National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science
Microprocessors and Digital Systems Laboratory

Cross-Layer Rapid Prototyping and Synthesis of Application-Specific and Reconfigurable Many-accelerator Platforms

Dissertation

Dionysios Diamantopoulos

Athens, 2015

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science
Microprocessors and Digital Systems Laboratory

Cross-Layer Rapid Prototyping and Synthesis of Application-Specific and Reconfigurable Many-accelerator Platforms

Dissertation

of

Dionysios Diamantopoulos

Submitted to the
School of Electrical and Computer Engineering
of National Technical University of Athens
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Athens, 2015

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science
Microprocessors and Digital Systems Laboratory

Cross-Layer Rapid Prototyping and Synthesis of Application-Specific and Reconfigurable Many-accelerator Platforms

Advisory Committee

.....	.....	.....
Dimitrios Soudris	Kiamal Pekmestzi	George Economakos
Assoc. Professor. N.T.U.A..	Professor N.T.U.A.	Assis. Professor N.T.U.A.

Promotion Committee

.....	.....	.....
Dimitrios Soudris	Kiamal Pekmestzi	George Economakos
Assoc. Professor N.T.U.A.	Professor N.T.U.A.	Assis. Professor N.T.U.A.

.....	.....
George Theodoridis	Dionysios Reisis
Assis. Professor U. Patras	Assoc. Professor N.K.U.A.

.....	.....
Michael Hübner	Dionisios Pnevmatikatos
Professor RUB	Professor T.U.C.

. . . . . .

Dionysios Diamantopoulos

PhD, School of Electrical and Computer Engineering
National Technical University of Athens, Greece
Diploma, Computer Engineering & Informatics Department
Polytechnic School, University of Patras, Greece

This dissertation was co-financed by the research programs of European Space Agency (ESA), "SPAring Robotics Technologies For Autonomous Navigation (SPARTAN)" (ESA / ESTEC ITT Refe-rence AO / 1-6512 / 10 / NL / EK), "Spartan EXTension Activity" (SEXTANT) (ESA / ESTEC ITT Reference 4000103357 / 11 / NL / EK) and "Code Optimisation Modication Partitioning" (COMPASS) (ESA / ESTEC ITT Reference). Also form European research programs FP7-248716 2PARMA and FP7-215244 MOSART. Finally, part of this dissertation was partially financed from national resources and the National Strategic Reference Framework (NSRF) 2007-2013 for the project "Next Generation Millimeter Wave Backhaul Radio".

Electronic version of this dissertation is available online at:
http://nestor.microlab.ntua.gr/

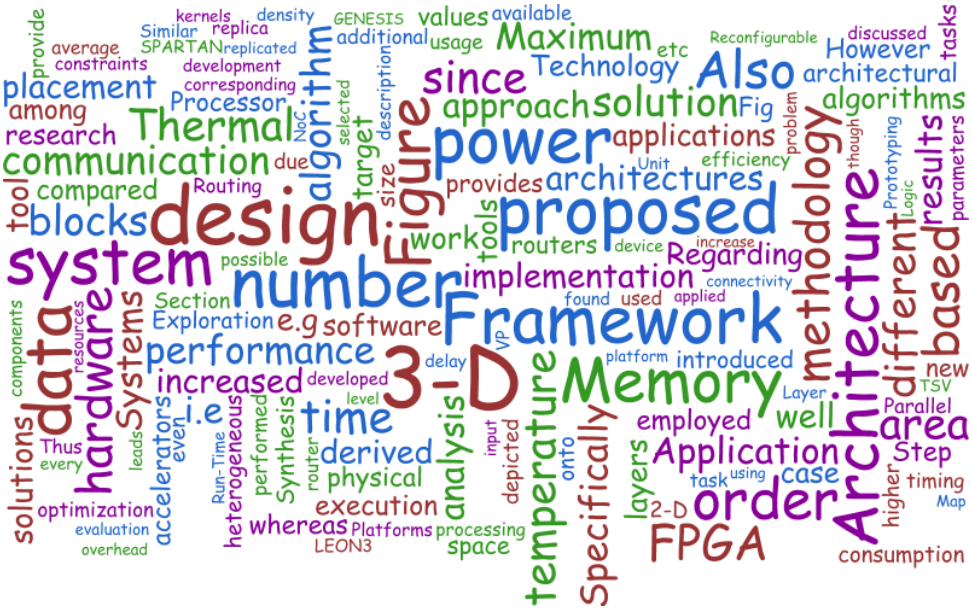


Figure 1 Thesis word cloud, after lexical analysis with *Detex v2.6* & *IBM Word Cloud build 32*.

Abstract

Technological advances of recent years laid the foundation consolidation of informatisation of society, impacting on economic, political, cultural and social dimensions. At the peak of this realization, today, more and more everyday devices are connected to the web, giving the term "Internet of Things". The future holds the full connection and interaction of IT and communications systems to the natural world, delimiting the transition to natural cyber systems and offering meta-services in the physical world, such as personalized medical care, autonomous transportation, smart energy cities etc. . Outlining the necessities of this dynamically evolving market, computer engineers are required to implement computing platforms that incorporate both increased systemic complexity and also cover a wide range of meta-characteristics, such as the cost and design time, reliability and reuse, which are prescribed by a conflicting set of functional, technical and construction constraints. This thesis aims to address these design challenges by developing methodologies and hardware/software co-design tools that enable the rapid implementation and efficient synthesis of architectural solutions, which specify operating meta-features required by the modern market. Specifically, this thesis presents a) methodologies to accelerate the design flow for both reconfigurable and application-specific architectures, b) coarse-grain heterogeneous architectural templates for processing and communication acceleration and c) efficient multi-objective synthesis techniques both at high abstraction level of programming and physical silicon level.

Regarding to the acceleration of the design flow, the proposed methodology employs virtual platforms in order to hide architectural details and drastically reduce simulation time. An extension of this framework introduces the systemic co-simulation using reconfigurable acceleration platforms as co-emulation intermediate platforms. Thus, the development cycle of a hardware/software product is accelerated by moving from a vertical serial flow to a circular interactive loop. Moreover the simulation capabilities are enriched with efficient detection and correction techniques of design errors, as well as control methods of performance metrics of the system according to the desired specifications, during all phases of the system development. In orthogonal correlation with the aforementioned methodological framework, a new architectural template is proposed, aiming at bridging the gap between design complexity and technological productivity using specialized hardware accelerators in heterogeneous systems-on-chip and network-on-chip platforms. It is presented a novel co-design methodology for the hardware accelerators and their respective programming software, including the tasks allocation to the available resources of the system/network. The introduced framework provides implementation techniques for the accelerators, using either conventional programming flows with hardware description language or abstract programming model flows, using techniques from high-level synthesis. In any case, it is provided the option of systemic measures optimization, such as the processing speed, the throughput, the reliability, the power consumption and the design silicon area. Finally, on addressing the increased complexity in design tools of reconfigurable systems, there are proposed novel multi-objective optimization evolutionary algo-

rithms which exploit the modern multicore processors and the coarse-grain nature of multithreaded programming environments (e.g. OpenMP) in order to reduce the placement time, while by simultaneously grouping the applications based on their intrinsic characteristics, the effectively explore the design space effectively.

