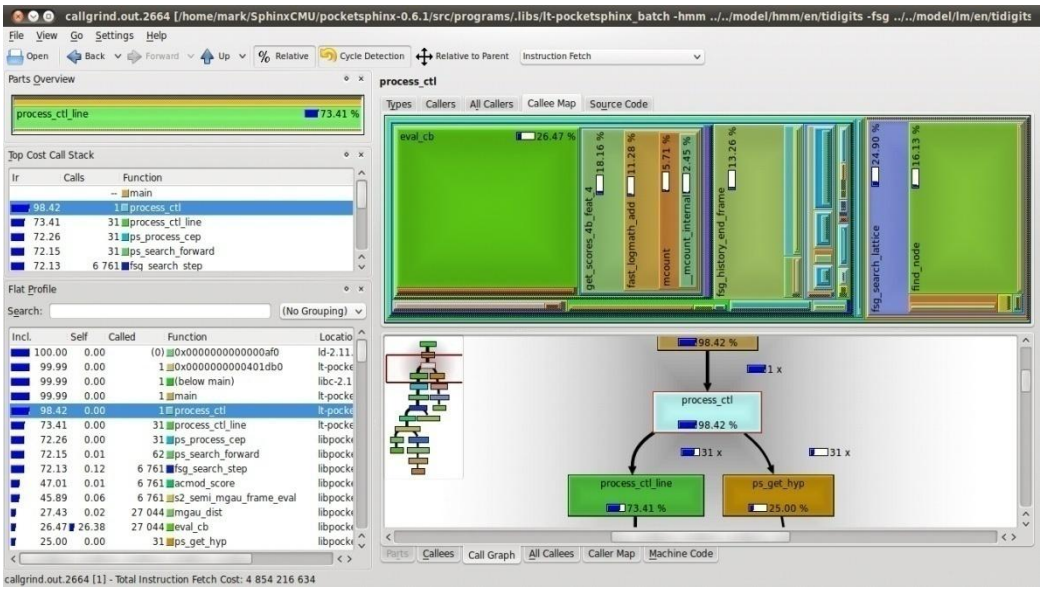

Figure 6.5 The Kcachegrind visualization.

The called-graphs of each program have been profiled by Kcachegrind tools, which were represented in **Fig III.1 – III.11**, **Appendix III**. The top most called functions in the "main()" by whole time spent in each program have shown as well. The top-four most called functions were displayed in **Table III.A** and the chosen two functions to be accelerated to the LegUp-Hybrid system has shown in green colour ticks.

6.5.2 Analysis of Speed and Resource Utilization

Each of the programs was entirely executed in the LegUp-HW. Besides, programs being profiled and identified most function calls earlier on, were executed in the LegUp-Hybrid. The results of all designed flows have shown in **Table III.B**, outline the characterization of each program, such as the representative data type, the number of lines, the number of functions, variables, and the number of operations. The results of *Fmax*, *LEs*, *Devices*, *REGs*, *# bits*, *MULs*, and *Times* for all benchmarks in terms of LegUp-HW, Hybrid1, Hybrid2, and SW flows on the Altera Cyclone II FPGAs were displayed in **Table III.D**. "Fmax" revealed the maximum frequency (MHz) of the overall system, "LEs" revealed the area breakdown within the FPGA expressing in the number of logic elements, "Devices" revealed the given suggested Cyclone II FPGAs, "REGs" revealed the number of registers used on the FPGA, "# bits" revealed the number of memory-bits used on the FPGA, "MULs" revealed the number of embedded multipliers used on the FPGA (9x9-bit), and "Times" revealed the total execution times (μs) in running the whole program. Finally, **Fig III.C** represented that the execution times and area results of all designed flows with the number of line / variables in the figure.

In presentation of figures, the data has shown that the LegUp-HW flow were the best performance of the benchmarks with the smallest area breakdown (some of large codes, such as *DFSIN* and *JPEG* have higher LEs usage than MIPS-SW). Furthermore, there has been found that the HW area breakdown on the number LEs are slightly dependent on the number of the "lines" and "variables" for most of the CHStone benchmarks. This implied that the FPGA resource estimation was able to calculate by these two factors. The average generated LEs per line / variables were *19* and *67*; and the mean value would give an approximate LEs utilization of the LegUp accelerator generation within Cyclone II FPGAs.

6.5.3 Distribution of DSPs and Block RAMs on FPGAs

DSP features on FPGAs can be used as coprocessors especially for multiplications computation; and significantly increase the performance then reduce the total LEs cost (Area), PAR congestions, and power consumptions for lower system costs (see page 32). Cyclone II FPGAs offer up to 300 9x9-bit embedded multipliers that are ideal for low-cost DSP blocks. Each embedded multiplier is configured as one 18x18-bit multiplier or two 9x9-bit multipliers [98]. In LegUp-2.0 synthesis result, it represents 9x9-bit multipliers only.

The numbers of 9x9-bit embedded multipliers in terms of LegUp-HW, Hybrid1, Hybrid2, and MIPS-SW flows for each program have shown in **Fig 6.6**. The dashed line data shows that the actual number of MUL/DIV operations from **Table III.B** in each program. For the observation of the figure, the multiplier utilizations of HW implementations in most of the codes were much higher than SW implementations. This was because the MUL/DIV operations bounded on the embedded MULs in the accelerator. Paying attention that *DFDIV*, *DFMUL*, and *DFSIN* have used the most amount of MULs resource; it was how and what they have saved significant numbers of LEs, hence speed-up the program prominently. In the same manner, the *LZW45K* has used the number of MULs more than its actual number of MUL/DIV operations and would have saved a lot of LEs then raised the execution time considerably. Unfortunately, I have not obtained its benchmark of LegUp-HW due to the failing synthesis of the system. Moreover, other codes such as *GSM* and *JPEG* have used a part

of MULs and remaining of them were implemented either on LEs or block RAMs; these would have certainly increased area significantly.

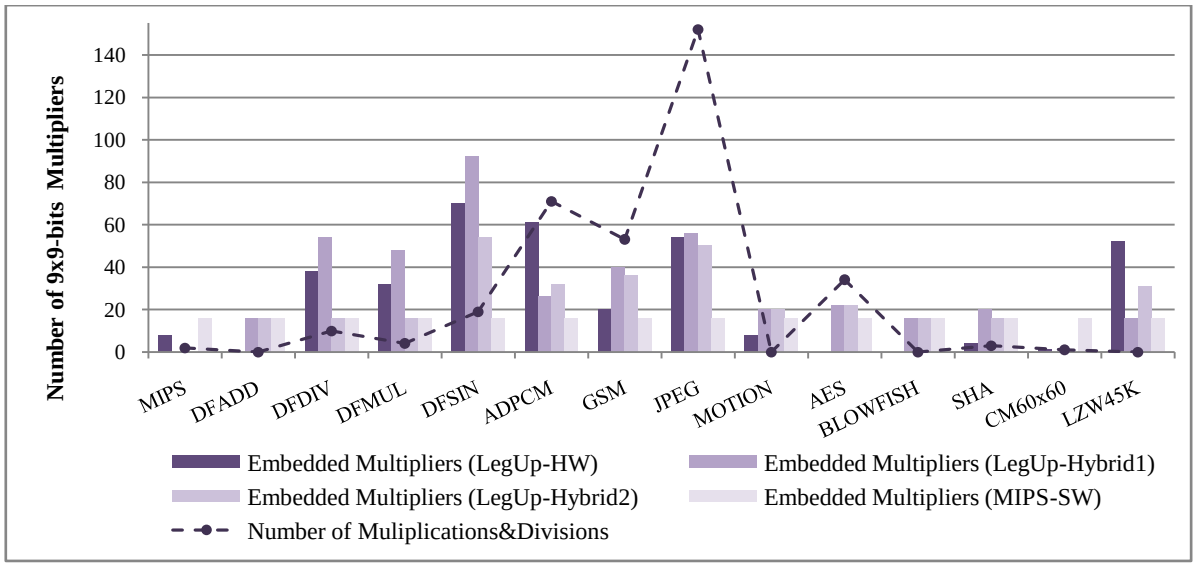

Figure 6.6 Embedded multiplier 9x9-bit elements distribution.

Other important features within FPGAs are embedded memory blocks. Cyclone II devices offer up to 1.1 Mbits of on-chip embedded memory through a fundamental M4K (4,608 bits) blocks. The RAM block is configured to provide various memory functions as single-port RAMs, dual-port RAMs, FIFO buffers, and ROMs, which are ideal using as program storage memory in embedded processors [99]. In LegUp-2.0 synthesis result, it represents the number of memory-bits.

Fig 6.7 has shown that the number of memory-bits usage in all designed flows of each program. The dashed line data shows that the number of test data vectors from **Table III.C** for each program, the block RAMs breakdown by the local memory of accelerators was very dependent on the size of input test data (constant declarations). During this observation, the LegUp-HW flow required much less memory-bits than the MIPS-SW alone. This was because the Tiger-MIPS processor contains cache memories (16KB of I&D-caches) as well as FIFOs, other peripherals, on-chip data caches, and the memory controller; which consume more memory-bits than the local accelerator RAM. Beyond that, the memory-bits consumption of *JPEG*, *BLOWFISH*, and *SHA* was very large, due to huge numbers of integer values were stored in many of "array" elements for encode / decode arithmetic, then required a lot of block RAMs on FPGAs. In the same way, the memory-bits usage of the *CM60x60* was quite numerable, due to the large number of 2-D matrix values were stored in arrays of multiplication arithmetic. Moreover, the number of memory-bits in the *LZW45K* was significantly large, due to out of the memory resource bounds with the FPGA synthesis, and did not fit on any suitable Cyclone II boards.

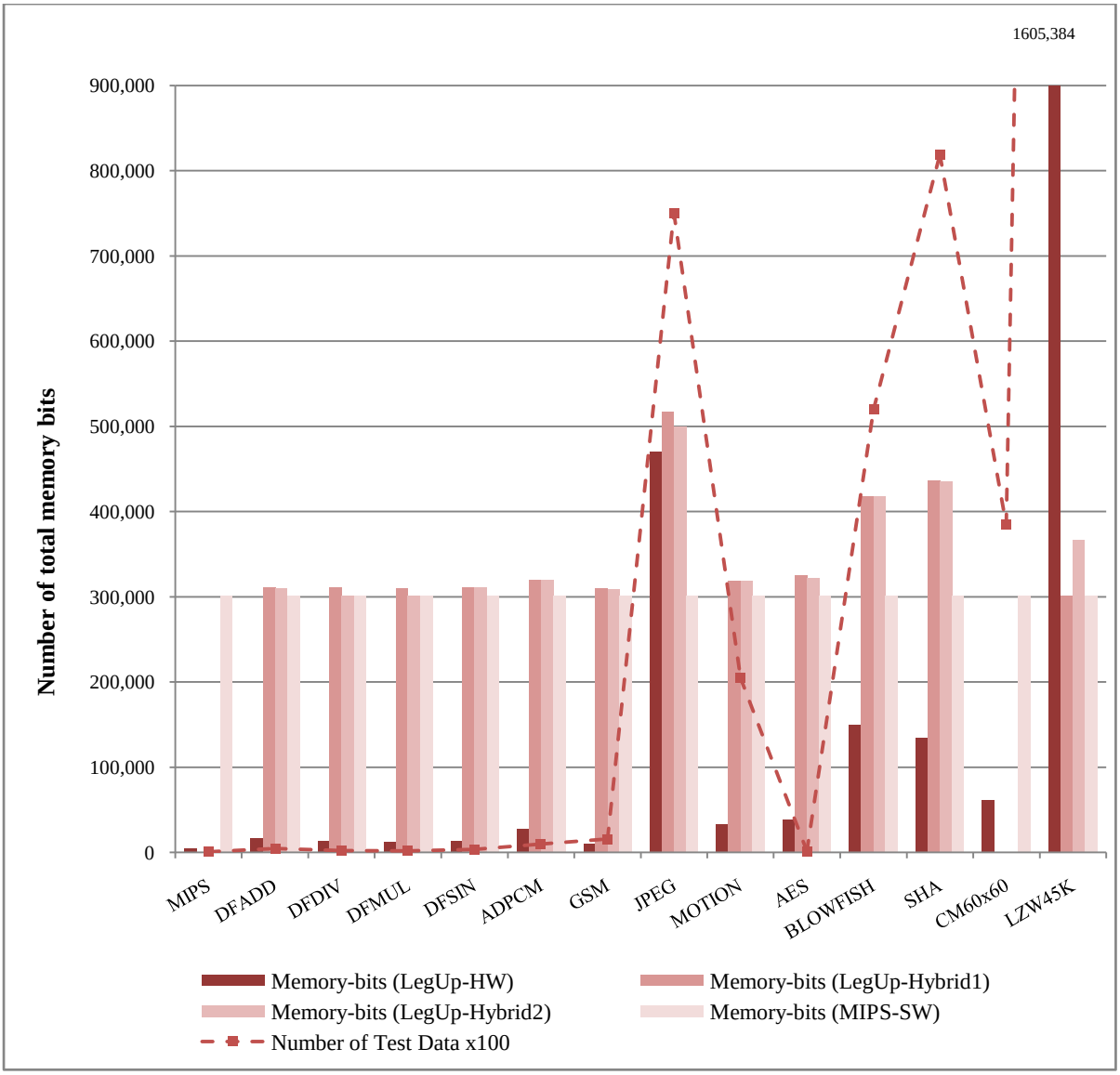

Figure 6.7 Total memory bits distribution.

6.5.4 Speed&Area Tradeoffs in LegUp System on FPGAs

To conclude, the results of average Speed and Area of LegUp-HW, Hybrid1, Hybrid2, and MIPS-SW flows have shown in **Fig 6.8** (Note: the benchmarks of *MIPS*, *CM60x60*, and *LZW45K* were not included, due to their results were not intact). The left vertical axis represented geometric mean execution time as well as the area (average number of LEs). Again, in terms of speed-up, the average execution times were significantly decreased when more computations were implemented in the hardware accelerator. Thus, the more codes implemented in the hardware accelerator, and the more speed-up obtained. In terms of area, the average results have shown that HW implementations taking considerably less area than Hybrid implementations, due to the Tiger-MIPS

processor was not included in the system; and a few more area than SW implementations. In Hybrid implementations, speed and area results were inversely proportional each other.

Moreover, large size codes could remained more area of the LegUp-HW than the MIPS-SW. To this experimental deduction, *682* lines or *191* variables of the designated codes would have probably reached the boundary of this circumstance.

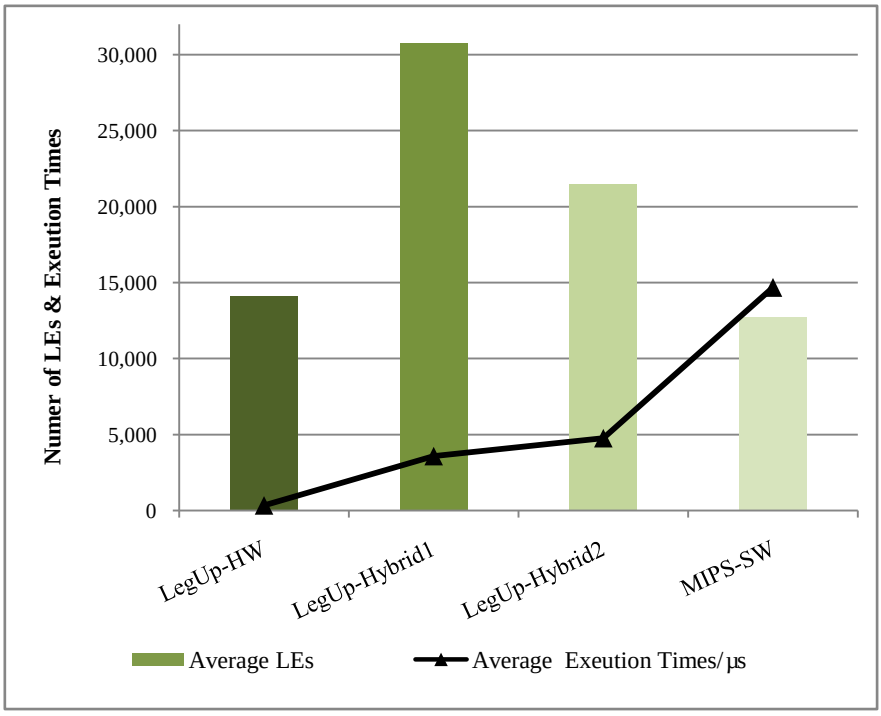

Figure 6.8 The overall performance and area results of LegUp.

To survey the speed-up versus the area cost of LegUp-HW and Hybrid1/2 flows, once again I implanted DSE equations from **Equation 3.5 - 3.7**. Meanwhile, the MIPS-SW flow was the baseline against the comparison. The results of program speed-up versus area penalty (the number of LEs) in terms of LegUp-HW and Hybrid1/2 flows of each program, which displayed in **Table III.E** and the histograms, were plotted in **Fig 6.9**.

As these results represented, *DFADD*, *DFDIV*, *DFMUL*, and *DFSIN* benchmarks in HW implementations have been extraordinary program speed-up. For this inference, it was owing to parentless SoftFloat arithmetic and most of them were implemented to DSP resource. Moreover, the speed-up of LegUp-HW was also briefly dependent on the percentages of the control statements in C programs, as showing in **Figure III.B**. Alone "*if*" and "*goto*" statements in a program were deeply affected on increasing performance by LegUp HLS implementations. However, codes with many "*for*" loops such as *ADPCM*, *JPEG*, *AES*, *BLOWFISH*, *SHA*, and *CM60x60*, which have appeared speed-up faintness.

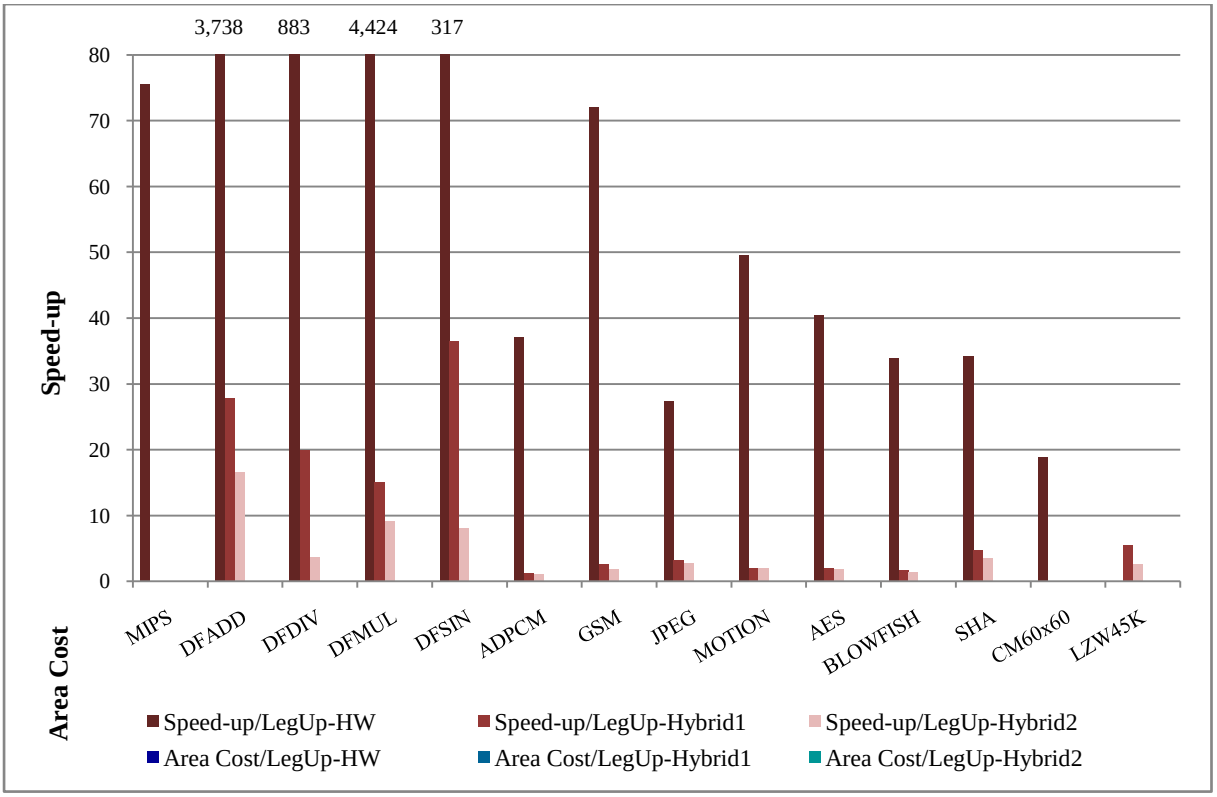

Figure 6.9 The percentage of Speed and Area tradeoffs distribution.

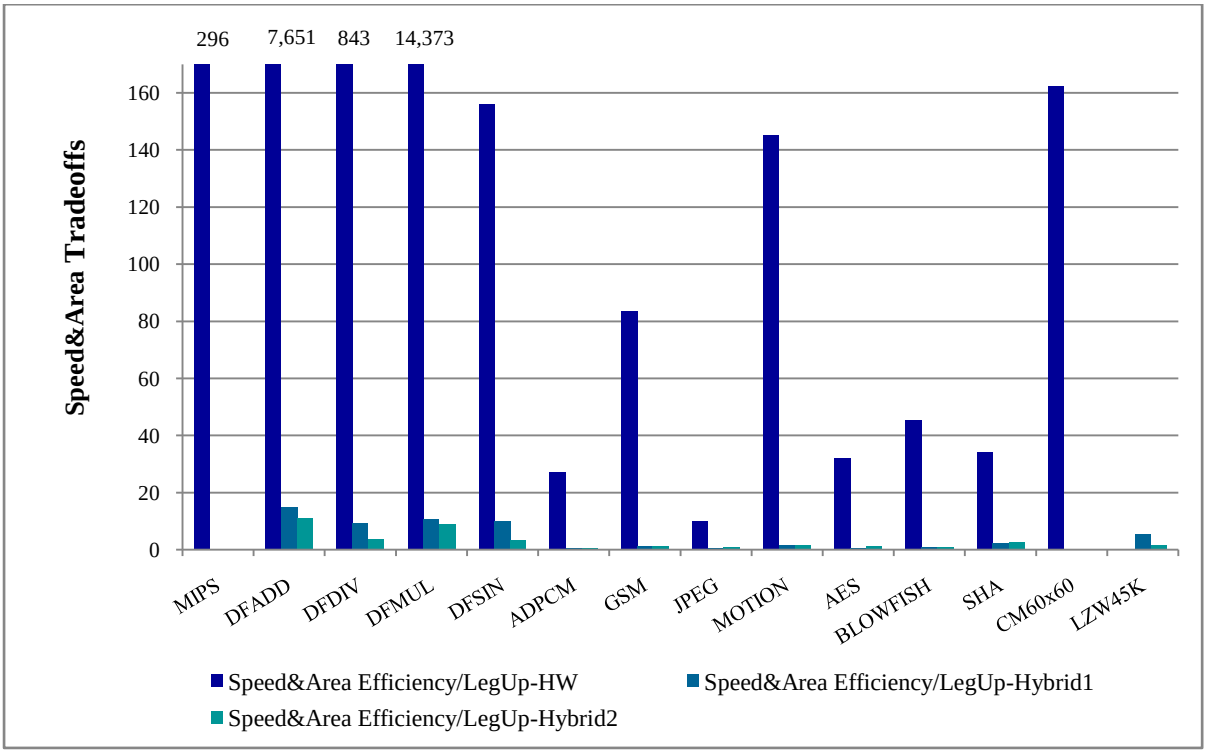

Figure 6.10 The LegUp speed and area efficency.

Moreover, the *CM60x60* and *LZW45K* programs were available to implement in the LegUp Co-design system. For the Speed&Area results of HLS implementations comparing to DLP implementations in **Section 5.5.3**, there has been shown that the *CM60x60* speed-up were significantly better in the HLS of LegUp-HW flows (*18.78*) compared to the best benchmark in the DLP of the 6-core LE1 CMP (*4.91*), and the area was reduced very much (*0.12*); the *LZW45K* speed-up of the LegUp-Hybrid1 (*5.43*) flow was very close to the benchmark at the 6-core of LEON3&LE1 Parallel-flows and the LegUp-Hybrid2 (*2.54*) flow was between at 2 and 3-core of LEON3&LE1 Parallel-flow. However, the area increased were (*1.03* in the LegUp-Hybrid1 and *1.54* in the LegUp-Hybrid2) relative smaller, which around the 2-core of LEON3&LE1 Parallel-flow.

The percentage of the Speed&Area Efficiency for each program in accordance with LegUp-HW and Hybrid1/2 flows are represented in **Fig 6.10**, as shown that the HW implementation was extremely advantage than Hybrids and SW implementations. For the Speed&Area Efficiency comparisons of *CM60x60* and *LZW45K* programs between HLS and DLP implementations, the HLS process outcomes overwhelming efficiency than the best benchmarks of the DLP process, such as the *CM60x60* LegUp-HW flow (*162*) compared to the 6-core of the LEON3 MP (*1.96*). Moreover, the efficiency of the LegUp-Hybrid1 flow in the *LZW45K* (*5.25*) was remained upper than any of the Parallel-SW flows and the LegUp-Hybrid2 flow (*1.65*) was very close to 3-core of LEON3&LE1 SMPs.

6.6 Summary

CHStone programs consist of many sub-functions that called by one another. It is an advantageous case study that suitable for "function-intensive" transformation using LegUp HLS tools. By using the Kcachegrind stimulator of code profiling, software designers can view the call-graphs behaviour and easily choose which functions to be accelerated in the hardware. As results of the Speed&Area evaluations of LegUp Co-design systems, it shows that the HW design flow is the superior tradeoffs, and followed by Hybrid1, Hybrid2, and SW flows. In addition, the analysis of LegUp-system affected by input applications, "the number of DSP usages" and "control-flows" of the codes that dominate the *speed-up* in accelerators; "the number of lines" and "variables" in a C code that dominate the *area cost* in accelerators; and "the size of input test data" in the code dominates the number of *block RAMs* breakdown on FPGAs. Moreover, the DLP programs executing in the HLS methodology also behaves excellent performance and area tradeoffs, but the input data volume are limited by the RAM resource of the FPGA.

Chapter 7

Overall Outline and Conclusion

7.1 Conclusion of LEON3, LE1, and Tiger-MIPS Soft-cores

To chapter 4, I represent the performance of LEON3, LE1, and Tiger-MIPS processors while running a wide-range of C standard benchmarks (fixed-point operations only). To briefly depict the Speed&Area comparisons them within FPGAs (LEON3 is the baseline), the Tiger-MIPS processor has the superior execution times of *2.52*, smaller area of *0.81*; and the LE1 has execution times of *1.04*, larger area of *1.55* times. To conclude the Efficiency ranking are ***Tiger-MIPS (3.11)***, ***LEON3 (1)*** and ***LE1 (0.67)*** in this experimental work.

Although the Tiger-MIPS processor triumphs the comprehensive tradeoffs, however the core is un-configurable and designed to operate on a specific Co-design platform (LegUp), and also possible targeted FPGAs are narrow (only on Altera Cyclone II or Starix IV). Moreover, there is no any commercial or research toolchain for MIPS multi-core processor support for MPSoC platform up to present. Unlike LEON3 and LE1 processors, the cores are highly configurable, multi-processor supported, and available to be targeted to many popular FPGA prototyping boards. Moreover, the LEON3 designed template is an excellent platform for complex SoC design as the GRLIB IP library incorporating LEON3/4 processors and many IP cores. Moreover, the LE1 is a VLIW type, which has double increased instruction throughput than RISCs of LEON3&Tiger-MIPS, however the performance results are not very exceptional, while the larger area is required due to the large number of registers / FUs usage and also the core needs a MicroBlaze to be on the system that consumes considerable area. However, the advantage of the LE1 processor is the number of execution resource for reducing the latency of the core that is optimizable to increase the performance, but the area will increase.

Through the benchmark analysis, the performance of each processor is affected by different algorithm types (see the outline on page 59), the software engineer can rewrite or optimize the C codes to best fit the excellent performance of each processor. In addition, the longer pipeline depths of a CPU will result longer program execution times and also designers need to keep the system frequency low on the FPGA-based to save power computation. The following key sentence outlines the conclusion for an ideal soft-core embedded processor on FPGA-based:

- **As many as compatible and reusable IP cores and targeted FPGAs (LEON3)**
- **Simpler instruction pipeline stages (MIPS)**

7.2 Conclusion of DLP in LEON3 MP and LE1 CMP

To chapter 5, the case study for DLP implementations of *CM60x60* and *LZW45K* benchmarks running in LEON3 and LE1 SMP systems on the Xilinx Virtex-6 LX240T. To summarize the overall results, all the data programs are very linearly speed-up in terms of the number of instruction/cores increase, and the LEON3 is yet better in moving data between CPU registers. To follow up the results, FPGA engineers can select the right number of the cores for the required performance within a FPGA. The Speed&Area Efficiency of LEON3 MP is slightly greater than the LE1 CMP when the number of cores increase. Moreover, by comparing the program speed-up along with the number of issue widths increasing, the performance of CM benchmarks in both systems are very close to each other and the LZW benchmarks of the LEON3 MP system are continuously 3 times better than the LE1 CMP from low to high numbers of cores; and resource utilizations within an FPGA are pretty linear increasing and very similar each other. Therefore, the Efficiency implemented by the instruction-level (ILP) of the LEON3 MP is approximately *3* times greater than the LE1 CMP. To be concluded, the LEON3 MP is however the best tradeoffs and choice of these multi-core processor implementations.