The efficiency of the proposed architectural templates, design tools and methodology flows is evaluated in relation to the existing edge solutions with applications from typical computing domains, such as digital signal processing, multimedia and arithmetic complexity, as well as from systemic heterogeneous environments, such as a computer vision system for autonomous robotic space navigation and many-accelerator systems for HPC and workstations/datacenters. The results strengthen the belief of the author, that this thesis provides competitive expertise to address complex modern - and projected future - design challenges.

Contents

List of Figures

List of Tables

Nomenclature

Roman Symbols

$3SLOG$	three (3) Step LOGarithmic search
$ADPCM$	Adaptive Differential Pulse-Code Modulation
AHB	Advanced High-performance Bus
$AMBA$	Advanced Micro-controller Bus Architecture
APB	Advanced Peripheral Bus
API	Application Programming Interface
$ASIC$	Application Specific Integrated Circuit
BLE	Basic Logic Element
CAD	Computer Aided Design
CLB	Configurable Logic Block
CPS	Cyber Physical Systems
CRC	Cyclic Redundancy Check
CV	Computer Vision
DMA	Direct Memory Access
DMM	Dynamic Memory Management
DSP	Digital Signal Processor
DSU	Debug Support Unit
DUT	Design Under Test
$DVFS$	Dynamic Voltage and Frequency Scaling
ECO	Engineering change orders
ECU	Engine Control Unit
EDA	Electronic Design Automation
ESL	Electronic System Level

$ESoC$ Ecosystem-on-a-chip

FFT Fast Fourier Transform

$FIFO$ First-In, First-Out

FS Full Search

FSB Front-Side Bus

FSM Finite-State Machine

HDL Hardware Description Language

HPC High Performance Computing

$HPWL$ Half-perimeter Wirelength

HS Hierarchical Search

HT HyperTransport

IC Integrated Circuit

IDW Inverse Distance Weighting

IP Intellectual Property

IPC Inter-Process Communication

ISA Instruction Set Architecture

ISS Instruction Set Simulator

$ITRS$ International Technology Roadmap for Semiconductors

LAB Logic Array Block

LUT Look-up Table

$MTTF$ Mean Time To Failure

MTU Maximum Transmission Unit

$NASA$ National Aeronautics and Space Administration

NIC Network Interface Controller

$NUCA$ Non-Uniform Cache Architecture

$PHODS$ Parallel Hierarchical One-Dimensional Search

$POSIX$ Portable Operating System Interface for Unix

QoR Quality of Results

QPI	Intel QuickPath Interconnect
ROS	Robotic Operating System
RTL	Register-Transfer Level
RTS	Run-Time Situation
SDR	Software Defined Radio
SiP	System-in-package
$SLAM$	Simultaneous Localization And Mapping
SNR	Signal-to-Noise Ratio
SoC	System-on-chip
$SPEF$	Standard Parasitic Exchange Format
SS	Spiral Search
$SURF$	Speeded Up Robust Features
$T-VPACK$	Timing-Driven Versatile Packing
$TDDB$	Time-Depended Dielectric Breakdown
TDP	Thermal Design Power
TSV	Through Silicon Via
VCD	Value Change Dump
VPR	Versatile Placement and Routing

1

Introduction

1.1. Thesis Research Background

Integrated circuits (IC) are the cornerstone of developments in both the inter-scientific community and the daily life of modern man. The impact of these developments are so strong, that the technological transitions over time are taken for granted. Consumers have come to expect more and more sophisticated electronic products, while the business world expects increased productivity through improved technology information systems. At the same time, approaching the macro scale of this trend, it seems that the maturity of the semiconductor industry is the catalyst productivity and growth in almost all sectors of economic activity, which already marks a market of about 3 trillion dollars [1], as depicted in Figure 5.19.

According to predictions [2], this impressive market promises to connect 2.67 billion machines to the Internet till the end of 2017 (from 1.11 today), exclusively as far as the upcoming "smart cities" are concerned, and 25 billion machines totally as far as the society, culture and finance are concerned till the end of 2020 (today 4,8). It is clear that these loud numbers are fueled by the scalability assurance of semiconductor technology regarding the market needs.

The last 50 years of - nearly linear - increase of the computational power and the corresponding decrease of the power consumption, support this belief. Nevertheless this linearity has been supplied by the association of two technological disciplines: Moore's law [3] and Dennard's law [4]. The first one which is valid till today [1], promises the doubling of the number of the elementary resources (for example transistors) which constitute a whole system in every 18 months. Practically, this law allows the increase of the computational power in each generation of technological completion. The second law, which expired in 2007, described the dimensions and the electrical characteristics of a transistor so that the sequential shrinkage of them to be possible, so as to improve at the same time the density

[1] since the interpretation of Moore's law is not unique, there is evidence that this law will not be valid soon or has been already invalid [5, 6].

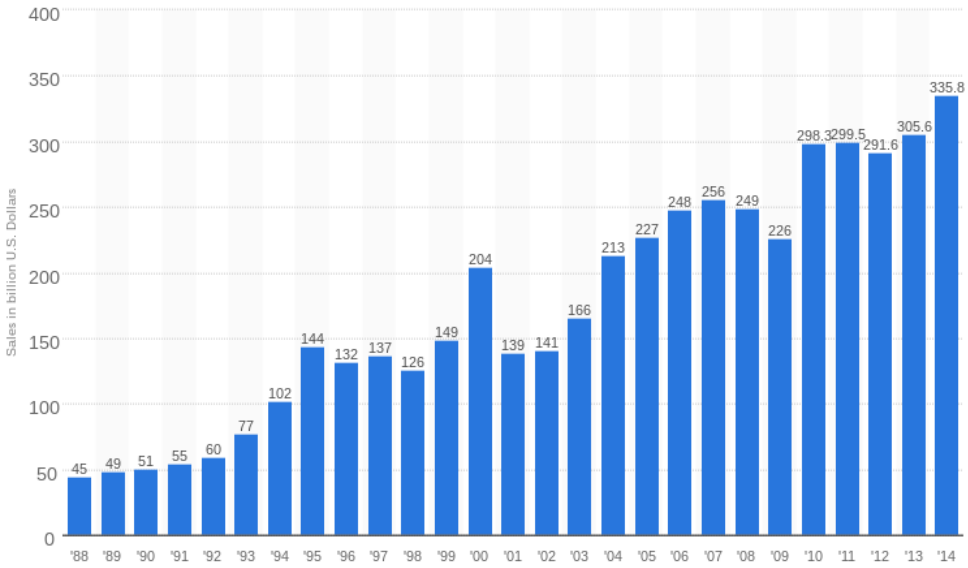


Figure 1.1 Worldwide semiconductor sales 1988-2014 (in billions dollars). Source: World Semiconductor Trade Statistics, [1].