To the local memories, the result shows that a 16KB I&D-cache system of the LEON3 MP is better to deal with "data synchronization confliction" than 2 parallel memory banks of LE1 CMP. However, using caches will consume some block RAMs. In addition, I do not have data results in the other parameter setting of memory banks; therefore these cannot be directly concluded the "cache-system" is a beneficial implementation than the "banked-system".

Furthermore, a VLIW processor needs more logic unit for instruction decoding, thus more registers and FUs are required, the longer gate delay, and it will limit the system clock period. Moreover, the FPGA resources could be used to make more effective implementations by the parallelism of instruction-levels (eg. 4-width solution rather than 4 processors with single-width, thus the area would be considered only by decode units increasing and not the whole cores). The following key sentence outlines the conclusion for an ideal soft-core SMP system within a certain FPGA-based:

- **SPARC RISC processors have superior benchmarks than VEX VLIW processors (LEON3)**
- **Implementing I&D caches for the system to reduce the data synchronization (LEON3)**
- **VLIW processors on FPGA resource may be saved by implementing longer instruction widths rather than increasing the number of the cores to make speed-up (LE1)**

7.3 Conclusion of HLS in LegUp Co-designs

To chapter 6, the case study for HLS implementations of CHStone programs running in LegUp Co-design system on the Altera DE2. To summarize the Speed&Area Efficiency, the HW flow has extremely speed-up with lowest area cost, then Hybrid1/2, and MIPS-SW flow. To follow the results, FPGA engineers can select the right size of the FPGA chip for each program of its performance. Furthermore, it shows that the HW efficiency of the *CM60x60* is significantly better than the best result of the DLP implementation; and Hybrid1/2 flows of the *LZW45K* are *2.5* times better than the 6-core (Hybrid1) and very close to the 3-core (Hybrid2) of DLP benchmarks respectively.

To be concluded, typically, the more pieces of programs accelerating to the hardware, the more execution times being seed-up, the more LEs resource can be preserved, and the best to implement the MUL/DIV operations to DSP units. The data-intensive programs can also be implemented in LegUp HLS tools. However, the LegUp architectures intend to store the constant variables in the local memory; thus the designers have to take care of the input / output sizes within the codes, otherwise, it will lead to a large amount of the block RAMs breakdown on the FPGA.

To compare HLS / DLP implementations, the HLS process (C to HDLs) generates the precise C codes to hardware circuits and representatively multiplications implemented on DSP resource, it does not use too many FPGA area (LUTs), and also speed-up considerably, hence results very high tradeoffs. In contrast, the DLP process is however lower performance implemented by multiple general-purpose processors and consumes many of LUT-FF Paris (soft µPs typically breakdown on conventional fabric and not on DSP resource). However, large programs with multiple *.c files linked by “Makefiles” are difficult to be implemented in HW/SW Co-design systems; general-purpose processors are more advantageous to handle such complex programs, and also keep constant variables in the main memory. The following key sentence outlines the conclusion for an ideal Co-design system within a certain FPGA-based:

- **The design tool needs to be versatile and powerful for verification**
- **Execute as many functions as possible to hardware to obtain high performance**
- **Multiplication applications are ideal to implement on DSP resource**

References

[1] John von Neumann, “First Draft of a Report on the EDVAC”, Between the United States Army Ordnance Department and the University of Pennsylvania, Moore School of Electrical Engineering University of Pennsylvania, 30-06-1945.

[2] The Brith of Transistors and ICs, 16 Brief History of Microprocessors, http://www-scm.tees.ac.uk/users/a.clements/History/History.htm, Retrieved December 30, 2012.

[3] Michael Kanellos, “Moore’s Law to roll on for another decade”, Staff Writer, CNET News, February 10, 2003, http://news.cnet.com/2100-1001-984051.html, Retrieved December 30, 2012.

[4] Applications of Embedded Systems, http://www.vectorindia.org/applications_of_embedded_systems.html, Retrieved December 30, 2012.

[5] Taho Dorta, Jaime Jiménez, José Luis Martín, Unai Bidarte, and Armando Astarloa, “Overview of FPGA-Based Multiprocessor Systems”, Department of Electronics and Telecommunications, University of the Basque Country, UPV/EHU Bilbao, Spain , International Conference on Reconfigurable Computing and FPGAs 2009, pp. 273.

[6] Rajeev Solomon, Peter Sandborn, and Michael Pecht, “Electronic Part Life Cycle Concepts and Obsolescence Forecasting”, IEEE Trans. on Components and Packaging Technologies, Dec. 2000, pp. 707-717.

[7] “Power Consumption at 40 and 45nm”, Xilinx, WP298 (v1.0) April 13, 2009.

[8] “Power Consumption in 65 nm FPGAs”, Xilinx, White Paper: Virtex-5 FPGAs, WP246 (v1.2) February 1, 2007.

[9] SOCcentral: ESL Design, http://www.soccentral.com/results.asp?CatID=484, Retrieved December 30, 2012.

[10] Andreas Gerstlauer, Christian Haubelt, Andy D. Pimentel, Todor Stefanov, Daniel D. Gajski, Fellow, and J ̈urgen Teich, MEMBERS IEEE, “Electronic System-Level Synthesis Methodologies”, pp. 1, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS.

[11] GIOVANNI DE MICHELI, FELLOW, IEEE, AND RAJESH K. GUPTA, MEMBERS IEEE, “Hardware/Software Co-Design”, PROCEEDINGS OF THE IEEE, VOL. 85, NO. 3, MARCH 1997, pp. 357.

[12] LEON3 Processor, EROFLEX GAISLER, http://gaisler.com/index.php/products/processors/leon3, Retrieved December 30, 2012.

[13] D. Stevens and V. Chouliaras, "Le1: A parameterizable vliw chipmultiprocessor with hardware pthreads support," in VLSI (ISVLSI), 2010 IEEE Computer SoCiety Annual Symposium on, july 2010, pp. 122 –126.

[14] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason Anderson, Stephen Brown, and Tomasz Czajkowski, *LegUp: High-Level Synthesis for FPGA-Based Processor/Accelerator Systems*, 1ECE Department, University of Toronto, Toronto, ON, Canada Altera Toronto Technology Centre, Toronto, ON, Canada (2011), pp. 2.

[15] CHStone, A Suite of Benchmark Programs for C-Based High-Level Synthsis, http://www.ertl.jp/chstone/, Retrieved December 30, 2012.

[16] Bharat Bhushan Agarwal and Sumit Prakash Tayal, Chapter 1: Fundamental of computer Architecture, "*Computer Architecture and Parallel Processing*", pp. 22, published by: University Science press, 113 Golden House, Daryaganj, New Delhi-110002, Copyright 2009 by Laxmi Publications Pvt. Ltd. All rights reserved.

[17] Joseph A. Fisher, "Very Long Instruction Word Architecture and the EL1-512", *Proceedings of the 10th annual international symposium on Computer architecture*, International Symposium on Computer Architecture, New York, NY, USA: ACM. pp. 140–150, ISBN: 0-89791-101-6, Volume 11 Issue 3, June 1983.

[18] History in the Computing Curriculum, 1970 to 1979, http://www.hofstra.edu/pdf/CompHist_9812tla6.PDF, Retrieved 30-12-2012.

[19] Daniel Nenni, A Brief History of Field Programmable Devices (FPGAs), Published on 26-08-2012, http://www.semiwiki.com/forum/content/1596-brief-history-field-programmable-devices-fpgas.html, Retrieved 30-12-2012.

[20] Steve Brachmann, eHow Contributor, Introduction to Xilinx, http://www.ehow.com/about_5390865_introduction-xilinx.html, Retrieved 31-12-2012.

[21] EE Times: Clive Maxfield, Xilinx unveil revolutionary 65nm FPGA architecture: the Virtex-5 family, http://www.eetimes.com/electronics-products/fpga-pld-products/4084228/Xilinx-unveil-revolutionary-65nm-FPGA-architecture-the-Virtex-5-family, Retrieved 31-12-2012.

[22] Grant Martin, Gary Smith, "High-Level Synthesis: Past, Present, and Future", High-Level Synthesis, Copublished by the IEEE CS and the IEEE CASS, IEEE Design & Test of Computers, July/August 2009, pp. 20 - 22.

[23] Henry Chang, L.R. Cooke, Merrill Hunt, Grant Martin, Andrew McNelly, Lee Todd, "*Surviving the SoC Revolution: A Guide to Platform-Based Design*", Kluwer Academic Publishers, Springer, 1999.

[24] EE Times: Richard Goering, High-level synthesis rollouts enable ESL, 31-5-2004, http://www.eetimes.com/electronics-news/4154974/High-level-synthesis-rollouts-enable-ESL, Retrieved 16-01-2013.

[25] Qualcomm's New Snapdragon S4: MSM8960 & Krait Architecture Explored, ANANDTECH, http://www.anandtech.com/show/4940/qualcomm-new-snapdragon-s4-msm8960-krait-architecture, Retrieved 30-12-2012.

[26] ARM Corp., Cortex-A9 Processor, http://www.arm.com/products/processors/cortex-a/cortex-a9.php, Retrieved 30-12-2012.

[27] Xilinx Inc., Automotive-grade XA Zynq-7000 All Programmable SoCs, http://www.xilinx.com/products/silicon-devices/soc/xa-zynq-7000/index.htm, Retrieved 31-12-2012.

[28] Xilinx Inc., MicroBlaze Soft Processor Core, http://www.xilinx.com/tools/microblaze.htm, Retrieved 31-12-2012.

[29] SILOS Verilog Simulator, SILVACO, http://www.silvaco.com/products/verilogSimulation/silos.html, Retrieved 30-12-2012.

[30] A Brief History of VHDL, Vhdl Designers Guide, KnowHow, DOULOS, http://www.doulos.com/knowhow/vhdl_designers_guide/a_brief_history_of_vhdl/, Retrieved 30-12-2012.

[31] Keshava Iyengar Satish, "Tutorial on Design For Testability (DFT): An ASIC Design Philosophy for Testability from Chips to Systems", VLSI Technology, Inc. 1109 Mckay Drive, M / S 43, San Jose, CA 9513, 0-7803-1375-5/93, © 1993 IEEE, pp. 130 - 139.

[32] Flynn, Michael, "Some Computer Organizations and Their Effectiveness", IEEE *Trans. Computer:* C-21, No.9, and September 1972, pp. 948 - 960.

[33] Ralph Duncan, Control Data Corporation, "A Survey of Parallel Computer Architectures", *IEEE Computer*: 5–16, February 1990, pp. 5 - 16.

[34] Dr. Heinz-Josef Schlebusch, Director R&D, Synopsys System Level Design, "SystemC based Hardware Synthesis Becomes Reality", System C at Euromicro DSD, Maastricht, The Netherlands, 1089-6503/00, © 2000 IEEE, pp. 1.

[35] NISC Technology & Toolset, http://www.ics.uci.edu/~nisc/, Retrieved 16-01-2013.

[36] SPARK: A Parallelizing Approach the High-Level Synthesis of Digital Circuits, http://mesl.ucsd.edu/spark/, Retrieved 16-01-2013.

[37] LegUp, http://legup.eecg.utoronto.ca/, Retrieved 16-01-2013.

[38] Xilinx Inc., All Programmable FPGAs, http://www.xilinx.com/products/silicon-devices/fpga/index.htm, Retrieved 31-12-2012.

[39] Altera Corp., Altera Stratix V FPGAs: Built for Bandwidth, http://www.altera.com/devices/fpga/stratix-fpgas/stratix-v/stxv-index.jsp, Retrieved 31-12-2012.

[40] Product Specification, Virtex-6 Family Overview, Xilinx, DS150 (v2.4) January 19, 2012.

[41] Doug Amos, Austin Lesea, Rene Richter, Chapter 1: Introduction: the challenge of system verification, "*FPGA-BASED PROTOTYPING METHODOLOGY MANUAL: Best practices in Design-for-Prototyping*", pp. 22, 100-108, Published by Synopsys, Inc., Mountain View ,CA, USA, Copyright 2011 Synopsys, Inc., Portions Copyright 2009-2011 Xilinx, Inc., Portions Copyright 2011 ARM Limited.

[42] Abeer Hyari, "A Comparative Study on Heterogeneous and Homogeneous Multiprocessors", University of Jordan, Department of Computer Engineering, Submission date: 2-DEC-2009.

[43] Appliced Micro Circuits Corporation, Computing Products, Product Selector Guide, http://www.apm.com/products/embedded/, Retrieved 18-01-2013.

[44] IBM PowerPC 440 Embeded Core, Printed in the United States of America November 2006, TGD01473-USEN-02.

[45] ARM Corp., Processors, http://www.arm.com/products/processors/, Retrieved 18-01-2013.

[46] MIPS TECHNOLOGIES, Processor Cores, http://www.mips.com/products/processor-cores/classic/, Retrieved 18-01-2013.

[47] Soft CPU Cores for FPGA, 1-Core Technologies, http://www.1-core.com/library/digital/soft-cpu-cores/, Retrieved 18-01-2013.

[48] Xilinx Inc., Products, http://www.xilinx.com/products/, Retrieved 18-01-2013.

[49] Altera, Corp., SoCs/Processers, Nios II, http://www.altera.com/devices/processor/nios2/ni2-index.html, Retrieved 18-01-2013.

[50] Leon3 Processor, EROFLEX GAISLER, http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53, Retrieved 18-01-2013.

[51] LATTICE SEMICONDUCTOR, Lattice IP Cores, LatticeMico32, http://www.latticesemi.com/products/intellectualproperty/ipcores/mico32/index.cfm, Retrieved 18-01-2013.

[52] EE Times: FPGA Soft Processor Design Considerarion, RC Cofer and Ben Harding, http://www.eetimes.com/design/programmable-logic/4014791/FPGA-Soft-Processor-Design Considerations, Retrieved 18-01-2013.

[53] Petar Borisov Minev, Valentina Stoianova Kukenska, Technical University of Gabrovo, "IMPLEMENTATION OF SOFT-CORE PROCESSORS IN FPGAs", International Scientifc conference 23 – 24 November 2007, pp. 1, GABROVO.

[54] Carlos Carvalho, "The Gap between and Memory Speeds", Departamento de Informática, Universidade do Minho 4710 - 057 Braga, Portugal, ICCA'02, pp. 27 - 34.

[55] Gregory V. Wilson, "The History of the Development of Parallel Computing", http://ei.cs.vt.edu/~history/Parallel.html, Retrieved 26-01-2013.

[56] Lars S. Nyland, Jan F. Prins, Allen Goldberg, and Peter H. Mills, IEEE Transactions on Software Engineering, "A Design Methodology for Data-Parallel Applications", Vol. 26, No. 4, April 2000, pp. 293 - 314.

[57] Daniel D. Gajski, Loganath Ramachandran, "Introduction to High-Level Synthesis," *IEEE Design and Test of Computers*, vol. 11, no. 4, pp. 44-54, Oct. 1994, doi:10.1109/54.329454.

[58] Jason Cong and Zhiru Zhang, "An Efficient and Versatile Scheduling Algorithm Based on SDC Formulation", Computer Science Department, University of California, Los Angeles, USA, June 24 – 28, 2006, ACM 1-59593-381-6/06/0007, pp. 433 - 438.

[59] Xilinx Inc., Virtex-6 Family Overview, DS150 (v2.4) January 19, 2012.

[60] Altera, Corp., Stratix III Device Family Architecture, http://www.altera.com/devices/fpga/stratix-fpgas/stratix-iii/overview/architecture/performance/st3-alm-structure.html, Retrieved 19-04-2013.

[61] Altera, Corp., White Paper, Stratix III FPGAs vs. Xilinx Virtex-5 Devices: Architecture and Performance Comparison.

[62] M. Aldham, J.H. Anderson, S. Brown, A. Canis, "Low-Cost Hardware Profiling of Run-Time and Energy in FPGA Embedded Processors," IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP), Santa Monica, CA, September 2011, pp. 61 - 68.

[63] Altera, Corp., San Jose, CA. Cyclone II Device Family Data Sheet, 2011.

[64] Chapter 63: LEON3 - High-performance SPARC V8 32-bit Processor GRLIB IP Core User's Manual Version 1.1.0 - B4113, pp. 666 - 680, Janaury 2012,Copyright Aeroflex Gaisler, 2012.

[65] Chapter 77: REGFILE_3P 3-port RAM generator (2 read, 1 write), GRLIB IP Core User's Manual Version 1.1.0 - B4113, pp. 819, Janaury 2012,Copyright Aeroflex Gaisler, 2012.

[66] "The SPARC Architecture Manual Version 8", Revision SAVO80SI9308, SPARC International Inc., 535 Middlefield Road, Suite 210 Menlo Park, CA94025, 415-321-8692.

[67] D. Stevens and V. Chouliaras, "Le1: A parameterizable vliw chip multiprocessor with hardware pthreads support," in VLSI (ISVLSI), 2010 IEEE Computer SoCiety Annual Symposium on, july 2010, pp. 122 –126.

[68] D. Stevens, V. Chouliaras, V. Azorin-Peris, J. Zheng, A. Echiadis, and S. Hu*, *Senior Member, IEEE*, "BioThreads: A novel VLIW-based Chip-Multi-Processor for accelerating biomedical image processing applications", June 2012, pp. 257 - 268.

[69] The Tiger "MIPS" Processor, ECAD and Architecture Practical Classes, Computer Laboratory, University of Cambridge, http://www.cl.cam.ac.uk/teaching/0910/ECAD+Arch/mips.html, Retrieved 19-04-2013.

[70] MIPS Architecture For Programmers Volume I-A: Introduction to the MIPS32 Arcitecture, MIPS Technologies, Inc., Document Number: MD00082, Revision 3.02, March 21, 2011.

[71] 6.2 A Pipelined Datapath, "*Computer Organization & Design the Hardware/Software Interface*", Second Edition 1998 , pp. 450-465, Morgan Kaufmann Publishers, Inc. San Francisco, California, Printed in the USA.

[72] LEON Bare-C Cross Compliation System (BCC), EROFLEX GAISLER, http://www.gaisler.com/index.php/products?option=com_content&task=view&id=147, Retrieved 25-05-2013.

[73] VEX Toolchain, http://www.hpl.hp.com/downloads/vex/ , Retrieved 16-05-2012.

[74] The LZ78 algorithms, http://oldwww.rasip.fer.hr/research/compress/algorithms/fund/lz/lz78.html, Retrieved 25-05-2013.

[75] SoftFloat, http://www.jhauser.us/arithmetic/SoftFloat.html, Retrieved 25-05-2013.

[76] SNU Real-time Benchmarks, http://archi.snu.ac.kr/realtime/benchmark/, Retrieved 15-01-2011.

[77] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communicatons systems," MICRO, 1 - 3 Dec 1997, pp. 330 - 335.

[78] A. C. Hung, "PVRG-JPEG CODE 1.1," Technical Report, Stanford University, 1993.

[79] AILab, http://www-ailab.elcom.nitech.ac.jp/, Retrieved 25-05-2013.

[80] M. R. Guthaus, J. S. Ringenberg, and D. Ernst, "MiBench: A free, commercially representative embedded benchmark suite," WWC, 2001, pp. 3 - 14.

[81] Virtex-6 FPGAs, Xilinx FPGAs.

[82] John L. Hennessy & David A. Patterson, "*Computer Architecture: A Quantitative Approach*", Third Edition 2003, Morgan Kaufmann Publishers, Printed in the USA.

[83] Thomas Anderson. "The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors", *IEEE Transactions on Parallel and Distributed Systems, vol. 1, num. 1,* January 1990, pp. 6 - 15.

[84] Mohsan Tanveer, M. Aqeel Iqbal, Farooque Azam," Using Symmetric Multiprocessor Architectures for High Performance Computing Environments", Dept. of SE, Foundation University, Institute of Engineering and Management Sciences (FUIEMS), Rawalpindi, Pakistan, International Journal of Computer Applications (0975 – 8887) Volume 27– No.9, August 2011, pp 1 - 6.

[85] GRLIB IP Library User's Manual Version 1.1.0 B4108, June, 2001, Copyright Aeroflex Gaisler, 2010.

[86] LogiCORE IP Processor Local Bus (PLB) v4.6 (v1.05a), Xilinx, DS531 September 21, 2010.

[87] Image Processing Convolutions, http://beej.us/blog/data/convolution-image-processing/, Retrieved 25-05-2013.

[88] T. A. Welch, "A technique for high-performance data compression," IEEE Computer, vol. 17, no. 6, pp. 8–19, 1984.

[89] L. S. Tan, S. P. Lau, and C. E. Tan, "Optimizing lzw text compression algorithm via multithreading programming," in Communications (MICC), 2009 IEEE 9th Malaysia International Conference on, dec. 2009, pp. 592 –596.

[90] Virtex-6 FPGA Memory Resource User Guide, UG363 (v1.7) September 24, 2013.

[91] Altera, Corp. Avalon Interface Specification, 2010.

[92] LegUp Documentation, Release 2.0, University of Toronto, December 19, 2011.

[93] Susan L. Graham Peter B. Kessler Marshall K. McKusick, "*gprof: a Call Graph Execution Profiler*", Computer Science Division Electrical Engineering and Computer Science Department University of California, Berkeley Berkeley, California 94720, PSD:18-1, pp. 120 - 126.

[94] Kcachegrind Call Graph Viewer, http://kcachegrind.sourceforge.net/html/Home.html, Retrieved 25-05-2013.

[95] The **LLVM** Compiler Infrastructure, http://www.llvm.org/, Retrieved 25-05-2013.

[96] C.Y. Huang, Y.S. Che, Y.L. Lin, and Y.C. Hsu, "Data path allocation based on bipartite weighted matching", Department of Computer Science, Tsing Hua University, Hsin-Chu, Taiwan 30043, Republic of China, 1990, pp. 499 - 504.

[97] ITU Recommendation G.722, "7 kHz audio-coding within 64 kbit/s", http://www.itu.int/rec/T-REC-G.722/e, Retrieved 07-05-2013.

[98] Altera, Corp**,** 12. Embedded Multipliers in Cyclone II Devices, Cyclone II Device Handbook, Volume 1, CII51012-1.2, February 2007.

[99] Altera, Corp**,** 8. Cyclone II Memory Blocks, Cyclone II Device Handbook, Volume 1, CII51008-2.4-1.2, February 2008.

Appendix

Appendix I *Benchmark Collections - SW Flow*

I.1 SW Flow in LEON3, LE1, and Tiger-MIPS Processors

Function name	LEON3 CycleCount	LEON3 (*75*MHz) Execution Time/μs	LE1 CycleCount	LE1(*75*MHz) Execution Time/μs	MIPS CycleCount	MIPS (*74.26*MHz) Execution Time/μs
cm62x62	1031175	13749	370019	4933.59	523449	2617.23
lzw45k	935278425	12470379	1173688753	15649183.37	1059760839	5298785.267
CHStone/mips	36675	489	47069	627.59	41407	207.02
CHStone/adpcm	247050	3294	111269	1483.59	163362	816.795
CHStone/gsm	98700	1316	39353	524.71	37575	187.86
CHStone/jpeg	3251850	43358	3973077	52974.36	3935361	19676.79
CHStone/motion	13650	182	-	179.147	30406	152.015
CHStone/aes	48900	652	49128	655.04	49275	246.36
CHStone/blowfish	717075	9561	830419	11072.25	910256	4551.265
CHStone/sha	775725	10343	574712	7662.83	1083598	5417.975
array	75	1	136	1.81	290	1.435
dhrystone	27825	371	28190	375.87	24966	124.815
div_const	975	13	289	3.85	186	0.915
fft	54000	720	33603	448.04	22603	113
fir	75	1	4409	58.79	8927	44.62
function_pointer	150	2	67	0.89	167	0.82
functions	75	1	69	0.92	110	0.535
hierarchy_test	75	1	133	1.77	93	0.45
llist	525	7	138	1.84	315	1.56
loadstore	0	0	10	0.13	3	0
loop	75	1	305	4.07	708	3.525
loopbug	150	2	483	6.44	192	0.945
malloc	3075	41	2407	32.09	2249	11.23
memory_access_test	3900	52	1750	23.33	3924	19.605
memset	75	1	1468	19.57	2346	11.715
ogg	35370000	471600	38486527	513153.69	56749338	283746.675
select	0	0	41	0.547	54	0.255
shift	150	2	68	0.91	111	0.54
signeddiv	375	5	164	2.19	224	1.105
struct	1425	19	642	8.56	3759	18.78
unaligned	0	0	187	2.49	3	0
switches	75	1	39	0.52	57	0.27
sra	75	1	42	0.56	90	0.435
tiger/sra	0	0	135	1.8	3	0

Table I.A Benchmarks of LEON3, LE1, and Tiger-MIPS single-cores.

Appendix II *DLP Benchmarks - Parallel-SW Flow*

II.1 Parallel-SW Flow in LEON3 MP and LE1 CMP

Number of CPUs	Convolution-Matrix 60*60 (CM60x60)				Lempel-Ziv-Welch 45KB (LZW45K)			
	LEON3 (60MHz)		LE1(50MHz)		LEON3 (60MHz)		LE1 (50MHz)	
	CycleCount	Execution Time/µs	CycleCount	Execution Time/µs	CycleCount	Execution Time/µs	CycleCount	Execution Time/µs
1	1030200	17170	370019	7400	935031720	15583862	1173688753	23473775
2	526020	8767	196894	3938	467804940	7796749	589744294	11794886
3	351660	5861	131283	2626	312165240	5202754	399362970	7987259
4	264180	4403	102708	2054	234231420	3903857	306433205	6128664
5	214920	3582	86102	1722	187311900	3121865	250415720	5008314
6	184320	3072	75395	1508	156329280	2605488	212593020	4251860

Table II.A LEON3 MPs and LE1 CMPs simulated execution cycles and times (at 60MHZ and 50MHZ).

Number of CPUs	LEON3 MP (60MHz)			LE1 CMP (50MHz)		
	RAMB36 (%)	RAMB18 (%)	DSP48 (%)	RAMB36 (%)	RAMB18 (%)	DSP48 (%)
1	76 (18.27)	50 (6.01)	4 (0.52)	101 (24.28)	8 (0.92)	7 (0.91)
2	86 (20.67)	100 (12.02)	8 (1.04)	117 (28.13)	16 (1.92)	11 (1.43)
3	32 (7.69)	246 (29.57)	12 (1.56)	133 (31.97)	24 (2.88)	15 (1.95)
4	42 (10.10)	328 (39.42)	16 (2.08)	149 (35.82)	32 (3.85)	19 (2.47)
5	52 (12.50)	410 (49.28)	20 (2.60)	165 (39.66)	40 (4.81)	23 (2.99)
6	62 (14.90)	396 (47.60)	24 (3.13)	181 (43.51)	48 (5.77)	27 (3.52)

Table II.B LEON3 MPs and LE1 CMPs (with SoC) RAMs & DSPs breakdown on XILINX VIRTEX 6 ML605 FPGA.

Number of CPUs	LEON3 MP (60MHz)		LE1 CMP (50MHz)	
	LUT-FF (%)	RAMBs (%)	LUT-FF (%)	RAMBs (%)
1	15596 (10.35)	126 (10.10)	24454 (16.22)	109 (8.73)
2	21470 (14.25)	186 (14.90)	33107 (21.97)	133 (10.66)
3	27507 (18.25)	278 (22.28)	42583 (28.25)	157 (12.58)
4	33372 (22.14)	370 (29.65)	49460 (32.82)	181 (14.50)
5	39222 (26.02)	462 (37.02)	58416 (38.76)	205 (16.43)
6	44838 (29.75)	458 (36.70)	66756 (44.30)	229 (18.35)

Table II.C LEON3 MPs and LE1 CMPs total area and RAMBs breakdown on XILINX VIRTEX 6 ML605 FPGA.

Number of CPUs	CM60x60 Speed-up		LZW45K Speed-up		Average Speed-up		Area cost		Speed&Area	
	LEON3	LE1	LEON3	LE1	LEON3	LE1	LEON3	LE1	LEON3	LE1
2	1.96	1.88	2.00	1.99	1.98	1.94	1.38	1.35	1.44	1.43
3	2.93	2.82	3.00	2.94	2.96	2.88	1.76	1.74	1.68	1.65
4	3.90	3.60	3.99	3.83	3.95	3.72	2.14	2.02	1.84	1.84
5	4.79	4.30	4.99	4.69	4.89	4.49	2.51	2.39	1.95	1.88
6	5.59	4.91	5.98	5.52	5.79	5.21	2.87	2.73	2.01	1.91

Table II.D The results of speed-up versus area cost and Speed&Area Efficiency of LEON3&LE1 SMPs.