of the silicon area, the processing frequency and the power performance. Practically, this law allowed the power consumption to be stable in each transition of new generation of completion during which the number of transistors was doubled. Also, due to the technology progress this law foresaw increase of the processing frequency till 40%. [7].

The combination of these two laws, allowed for almost four decades the increase of the supply voltage and the threshold voltage as much as the scale integration of the size of transistors. In that way the designers were allowed to decrease the supply power per transistor, so that the power density (power consumption per silicon area) remained almost stable, scaling from one technological node to another.

Nevertheless, on nodes of several nanometers (deep sub-micron technology nodes), the decrease of the threshold voltage leads to an exponential increase of the leakage power. Figure 5.17 illustrates the scaling of static and dynamic power consumption of integrated circuits, with respect to the technology node. As shown, given a set of typical integrated circuits designs, the dynamic energy consumption in the technological node of 20 nm has almost surpassed the corresponding static. In this way, the threshold voltage is no longer scalable and as a result the supply voltage cannot be scaled further without affecting the performance. This decrease in performance is illustrated in Figure 5.20. In particular, this figure shows the forecasts of ITRS organization for the scaling of the maximum operating frequency at regular chronological studies. As shown, the projections of each new study is pessimistic regarding the scaling of processing frequency, indicating an expected

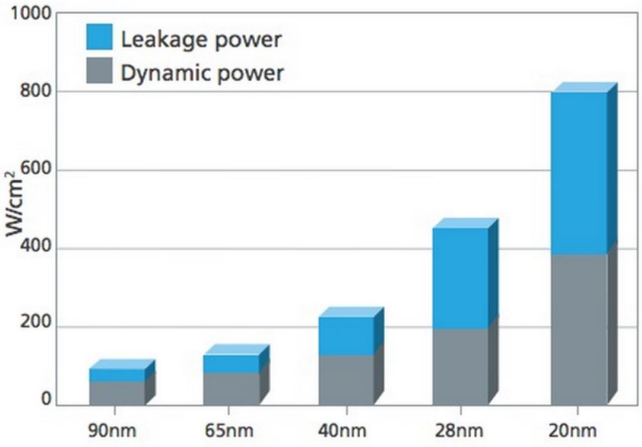


Figure 1.2 Scaling of static and dynamic power consumption of integrated circuits regarding to the technology node. Source: Mentor Graphics 2014.

increase of 41%/year in 2001, which dropped to a 4% in 2011.

As a result, although more transistors can be manufactured per silicon area, the supply power per transistor doesn't scale in analogy and as a result the power density increases. This phenomenon becomes more intense in conjunction with the natural limits imposed by the package materials and the cooling technology for the maximum power and the maximum power density. This description highlights the introduction of the "Dark Silicon" era [7–10].

Another limitation of the modern semiconductor industry refers to the economic cost of manufacturing integrated circuits. Modern lithography design methods demanded continuous innovative solutions for each new technology node (high-k, Metal Gate, Strain, SiGe, Tri-gate, etc.), while is is expected that new nodes shall require respective research and implementing innovations, increasing manufacturing costs [11]. Apart from the research community, also a lot of companies of the semiconductor industry have recognized this problem, e.g. Nvidia Inc. [12]. Figure 3.7 shows the typical manufacturing cost per gate for recent technology nodes. As shown, in the past (90 nm, 65 nm, 45 nm, 28 nm), the cost per gate at each node increases. At the same time, the number of gates per wafer increases. This contrast was almost linearly bijective, thus the average cost per wafer remained stable. However the technology node of 20 nm appears to increase the cost over previous technology generations. It is expected that the manufacturing of integrated circuits in newer nodes is disadvantageous compared to existing ones, or even earlier. The three-dimensional integration technology (3-D) and heterogeneity have been identified as promising solutions to the problem of scaling manufacturing costs [13].

The research community has already proposed a remarkable set of alternative approaches for the problem above, such as the heterogeneous multi-processing architectures [14–19], the approximate computing [20–26] and the energy man-

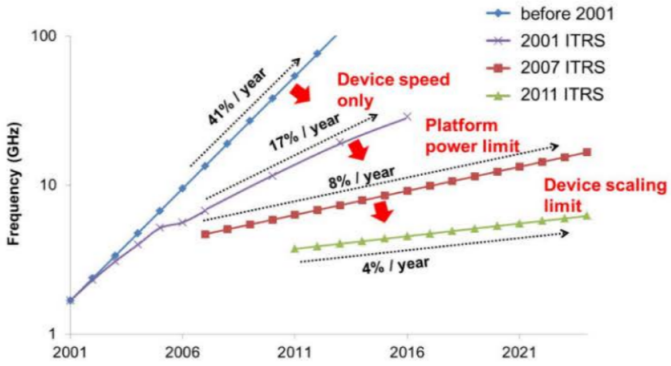


Figure 1.3 Projections of the ITRS organization for the scaling of the maximum operating frequency at regular chronological studies.

agement for architectures of "Dark" and "Dim" Silicon [27–33]. Given the recent published results, it seems that the most effective approach is the creation of heterogeneous and specific purpose accelerators which increase the computational power per energy consumption and silicon area. In that direction, through this work, four basic constraints are identified as major limiters of building the aforementioned architectures: a) the programming problem, b) the problem of software/hardware partitioning, c) the problem of cost/time design and d) the problem of finding adequate CAD tools of optimal design. Identifying these constraints, this thesis proposes architecture templates and design frameworks for the "Dark Silicon" era.

1.2. Dissertation Overview

The above introductory field emerges the need for methods, tools and architectures which provide viability during the continuous scaling of technology and energy. The background design of computer systems in recent years has shown that scaling in energy and technology can be achieved with the customization of the architecture and the materials/manufacturing technology, respectively.

Figure 5.12 depicts different architectural designs approaches, with respect to their energy efficiency. It is obvious that the specialized architectures can lead to increased CPU power per energy unit (MOPS/mW) by a factor of 100×, compared to conventional general purpose processors. Also, it is stressed that FPGAs devices are a flexible platform of energy efficiency, according to their programming. However, the transition to specialized and/or reconfigurable architectures corresponds to increased difficulty in design and programming of appropriate software.

Figure 3.9 shows the performance growth rate introduced by conventional technology scaling and architectural/materials innovation for every technology node. As shown, the performance improvement of recent technology nodes is attributed, with a small percentage, to the conventional technology scaling. However, the highest profit is attributed to innovative techniques of architecture improvements

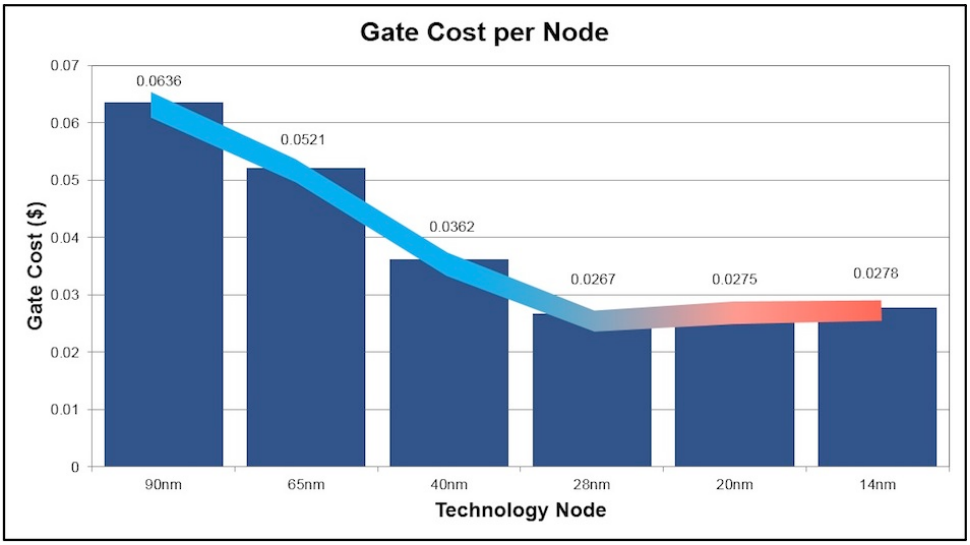


Figure 1.4 Scaling manufacturing costs per gate for different technology nodes of integration. Source: IBS 2012

and manufacturing materials.

This thesis is positioned towards the direction of the two aforementioned motivations, a holistic methodology framework of which is presented in Fig 5.14. It aims to capture a new idea, evolved in the base of a new product/hardware/software service. The starting point of the developed methodological components is the conceptual level, meaning that stage that captures the desired characteristics of a new product or service of a computer system. Continues with the mapping of the desired specifications of a developed product as well as the functional constraints. An important stage in this level is the organization of the tasks. Lately a large number of specific managing works tools has been developed in the frames of a software/hardware work, which depending on the complexity, can manage different phases of the production procedure like the estimation and design, the programming, the cost control, the budget management, the allocation of resources, the communication of colleagues, the decision making, the quality management and the organization of management and technical paperwork (π.χ. [34, 35]).

Having prescribe the required operating characteristics, the development team is asked to standardize the operating environment in which the under-development product will be operational. This level (Programming Environment) is the result of a new approach of the production process and it is responsible for the specification of all those environmental factors with which the developed product shall interact. The Concept Level with the Programming Environment constitute the Ecosystem Level which through early detection of the aforementioned characteristics of the final system, ensures adaptation of the design flow to the new trends of the market, where most growth is driven by the market itself and not necessarily from

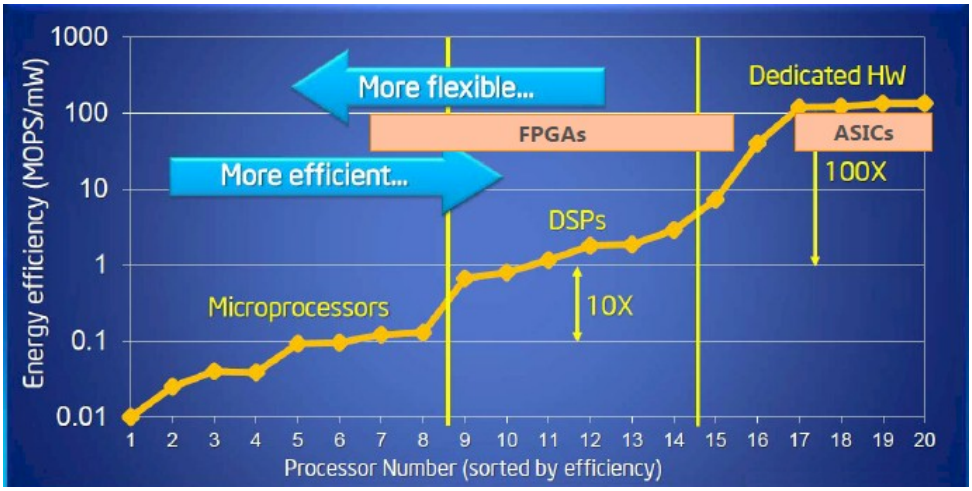


Figure 1.5 Comparison of different architectural designs approaches, with respect to their energy efficiency. Source: Bob Broderson, Berkeley Wireless group, ISSCC proceedings, Microsoft, 2011.

technological developments [36].

The next level is established by the design of the reference platform (Platform Level), which forms the representations of operating characteristics to the actual system. This stage starts by partitioning hardware/software based on a analysis process (profiling) that constitute the inherent characteristics of the application and based on the retrieved results, classifies the total operating tasks for execution with either software or dedicated hardware accelerating modules. The hardware/software co-design procedure is enriched with the proposed rapid prototyping technique, i.e. a process that ensures faster product development as it allows the development of software throughout the development of the hardware, making use of virtual hardware modules that simulate the specialized hardware. Compared to existing prototyping solutions, the proposed methodology is superior with respect to speed, accuracy and debugged design completion, since it introduces FPGAs devices in the process of simulation. This action allows part of the hardware developed in tandem with the software, being directly tested on original hardware (FPGA) in the early design stages, where the final system is not complete.