Appendix III *HLS Benchmarks - HW, Hybrids, and SW Flows*

III.1 Called-graphs of Profiling for Benchmarks by Kcachegrind

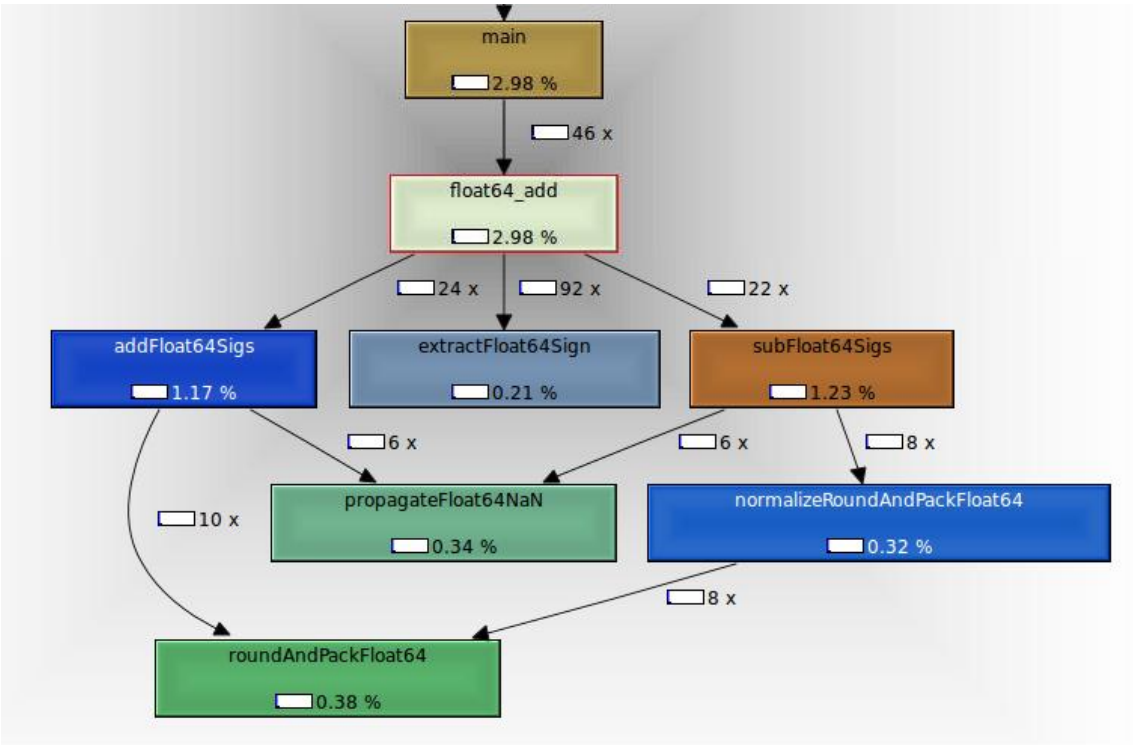

Figure III.1 The called-graphs of **DFADD**.

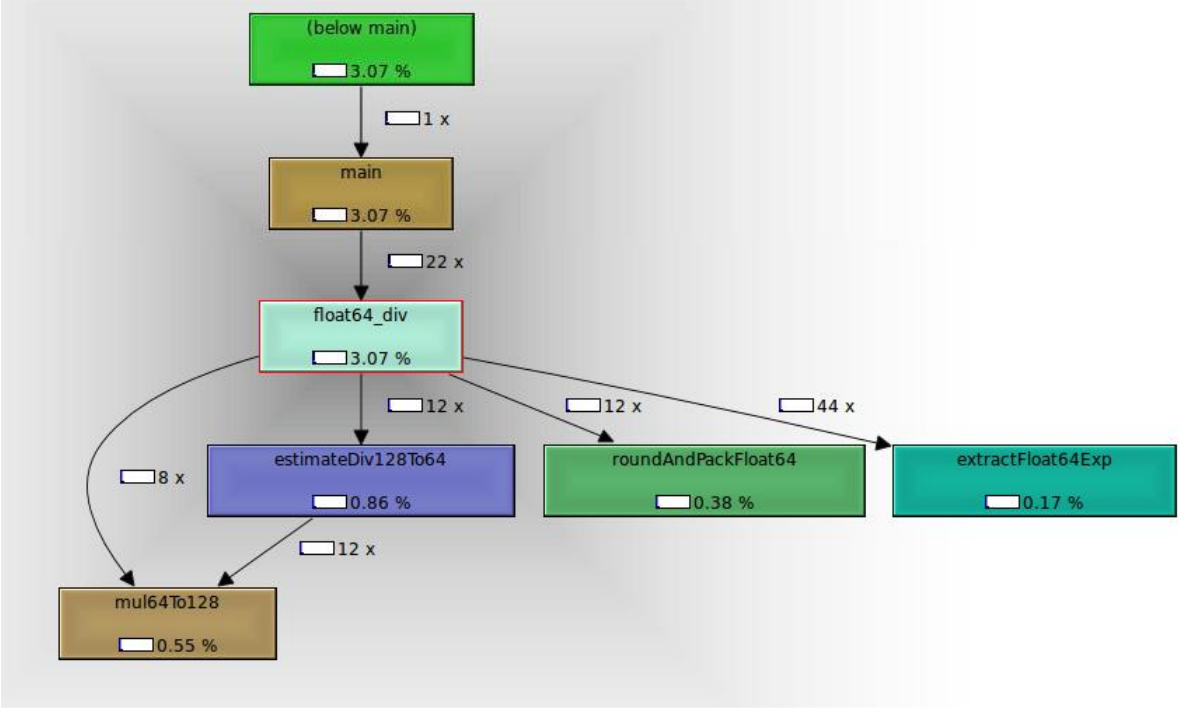

Figure III.2 The called-graphs of **DFDIV**.

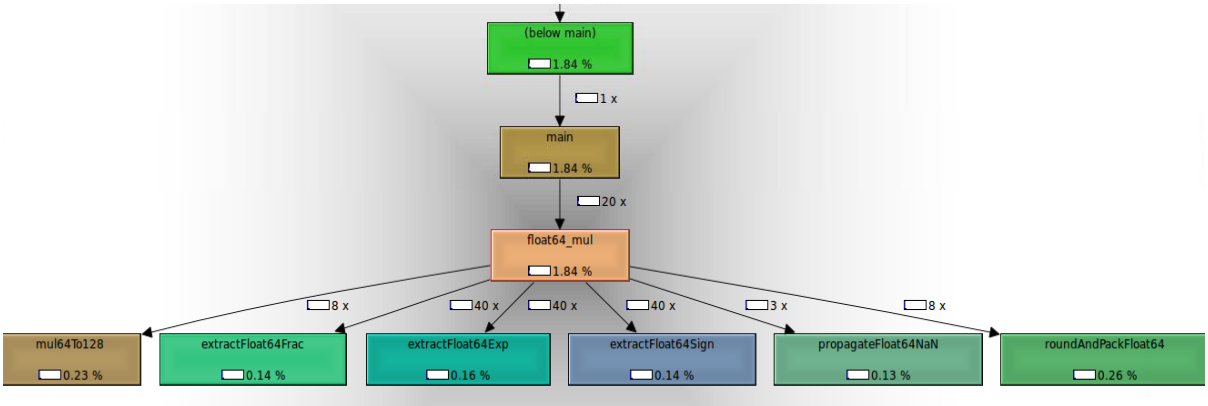

Figure III.3 The called-graphs of **DFMUL**.

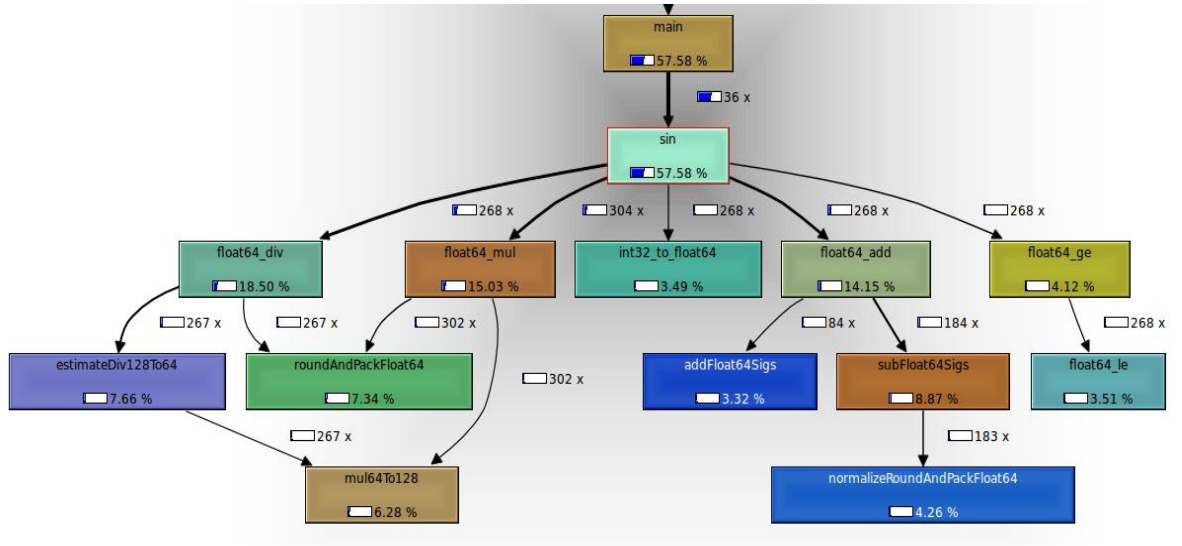

Figure III.4 The called-graphs of **DFSIN**.

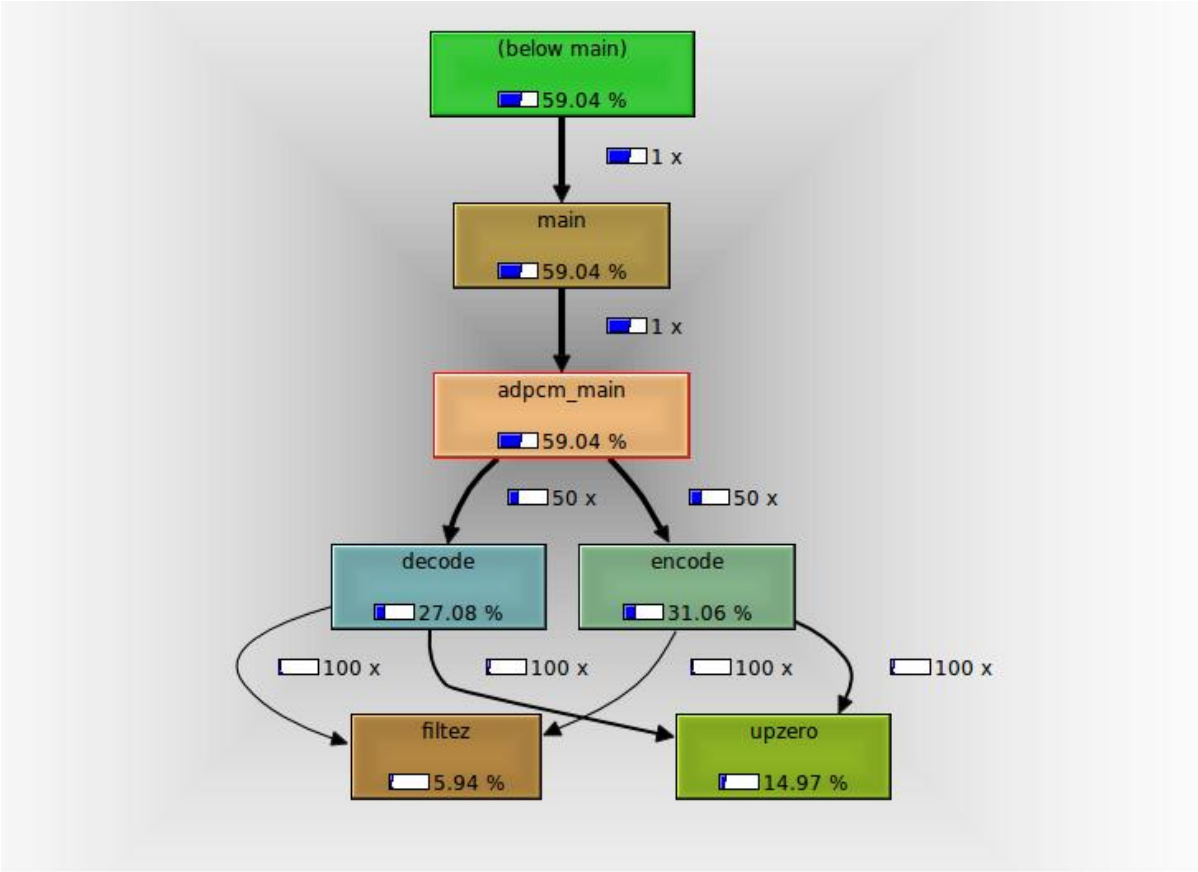

Figure III.5 The called-graphs of **ADPCM**.

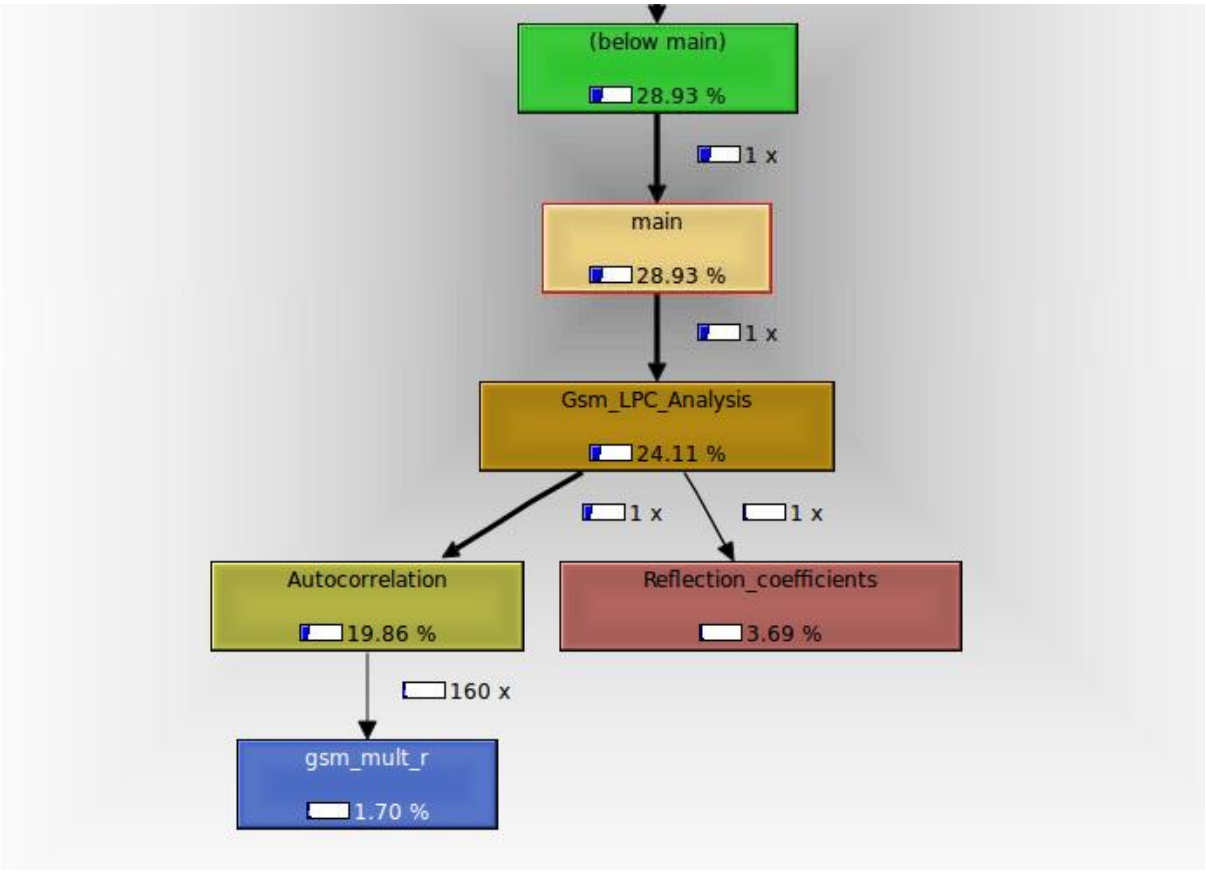

Figure III.6 The called-graphs of **GSM**.