The hardware development takes place at the stage of architectural design. This step is orthogonal independent to the virtual prototyping phase, since the subsystems that are gradually synthesized, they can be used in virtual prototyping replacing their virtual versions. Thereby, the process of simulation is accelerated and the overall systems is evaluated with greater accuracy. The stage of architectural synthesis is decomposed onto three distinct methodological analysis, the Acceleration Datapath Synthesis, the System Architecture Exploration and the Synthesis Flow Optimization. Each of these steps addresses different problems of modern design of integrated circuits for both reconfigurable (FPGAs) and application-specific (ASICs) platforms. Specifically, the Acceleration Datapath Synthesis includes the accelera-

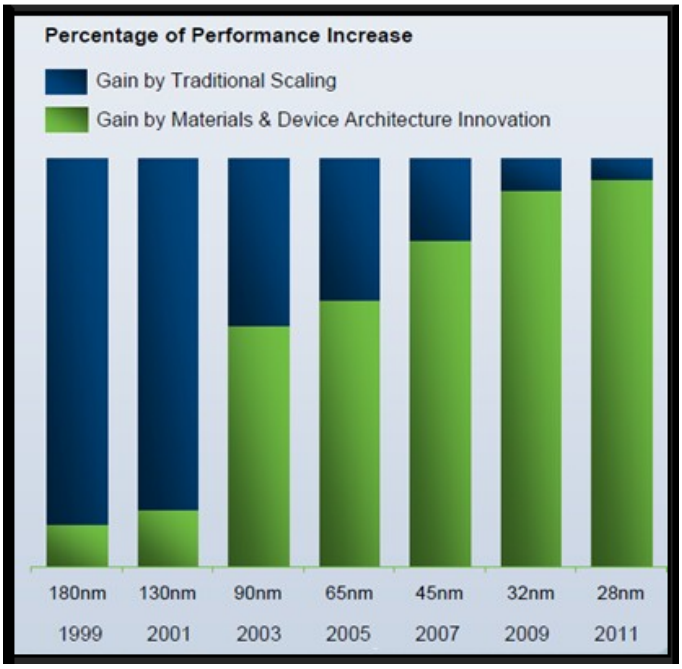


Figure 1.6 Performance growth rate of conventional technology scaling and architecture/materials innovation for each technology node. Source: IBM Microelectronics, Intel IC Insights, 2012.

tion of computational intensive tasks of the system. We propose the use of multiple coarse-grained accelerators, as they offer higher productivity per power consumption ratio and therefore, they form an attractive solution to the "Dark Silicon" problem [37, 38]. System Architecture Exploration refers to the investigation of the appropriate interconnection scheme of the under-development integrated system. This thesis studies heterogeneous architectural templates based on System-on-Chip (SoC) and Network-on-Chip (NoC) topologies. The last stage of the architectural synthesis refers to the optimization of the synthesis flow. This procedure provides the introduction of new design techniques, utilizing abstraction design methods of high-level synthesis (HLS), which further accelerate the development flow. At the same time this step allows programming flexibility in the programming of heterogeneous many-accelerator systems. Moreover, at this stage it is proposed a fast and automated way of design space exploration of architectural solutions that combine HLS and virtual prototyping techniques in order to find a set of optimal solutions (Pareto optimal) under a multi-objective analysis environment.

The last step (Physical Layer) of the holistic proposed methodological framework refers to the physical mapping of the developed modules, through previous procedures, in real implementation platforms. The proposed procedures allow the placement of designs either to reconfigurable or application-specific platforms, depending on the desired characteristics which are reflected in the ecosystem layer, but

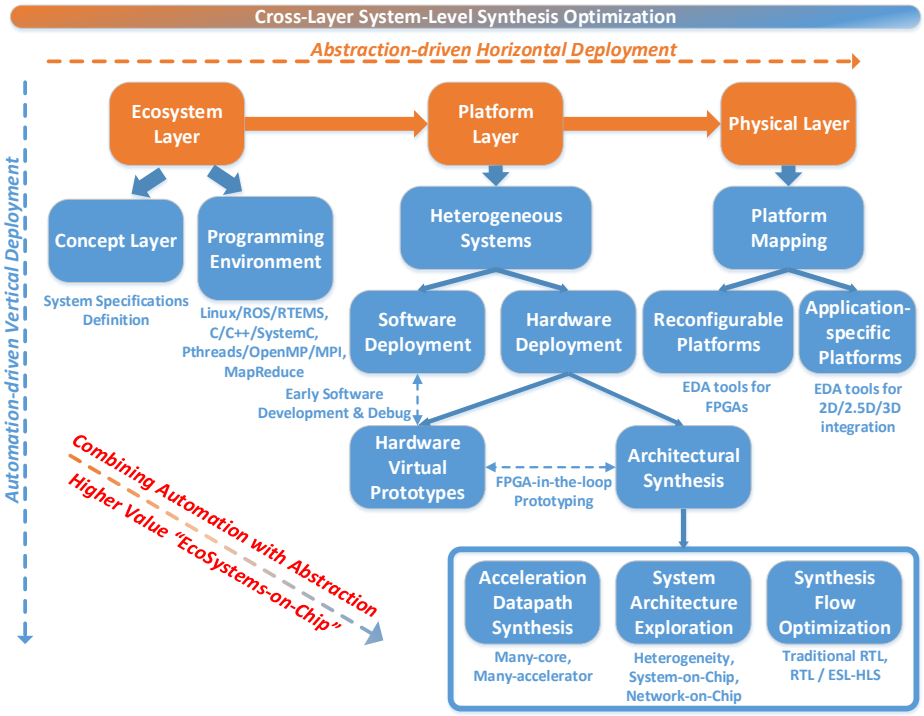


Figure 1.7 Dissertation overview.

also according to the constraints posed by the respective market. Specifically, the FPGA design flow is enriched through the proposed placement algorithms. These algorithms employ techniques of natural selection and evolutionary theory, providing optimal design solutions with reference to the maximum achieved operating frequency, wire-length, silicon area and power consumption. Such exploration is enabled through vertical and horizontal parallelization of the proposed algorithms and their multi-objective optimization mechanisms. At the same time, the mapping to application-specific platforms (ASICs) is enriched. By varying the micro-architecture, the proposed techniques achieve a reduction of emitted temperature and therefore they reduce the aging rate of the circuit, leading to increased reliability. Furthermore, we study the gains from the transition to the new promising three-dimensional integration technology (3-D), for which, to date, there is a lack of design tools, both academic and industrial. To this regard, it is suggested a novel design flow based on reliable conventional design tools (2-D) which manage to quantify the implementation gains from the transition to three-dimensional integration. The developed software tools are compatible with the file types of industrial standards, so they offer great flexibility in their adoption of alternative design flows and tools from different industrial vendors.

Overall, this thesis provides methodologies and tools at different design levels of integrated hardware/software systems bring-up. Through a structured design flow, it is attempted a coherent interaction of individual design flows, in order to establish a correlation of the characteristics sought by the market and those that can be offered by the technology. Basic characteristic of this effort is the clear conceptual separation of design levels, as well as the definition of specific and non-overlapping optimization mechanisms. The complete contribution to the proposed design flow targets design systems that can be deployed through a rapid, multi-objective optimized environment, incorporating largely systemic complexity. This degree of integration contributes to the emergence of a new generation of integrated systems which, in the context of this thesis, are assigned the term "Ecosystem-on-Chip" (ESoC).

1.2.1. Chapters Organization

The proposed framework and the corresponding design tools, which are developed in the research directions of this study, are organized according to the following structure:

- Chapter 1, i.e. the current chapter, presents the latest technology developments and trends of the dynamically evolving market of IT systems. Alongside, there are analyzed the limitations of existing design methods for systems that offer long term sustainability in the mentioned technological scaling and marketing trends. This chapter summarizes the key growth drivers of the thesis, through the presentation of a holistic methodological framework, which prioritises the individual contributions into discrete levels and describes the binding relationships among them.

- Chapter 2 presents the proposed methodology of rapid prototyping. It introduces the concept of virtual prototyping using the "FPGA-in-the-loop" technique, which speeds up the simulation process and enables the effective exploration of the solutions space during architecture optimization.

- Chapter 3 presents the cross-layer synthesis of heterogeneous architectures. Referring to the holistic methodology of the thesis (Figure 5.14), the chapter contains the Acceleration Datapath Synthesis, the System Architecture Exploration and the Synthesis Flow Optimization.

- Chapter 4, presents the proposed design tools for reconfigurable platforms. It introduces the characterization and classification technique, used during the phase of the synthesis, according to application's intrinsic characteristics. Also it is presented a multi-objective genetic algorithm for the placement problem, which utilizes the aforementioned characterization technique to guide the placement to Pareto optimal points depending on the classification of the application.

- Chapter 5, provides the architectural template and the programming environment for many-accelerator platforms. The proposed architecture promises

high accelerator density with minimal energy footprint. It utilizing FPGAs as a medium of programming coarse-grained accelerators. The incorporation of high-level-synthesis techniques allows the easy programming and the system-level deployment of such mass-parallelism architectures.

- Chapter 6, concludes the findings of this study and highlights the future extensions arising from the use of the proposed methodologies.

References

[1] Statista, *Statista statistics portal, global semiconductor sales from october 2011 to february 2015 (in billion u.s. dollars),* .

[2] Gartner, *Gartner inc. technology research,* .

[3] G. Moore, *Cramming more components onto integrated circuits,* Proceedings of the IEEE **86**, 82 (1998).

[4] R. Dennard, F. Gaensslen, H.-N. YU, V. Rideout, E. BASSOUS, and A. R. LEBLANC, *Design of ion-implanted mosfet's with very small physical dimensions,* Proceedings of the IEEE **87**, 668 (1999).

[5] CNET.com, *End of moore's law: It's not just about physics,* .

[6] Recode.net, *Moore's law hits 50, but it may not see 60,* .

[7] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, *Power challenges may end the multicore era,* Commun. ACM **56**, 93 (2013).

[8] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, *The eda challenges in the dark silicon era: Temperature, reliability, and variability perspectives,* in *Proceedings of the 51st Annual Design Automation Conference*, DAC '14 (ACM, New York, NY, USA, 2014) pp. 185:1–185:6.

[9] C. Mack, *Fifty years of moore's law,* Semiconductor Manufacturing, IEEE Transactions on **24**, 202 (2011).

[10] M. Taylor, *Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse,* in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE* (2012) pp. 1131–1136.

[11] A. Mallik, J. Ryckaert, A. Mercha, D. Verkest, K. Ronse, and A. Thean, *Maintaining moore's law: enabling cost-friendly dimensional scaling,* (2015) pp. 94221N–94221N–12.

[12] J. Hruska, *Nvidia deeply unhappy with tsmc, claims 20nm essentially worthless,* .

[13] M. D. I. Zvi Or-Bach, *Is the cost reduction associated with ic scaling over?* .

[14] J. Cong, M. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, *Architecture support for accelerator-rich cmps,* in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE* (2012) pp. 843–849.

[15] J. Cong and B. Xiao, *Optimization of interconnects between accelerators and shared memories in dark silicon,* in *Computer-Aided Design (ICCAD), 2013 IEEE/ACM International Conference on* (2013) pp. 630–637.

[16] N. Goulding-Hotta, J. Sampson, G. Venkatesh, S. Garcia, J. Auricchio, P. Huang, M. Arora, S. Nath, V. Bhatt, J. Babb, S. Swanson, and M. Taylor, *The greendroid mobile application processor: An architecture for silicon's dark future,* Micro, IEEE **31**, 86 (2011).

[17] Y. Turakhia, B. Raghunathan, S. Garg, and D. Marculescu, *Hades: Architectural synthesis for heterogeneous dark silicon chip multi-processors,* in *Design Automation Conference (DAC), 2013 50th ACM / EDAC / IEEE* (2013) pp. 1–7.

[18] G. Venkatesh, J. Sampson, N. Goulding-Hotta, S. K. Venkata, M. B. Taylor, and S. Swanson, *Qscores: Trading dark silicon for scalable energy efficiency with quasi-specific cores,* in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-44 (ACM, New York, NY, USA, 2011) pp. 163–174.

[19] M. J. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks, *The accelerator store: A shared memory framework for accelerator-based systems,* ACM Trans. Archit. Code Optim. **8**, 48:1 (2012).

[20] M. R. Choudhury and K. Mohanram, *Approximate logic circuits for low overhead, non-intrusive concurrent error detection,* in *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '08 (ACM, New York, NY, USA, 2008) pp. 903–908.

[21] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, *Impact: Imprecise adders for low-power approximate computing,* in *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design*, ISLPED '11 (IEEE Press, Piscataway, NJ, USA, 2011) pp. 409–414.

[22] J. Han and M. Orshansky, *Approximate computing: An emerging paradigm for energy-efficient design,* in *Test Symposium (ETS), 2013 18th IEEE European* (2013) pp. 1–6.

[23] P. Kulkarni, P. Gupta, and M. Ercegovac, *Trading accuracy for power with an underdesigned multiplier architecture,* in *VLSI Design (VLSI Design), 2011 24th International Conference on* (2011) pp. 346–351.