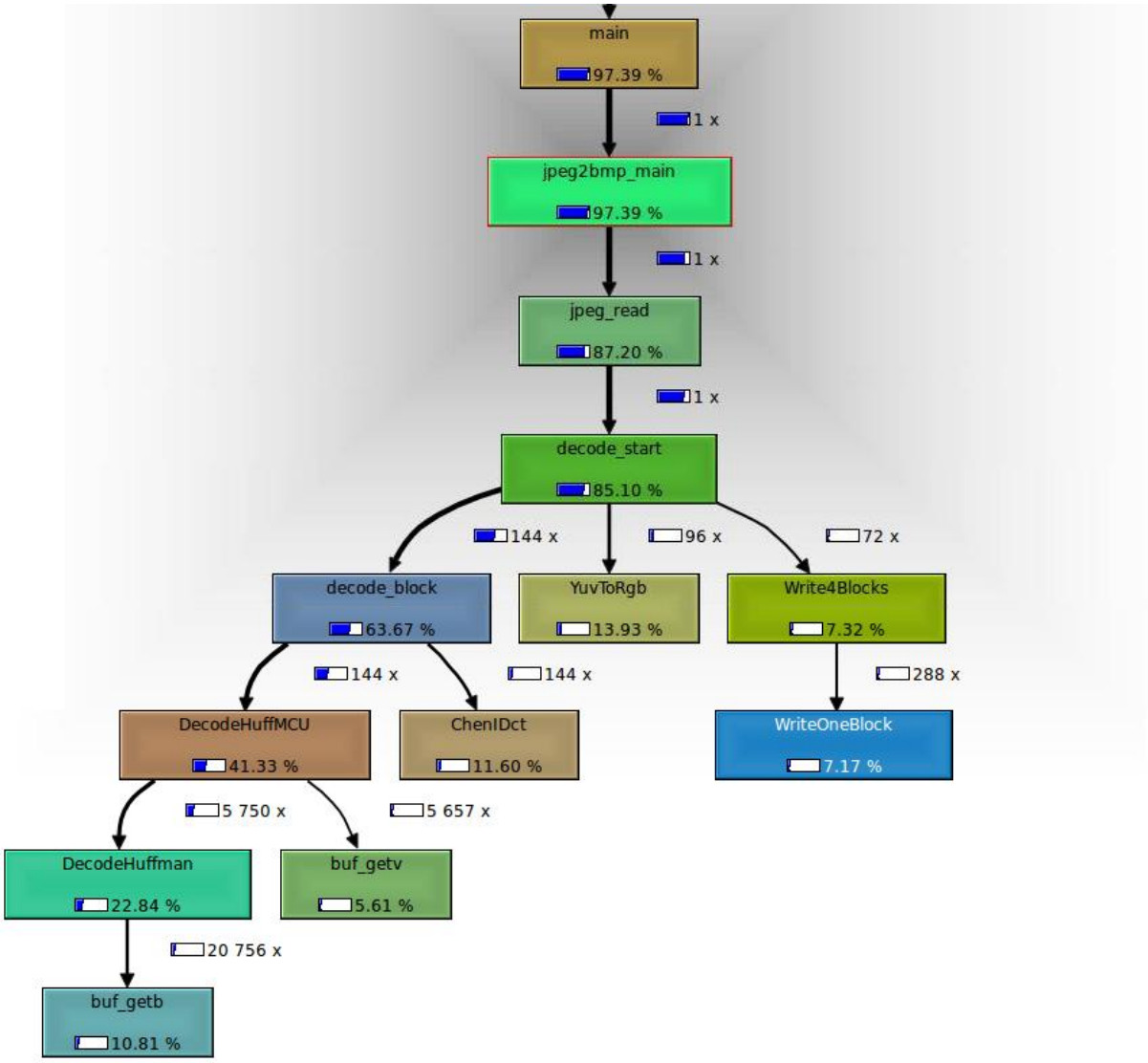

Figure III.7 The called-graphs of **JPEG**.

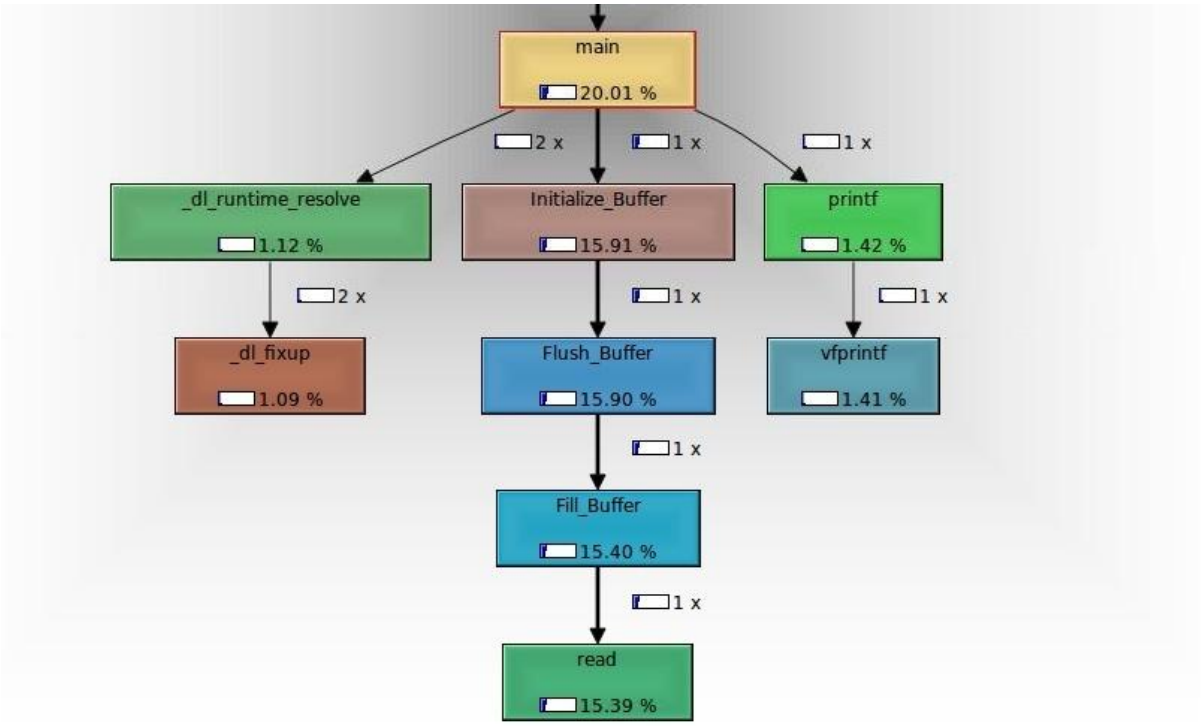

Figure III.8 The called-graphs of **MOTION**.

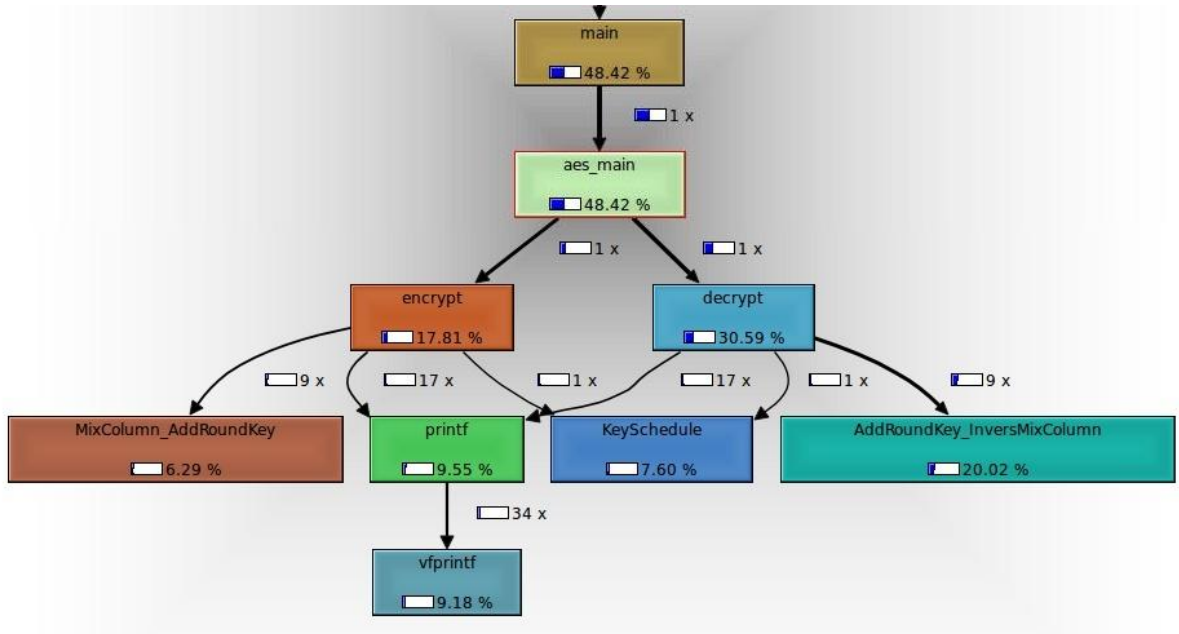

Figure III.9 The called-graphs of **AES**.

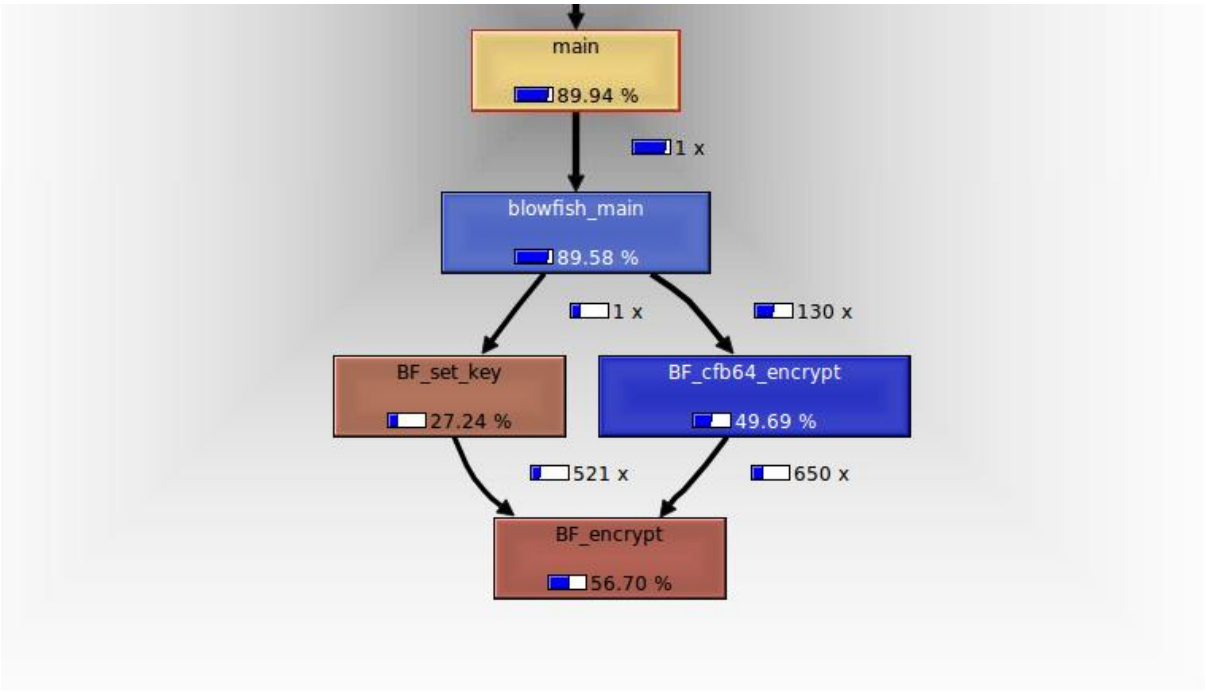

Figure III.10 The called-graphs of **BLOWFISH**.

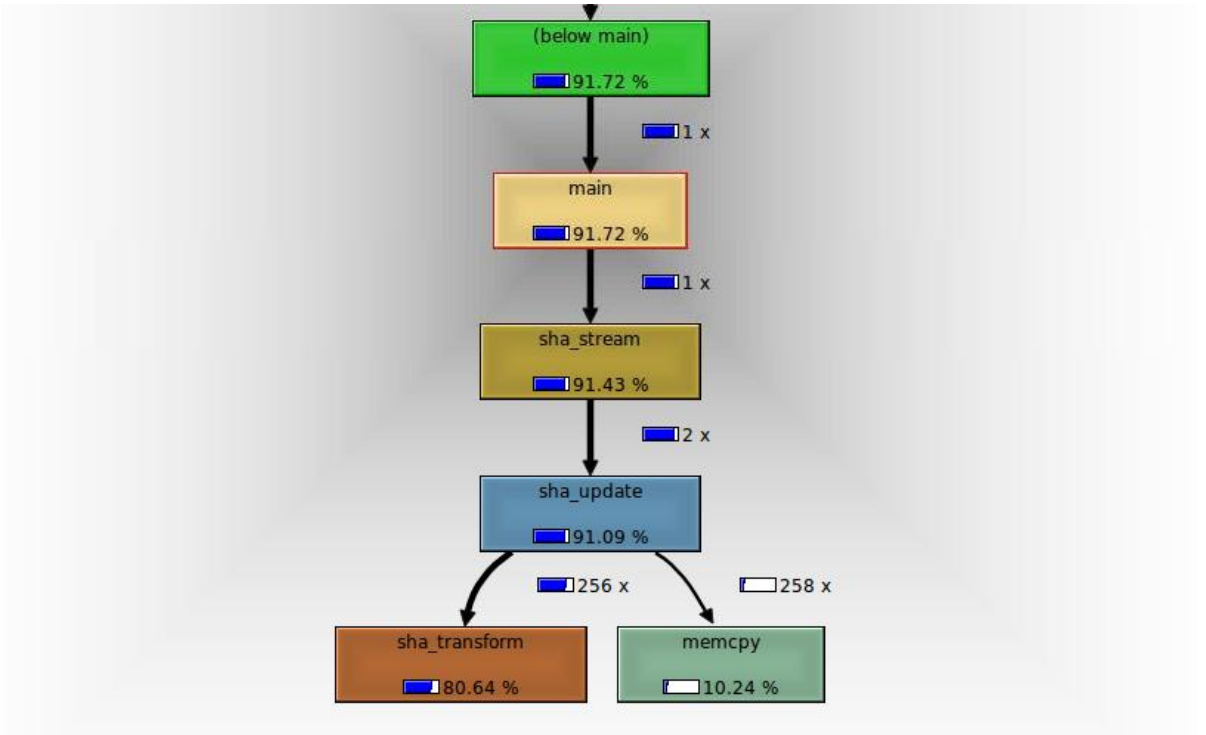

Figure III.11 The called-graphs of **SHA**.