[24] D. Mohapatra, V. Chippa, A. Raghunathan, and K. Roy, *Design of voltage-scalable meta-functions for approximate computing,* in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011* (2011) pp. 1–6.

[25] A. Verma, P. Brisk, and P. Ienne, *Variable latency speculative addition: A new paradigm for arithmetic circuit design,* in *Design, Automation and Test in Europe, 2008. DATE '08* (2008) pp. 1250–1255.

[26] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, *Architecture support for disciplined approximate programming,* in *ACM SIGPLAN Notices*, Vol. 47 (ACM, 2012) pp. 301–312.

[27] J. Allred, S. Roy, and K. Chakraborty, *Designing for dark silicon: A methodological perspective on energy efficient systems,* in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '12 (ACM, New York, NY, USA, 2012) pp. 255–260.

[28] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, *Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits,* Proceedings of the IEEE **98**, 253 (2010).

[29] V. Hanumaiah, S. Vrudhula, and K. Chatha, *Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors,* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **30**, 1677 (2011).

[30] V. Hanumaiah and S. Vrudhula, *Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling,* Computers, IEEE Transactions on **63**, 349 (2014).

[31] U. Karpuzcu, A. Sinkar, N. S. Kim, and J. Torrellas, *Energysmart: Toward energy-efficient manycores for near-threshold computing,* in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on* (2013) pp. 542–553.

[32] D. Markovic, C. Wang, L. Alarcon, T.-T. Liu, and J. Rabaey, *Ultralow-power design in near-threshold region,* Proceedings of the IEEE **98**, 237 (2010).

[33] L. Wang and K. Skadron, *Implications of the power wall: Dim cores and reconfigurable logic,* Micro, IEEE **33**, 40 (2013).

[34] Microsoft, *Microsoft project, project management software,* (1992-2015).

[35] Trello, *Trello, web-based project management software,* (2011-2015).

[36] Synopsys, *Changing market drivers, synopsys insight newsletter, issue 1,* (2015).

[37] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, *Dark silicon and the end of multicore scaling,* in *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11 (ACM, New York, NY, USA, 2011) pp. 365–376.

[38] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, *Toward dark silicon in servers,* Micro, IEEE **31**, 6 (2011).

2

Rapid Prototyping Exploiting Hybrid-Virtual System-on-Chips

In embedded system domain there is a continuous demand towards providing higher flexibility for application development. This trend strives for virtual prototyping solutions capable of performing fast system simulation. Among others, such a solution supports concurrent hardware/software system design by enabling to start developing, testing and validating the embedded software substantially earlier than it has been possible in the past. Towards this direction, throughout this chapter we introduce a new framework, named Plug&Chip, targeting to support rapid prototyping of 2-D and 3-D digital systems. In contrast to relevant approaches, our solution provides higher flexibility by enabling incremental system design, whereas it can also handle platforms developed with the usage of 3-D integration technology.

2.1. Research Motivation for Virtual Prototyping

With vastly increased complexity and functionality especially in the nanometer era, where hundreds of millions of transistors on one chip are integrated, the design of complex Integrated Circuits (ICs) has become a challenging task. In addition to that, the continuously increased demand for even higher performance (i.e. in terms of operation frequency, power consumption, etc), imposes that new design techniques are absolutely required. Three-dimensional (3-D) integration, which contain multiple layers of active devices, have the potential to address these requirements, since it dramatically enhances chip performance and functionality, while it reduces the distance among devices on a chip [1].

Apart from the technology-oriented parameters that affect the efficiency and/or the flexibility of a digital system, the tight time-to-market requirements make conventional ways for product development (e.g. start software development after

finalizing hardware) to lead usually in missed market windows and revenue opportunities. Hence, there is an absolute requirement for software developers to get an early start on their work, long before the RTL (register-transfer level) of the hardware is finalized. This problem becomes far more important if we take into consideration that software aspects of ICs can account for 80%, or more, of embedded systems development cost [2], making the conventional way for product development insufficient. For instance, the International Technology Roadmap for Semiconductors (ITRS) predicts that software development costs will increase, and will reach rough parity with hardware costs, even with the advent of multi-core software development tools [2].

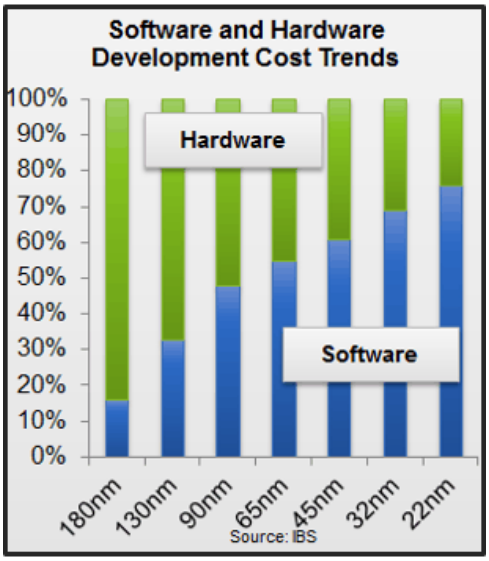


Figure 2.1 Cost development trends for hardware and software deployment, in relation to technology scaling.

In relation to the above observation, another crucial design parameter of integrated circuits is the scaling of the design costs with respect to the technological node integration. As shown in Figure 5.14, software development costs rise constantly in advanced technology nodes. Relative recent survey showed that most projects are delayed due mainly software development, while the largest percentage of its workforce design team deals with the software [3].

The supporting tools are also crucial for deriving an optimum solution. The existing Electronic Design Automation (EDA) flows are built on the fundamental premise that models are freely interchangeable among vendors and have interoperability among them. In other words, this imposes that models can be written, or obtained from other vendors, while it is known a priori that they will be accepted by any vendor tool for performing different steps of physical prototyping (e.g. architecture's analysis, simulation, synthesis, etc). Even though this concept seems straightforward and promising, it has been proven completely elusive in the world

of Electronic System Level (ESL). Specifically, the existing ESL solutions do not provide neither model interoperability, nor independence between model and software tools. Consequently, the adoption of ESL flows between different vendors could be though as a desired feature.

Towards this direction, and as research pushes for better programming models for multi-processor and multi-core embedded systems, Virtual Platforms (VP) solve one of today's biggest challenges in physical design: to enable sufficient software development, debug and validation before the hardware device becomes available. More specifically, with the virtualization feature, it is possible to model a hardware platform consisted of different processing cores, memories, peripherals, as well as interconnection schemes, in the form of a simulator [4]. Furthermore, as the task of hardware development progressively proceeds, it is feasible to redistribute to software teams updated versions of the VP that enable even better description of target architecture.