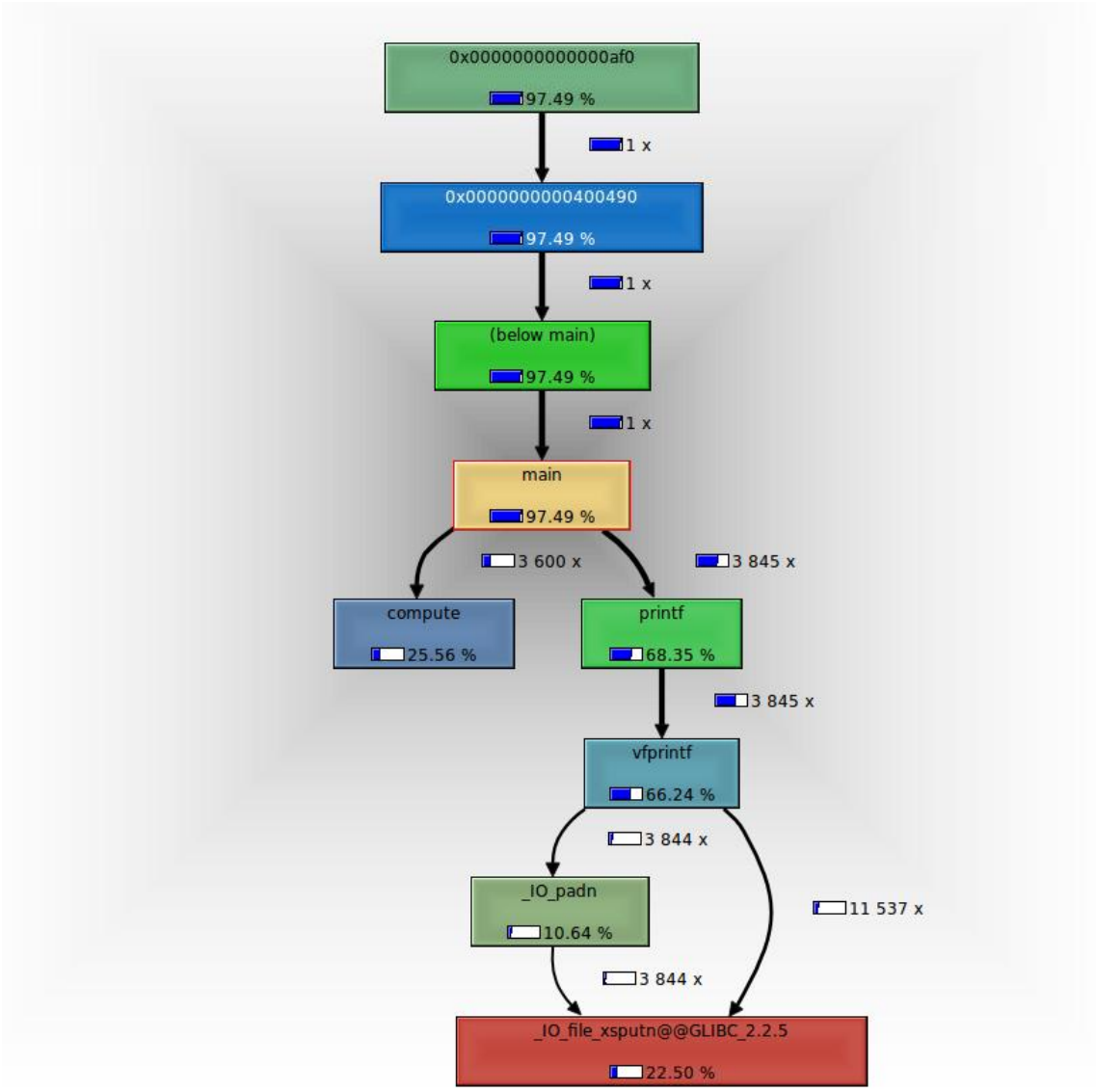

Figure III.12 The called-graphs of **CM60x60**.

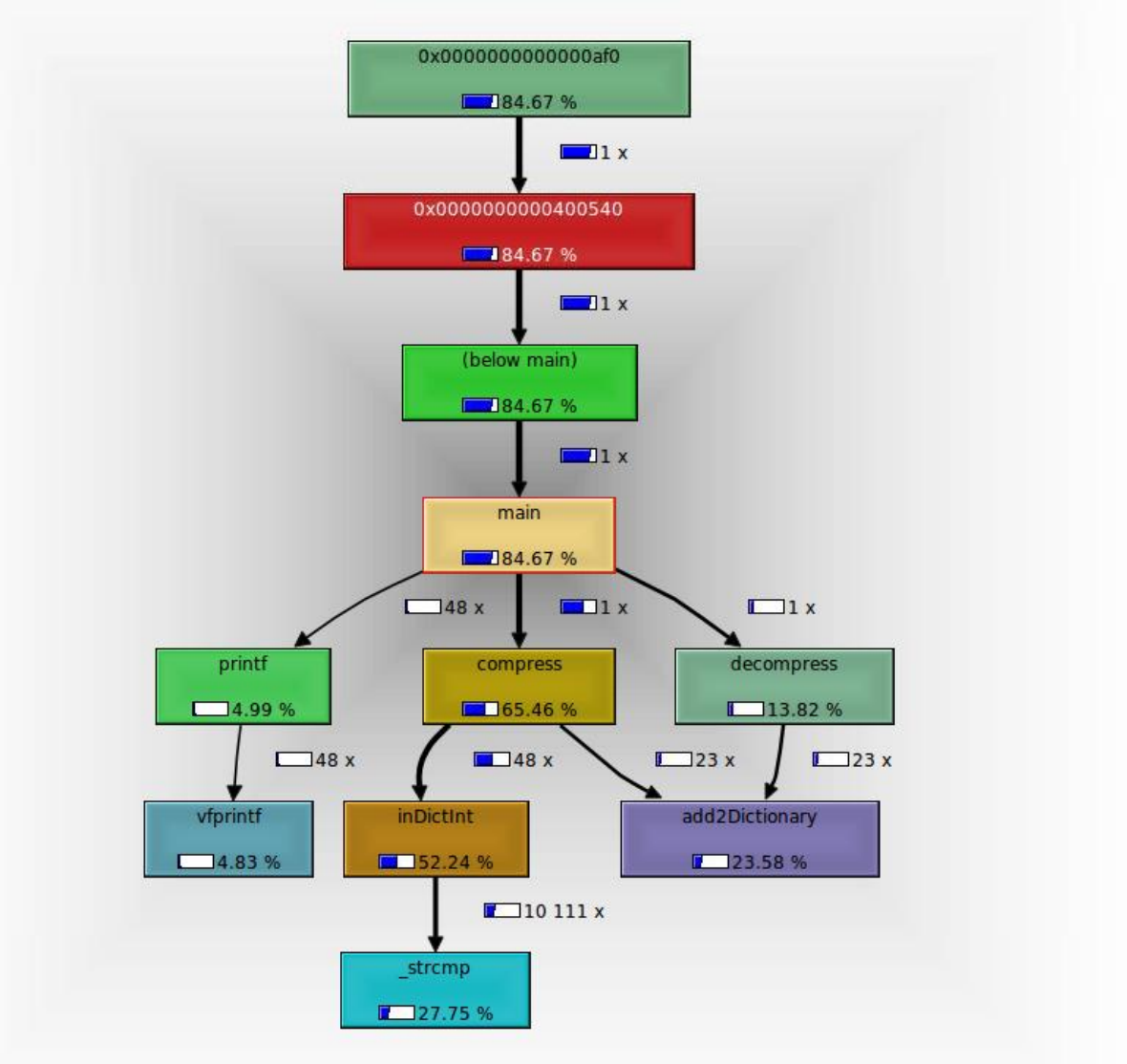

Figure III.13 The called-graphs of **LZW45K**.

III.2 Top Most Called-functions for Benchmarks

	Function1 (Incl%)	**Function2 (Incl%)**	**Function3 (Incl%)**	**Function4 (Incl%)**
DFADD	float64_add (2.98) ˅	subFloat64Sigs (1.23) ˅	addFloat64Sigs (1.17)	roundAndPackFloat64 (0.38)
DFDIV	float64_div (3.07) ˅	roundAndPackFloat64 (0.38)	extractFloat64Exp (0.17) ˅	extractFloat64Exp (0.15)
DFMUL	float64_mul (1.84) ˅	roundAndPackFloat64 (0.38)	extractFloat64Exp (0.16) ˅	extractFloat64Frac (0.14)
DFSIN	sin (57.58) ˅	float64_div (18.50) ˅	float64_mul (15.03)	float64_add (14.15)
ADPCM	adpcm_main (59.04)	encode (31.06) ˅	decode (27.08) ˅	upzero (14.97)
GSM	Gsm_LPC_Analysis (24.1) ˅	Autocorrelation (19.86) ˅	Reflection_coefficients (3.69)	Quantization_and_coding (0.39)
JPEG	jpeg2bmp_main (97.39)	jpeg_read (87.20) ˅	decode_start (85.10) ˅	decode_block (63.67)
MOTION	Flush_Buffer (16.34) ˅	Initialize_Buffer (15.91) ˅	Fill_Buffer (15.82)	read (15.81)
AES	aes_main (48.42)	decrypt (30.59) ˅	AddRoundKey_InversMixColumn (20.02) ˅	encrypt (17.81)
BLOWFISH	Blowfish_main (89.94)	BF_encrypt (56.70) ˅	BF_cfb64_encrypt (49.69) ˅	BF_set_key (27.24)
SHA	sha_stream (91.43) ˅	sha_update (91.09)	sha_transform (80.96) ˅	Memcpy (10.24)
CM60x60	compress (65.46) ˅	inDictInt (52.24)	add2Dictionary (23.58)	decompress (13.82) ˅
LZW45K	compress (65.46) ˅	inDictInt (52.24)	add2Dictionary (23.58)	decompress (13.82) ˅

Table III.A The top-four function-called in HLS benchmarks.

III.3 HW, Hybrids, and SW Flow in LegUp

Programs	Line of the Codes	Function	Scalar Variable	Array Variable	Add/Sub	MUL	DIV	Comparison	Shift	Logic
MIPS	232	1	30	5	17	2	0	12	22	23
DFADD	494	17	121	4	36	0	0	72	65	129
DFDIV	419	19	110	4	45	8	2	50	56	65
DFMUL	363	16	91	4	28	4	0	34	41	55
DFSIN	789	31	285	3	136	17	2	181	214	310
ADPCM	547	15	268	26	156	69	2	73	81	24
GSM	380	12	149	10	250	53	0	109	44	41
JPEG	1397	31	393	46	1038	148	6	243	293	132
MOTION	441	13	274	12	844	0	0	444	350	166
AES	723	11	345	11	510	22	12	48	758	370
BLOWFISH	1413	6	110	12	280	0	0	15	159	370
SHA	1286	8	64	6	133	0	3	32	59	87
CM60x60	47	2	7	2	11	1	0	0	0	0
LZW45K	153	10	20	8	8	0	0	6	0	4

Table III.B Characterizations of HLS programs.

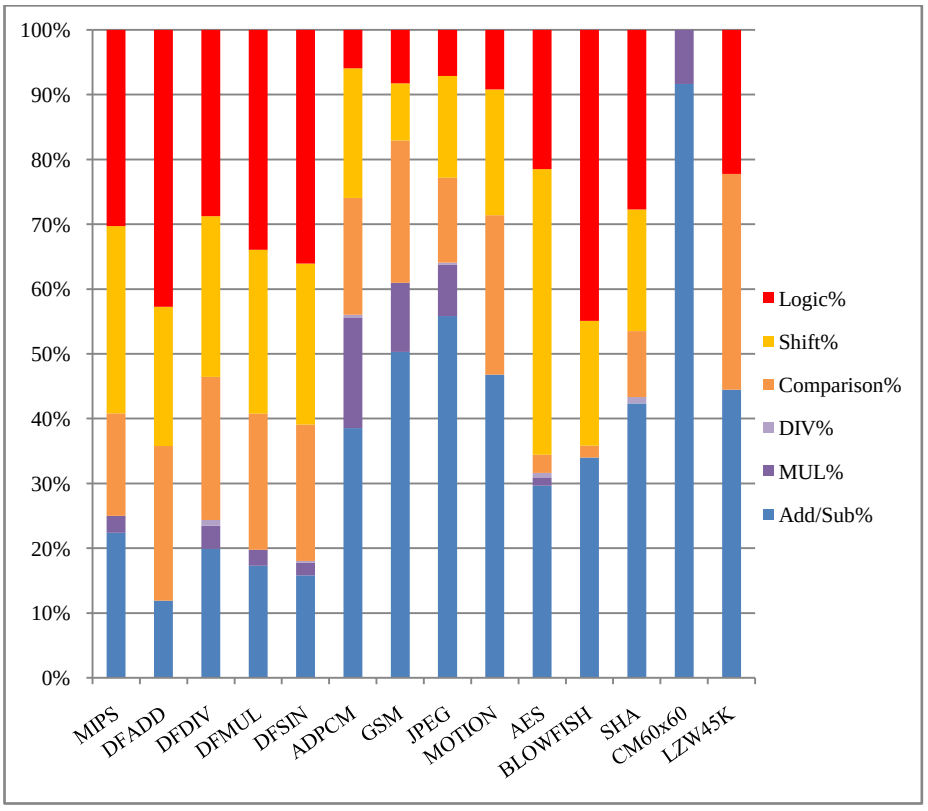

Figure III.A The percentage distributions of algorithmic types in HLS benchmarks.

Programs	if	Switch	while	for	goto	Test Data Length
MIPS	4	3	1	3	0	8
DFADD	52	24	0	25	0	46
DFDIV	97	0	1	17	30	22
DFMUL	214	64	27	90	75	20
DFSIN	351	0	191	6	36	36
ADPCM	28	10	0	24	0	100
GSM	6	8	5	5	0	160
JPEG	3	0	9	20	0	7,506
MOTION	4	3	1	3	0	2,048
AES	52	24	0	25	0	16
BLOWFISH	97	0	1	17	30	5,200
SHA	214	64	27	90	75	8,192
CM60x60	0	0	0	6	0	3,844
LZW45K	6	0	2	4	0	45,000

Table III.C Control flows and test-data length of HLS benchmarks.

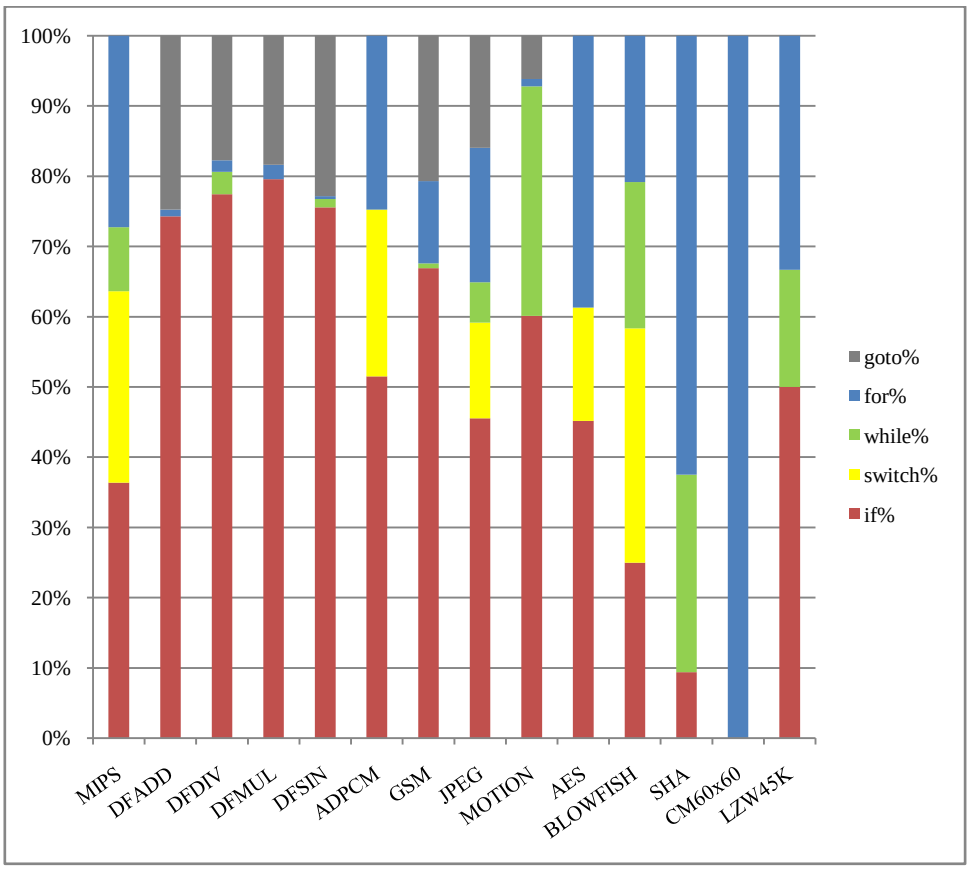

Figure III.B The percentage distributions of C control flows in HLS benchmarks.

Programs	Flows	Fmax/MHz	LEs (%)	Devices	REGs (%)	# bits (%)	MULs (%)	Times/μs
MIPS	*HW*	106	3,256 (39)	EP2C8T144C6	1555 (19)	4,480 (3)	8 (22)	10.369
	Hybrid1	-	-	-	-	-	-	-
	Hybrid 2	-	-	-	-	-	-	-
	SW	74.26	13,203 (40)	EP2C35F484C6	5975 (18)	301,477 (62)	16 (23)	783.062
DFADD	*HW*	139	6,233 (43)	EP2C15AF256C6	2980 (21)	17,056 (7)	0 (0)	1.611
	Hybrid1	139	23,976 (47)	EP2C50F484C6	11104 (33)	311,449	16 (23)	216.470
	Hybrid 2	139	19,236 (58)	EP2C35F484C6	9234 (28)	310,041 (64)	16 (23)	361.927
	SW	74.26	13,203 (40)	EP2C35F484C6	5975 (18)	301,477 (62)	16 (23)	6021.711
DFDIV	*HW*	2	13,357 (71)	EP2C20F256C6	9432 (50)	13,495 (6)	38 (73)	4.515
	Hybrid1	2	27,461 (54)	EP2C50F484C6	15001 (30)	311,120 (52)	54 (31)	200.714
	Hybrid 2	2	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	1109.128
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	3986.587
DFMUL	*HW*	66	3,927 (48)	EP2C8T144C6	1989 (24)	12,032 (7)	32 (89)	0.585
	Hybrid1	66	18,037 (54)	EP2C35F484C6	8859 (26)	310,041 (64)	48 (69)	171.620
	Hybrid 2	66	12,843 (39)	EP2C35F484C6	6296 (19)	301,563 (62)	16 (23)	283.017
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	2587.865
DFSIN	*HW*	2	25,853 (78)	EP2C35F484C6	15127 (46)	13,911 (3)	70 (100)	128.869
	Hybrid1	2	46,177 (67)	EP2C70F672C6	23023 (34)	311,056 (27)	92 (31)	1117.682
	Hybrid 2	2	29,261 (58)	EP2C50F484C6	16710 (33)	311,056 (52)	54 (31)	5079.276
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	40790.120
ADPCM	*HW*	8	17,494 (53)	EP2C35F484C6	9764 (29)	27,646 (6)	61 (87)	63.049
	Hybrid1	88	22,969 (69)	EP2C35F484C6	10569 (32)	319,449 (66)	26 (37)	1944.456
	Hybrid 2	88	22,185 (67)	EP2C35F484C6	8810 (27)	319,449 (66)	32 (46)	2170.890
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	2338.421
GSM	*HW*	88	10,999 (59)	EP2C20F256C6	5473 (29)	10,144 (4)	20 (38)	10.719
	Hybrid1	88	25,890 (51)	EP2C50F484C6	12234 (24)	309,657 (52)	40 (23)	304.922
	Hybrid 2	88	21,096 (64)	EP2C35F484C6	10421 (31)	309,081 (64)	36 (51)	426.667
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	772.363
JPEG	*HW*	8	34,812 (51)	EP2C70F672C6	16440 (24)	470,054 (41)	54 (18)	2681.559
	Hybrid1	8	65,043 (95)	EP2C70F672C6	28090 (41)	517,527 (45)	56 (19)	23025.867
	Hybrid 2	8	42,642 (62)	EP2C70F672C6	19463 (28)	500,445 (43)	50 (17)	27478.458
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	73564.303
MOTION	*HW*	153	4,360 (53)	EP2C8T144C6	2180 (26)	33,312 (20)	8 (22)	12.761
	Hybrid1	153	15,886 (48)	EP2C35F484C6	7237 (22)	319,385 (66)	20 (29)	305.949
	Hybrid 2	153	15,655 (47)	EP2C35F484C6	7404 (22)	319,385 (66)	20 (29)	313.671
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	632.821
AES	*HW*	8	16,135 (49)	EP2C35F484C6	7945 (24)	38,632 (8)	0 (0)	31.435
	Hybrid1	8	44,451 (65)	EP2C70F672C6	18832 (27)	325,241 (28)	22 (7)	650.839
	Hybrid 2	8	19,389 (38)	EP2C50F484C6	8925 (18)	321,867 (54)	22 (13)	704.854
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	1270.939
BLOWFISH	*HW*	215	9,514 (66)	EP2C15AF256C6	6077 (42)	150,240 (63)	0 (0)	395.959
	Hybrid1	215	20,725 (41)	EP2C50F484C6	10351 (20)	418,329 (70)	16 (9)	8156.948
	Hybrid 2	215	23,396 (46)	EP2C50F484C6	11418 (22)	418,393 (70)	16 (9)	9876.354
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	13409.624
SHA	*HW*	8	12,842 (39)	EP2C35F484C6	7344 (22)	135,048 (28)	4 (6)	467.753
	Hybrid1	8	27,732 (41)	EP2C70F672C6	13574 (20)	436,321 (38)	20 (7)	3345.570
	Hybrid 2	8	17,836 (35)	EP2C50F484C6	8869 (17)	435,641 (73)	16 (9)	4588.006
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	16037.925
CM60x60	*HW*	153	1,475 (32)	EP2C5T144C6	411 (9)	61,832 (52)	1 (4)	1719.723
	Hybrid1	-	-	-	-	-	-	-
	Hybrid 2	-	-	-	-	-	-	-
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	32296.661
LZW45K	*HW*	215	14,456	-	4075	1,605,384	52	-
	Hybrid1	215	13,203 (40)	EP2C35F484C6	5975 (18)	301,477 (62)	16 (23)	5625482.233
	Hybrid 2	215	19,644 (59)	EP2C35F484C6	8620 (26)	366,385 (76)	31 (44)	12047403.128
	SW	74.26	12,759 (38)	EP2C35F484C6	5974 (18)	301,477 (62)	16 (23)	30571490.632

Table III.D Synthesis results for HLS benchmarks on ALTERA CYCLONE II FPGAs.

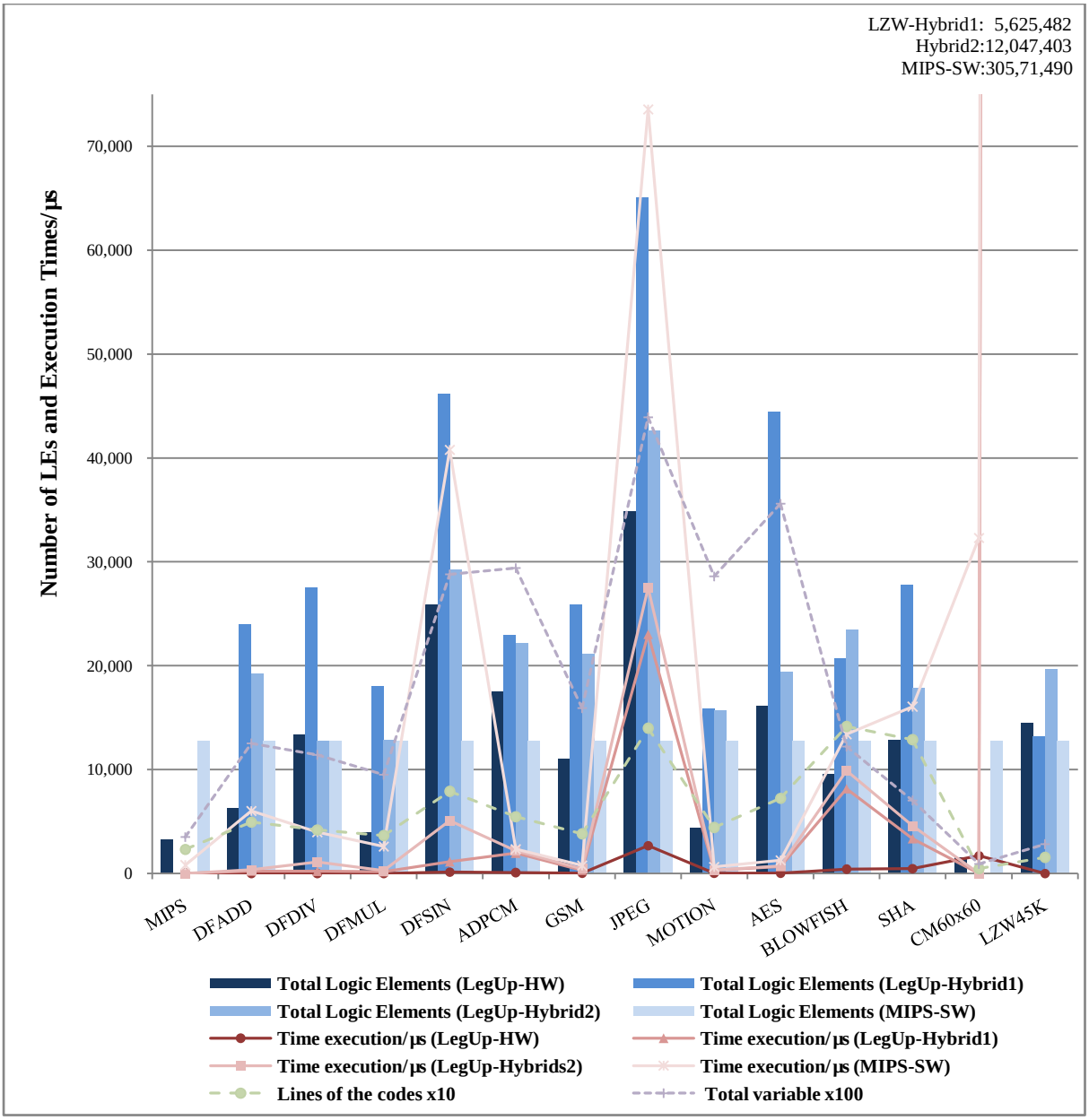

Figure III.C The execution times and resource distribution results on ALTERA CYCLONE II FPGAs.

Program	Flows	Speed-up	Area Cost	Speed&Area
MIPS	*HW*	75.52	0.26	295.93
	Hybrid1	-	-	-
	Hybrid 2	-	-	-
DFADD	*HW*	3737.87	0.49	7651.45
	Hybrid1	27.82	1.88	14.80
	Hybrid 2	16.64	1.51	11.036
DFDIV	*HW*	882.97	1.05	843.43
	Hybrid1	19.86	2.15	9.23
	Hybrid 2	3.59	1.00	3.59
DFMUL	*HW*	4423.70	0.31	14372.80
	Hybrid1	15.08	1.41	10.67
	Hybrid 2	9.14	1.01	9.08
DFSIN	*HW*	316.52	2.03	156.21
	Hybrid1	36.50	3.62	10.08
	Hybrid 2	8.03	2.29	3.50
ADPCM	*HW*	37.09	1.37	27.05
	Hybrid1	1.20	1.80	0.67
	Hybrid 2	1.08	1.74	0.62
GSM	*HW*	72.06	0.86	83.59
	Hybrid1	2.53	2.03	1.25
	Hybrid 2	1.81	1.65	1.09
JPEG	*HW*	27.43	2.73	10.05
	Hybrid1	3.19	5.10	0.63
	Hybrid 2	2.68	3.34	0.80
MOTION	*HW*	49.59	0.34	145.12
	Hybrid1	2.07	1.25	1.66
	Hybrid 2	2.02	1.23	1.64
AES	*HW*	40.43	1.26	31.97
	Hybrid1	1.95	3.48	0.56
	Hybrid 2	1.80	1.52	1.19
BLOWFISH	*HW*	33.87	0.75	45.42
	Hybrid1	1.64	1.62	1.01
	Hybrid 2	1.36	1.83	0.74
SHA	*HW*	34.29	1.01	34.07
	Hybrid1	4.79	2.17	2.21
	Hybrid 2	3.50	1.40	2.50
CM60x60	*HW*	18.78	0.12	162.45
	Hybrid1	-	-	-
	Hybrid 2	-	-	-
LZW45K	*HW*	-	-	-
	Hybrid1	5.43	1.03	5.25
	Hybrid 2	2.54	1.54	1.65

Table III.E The results of speed-up versus resource cost and Speed&Area Efficiency of LegUp.