The concept of virtualization is also important for hardware architects, as it enables easier verification of IP (Intellectual Properties) kernels. This feature could be employed both in the case where only a few of the application's kernels have to be developed in hardware, as well as if incremental system prototyping is performed. In both cases, the virtualization feature provides all the necessary mechanisms for performing co-simulation and verification between the IPs developed in RTL (Register Transfer Level) and the rest application's functionalities executed onto the VP.

Throughout this research we introduce a software-supported methodology for performing rapid prototyping of complex 2-D and 3-D SoCs. For this purpose, we have automated the procedures dealing with system modeling and simulation between system's kernels mapped onto different platforms, namely the host PC, the VP and the target implementation medium (e.g. FPGA, ASIC, etc). Since multiple platforms are employed during this analysis, we have also developed a generic communication scheme for providing the desired data transfers under various parameters (e.g. packet size, maximum transmission unit (MTU), etc). Another challenging feature tackled throughout this research affects the potential gains of designing this SoC with the adoption of 3-D technology. Since the physical implementation tools for realizing 3-D designs [5] [6] cost a lot, the introduced framework can estimate the performance efficiency by incorporating such an advanced interconnection technology.

The rest of this chapter is organized as follows: Section 2 summarizes the state-of-the-art in the domain of virtual prototyping. Based on this analysis, we highlight the open issues, as well as the motivation of this research work. The proposed methodology for performing rapid prototyping is discussed in Section 3. Section 4 provides a number of experimental results that prove the effectiveness of introduced methodology. Finally, conclusions are summarized in Section 5.

2.2. Existing approaches for system modeling with virtual platforms

This section summarizes a number of representative approaches in the domain of platform virtualization. The scope of this analysis is to identify limitations of existing solutions, as well as their impact to sufficient rapid prototyping of complex digital systems. Additionally, the open issues, as they will be concluded from this analysis will be used as a motivation for the introduced Plug&Chip framework discussed throughout this research.

As the strict requirements for accuracy and development time cannot be easily met with the usage of conventional system prototypes, such as VP, the use of hybrid prototyping becomes the most viable solution. More specifically, the term *hybrid prototyping* refers to a scenario, where only a portion of the design is mapped to a physical device (i.e. FPGA, ASIC, CUDA, etc), another portion is simulated on a VP running on host, while the rest system runs natively onto the host PC. According to the literature, this enables to combine the accuracy and speed of the hardware platforms with the flexibility and visibility found in VPs [7]. Since this approach incorporates different methodologies which should be combined, it is imposed that the appropriate interfaces between consecutive techniques and tools have also to be developed. Recently, such frameworks are of increased importance, as the complexity of digital systems increases exponentially [2]. Furthermore, the flexibility provided by existing frameworks to start physical implementation from different abstraction levels than RTL (e.g. SystemC) imposes that software tools that automate the procedures of simulation, debugging and verification should be sufficiently addressed.

A typical interface for hybrid prototyping is the SCE-MI solution from Accelera [8], which provides a communication interface between the host PC and an HDL simulator. In more detail, the host PC at SCE-MI executes a testbench software for manipulating the interface. To the other side, the system's IP kernels developed in HDL include all the necessary mechanisms for accepting the stimuli through the interface and send back to the host the appropriate data.

Next subsections discuss in more detail a number of relevant approaches for supporting the communication between Host-to-VP (Host2VP), as well as VP-to-Hardware (VP2HW).

2.2.1. Communication between Host PC and Virtual Platform

This section provides a qualitative analysis regarding the first part, namely the communication between Host PC and VP (Host2VP) found in virtualization environments for supporting the hybrid prototyping approach. The results of this analysis are summarized in Table 2.1, whereas for sake of completeness, the last column of this table refers to the Plug&Chip introduced throughout this research.

Table 2.1 Qualitative comparison among Host-to-VP communication mechanisms.

	Altera Virtual Target [9]	QEMU/SystemC [10] [11]	HySim [12]	Semi-hosting [4]	Proposed
Abstraction Levels	Functional	Functional, TLM, RTL	Timed Functional	Functional	TLM, RTL
Accuracy in VP	Low	Cycle Accurate	Cycle Accurate [1]	Low	Cycle Accurate [1]
Data Synchronization	Built-in	Manual	Built-in	Manual	Built-in
SW execution to Host	Fast	Medium	Fast	Depends on Packet Size [2]	Fast
Simulation to VP	Fast	Medium	Medium	Depends on Packet Size [2]	Medium
Platform for Host software	Physical System	QEMU	Physical System	Physical System	Physical System
Supported CPUs for VP	ARM	Any	Any [1]	Any	Any
Number of CPUs in VP	Max 2	Unlimited	Unlimited	Unlimited	Unlimited
Communication mechanisms	Driver-based [3]	Built-in [4] or IPC	Via custom wrapper	Hard disk	IPC, Driver-based
Intrusive to host	No	No	Yes	Yes	No
Intrusive to VP software	No	No	Yes	Yes	No
TLM peripherals support	No	Yes	No	No	Yes
Multiple VP frameworks	No	Yes	Yes	No	Yes

The first approach depicted in Table 2.1 involves a PC-based simulator provided by Altera [9]. This simulator enables early software development by emulating the ARM Cortex-A9 CPU(s) and the peripherals which are included into the Altera SoC FPGA devices. In order to support hybrid prototyping, the simulator is connected with the host PC through a driver-based communication mechanism. Even though [9] supports fast functional simulation, the low timing accuracy (since it uses only functional simulation), introduces limitations to detailed software validation.

The second column corresponds to an approach where the QEMU x86 emulator is combined with a SystemC model [10] [11]. Although SystemC ensures the universality of this solution, as well as the unlimited support for CPU cores in the VP side, there are performance limitations related to the software execution on the x86 processor since it is actually emulated. Also, there are no standard synchronization mechanisms for ensuring the non-corrupted data transfers. Regarding the HySim (Hybrid Processor Simulation) approach [12] the communication with the host PC is established a custom wrapper, which acts as an abstract simulator for the out-of-interest software part. This tool enables to perform functional simulation of target

[1] Depending on the Instruction Set Simulator (ISS).
[2] High-speed for large data packets; low-speed for small data packets.
[3] In a driver-based communication scheme, the simulator is accessed through device driver.
[4] QEMU ISS is used as a SystemC module.

system, whereas regarding the accuracy of timing simulation, it depends on the selected Instruction Set Simulator (ISS) and the abstraction level. A drawback of this technique involves the additional effort required by designer in order to modify the software executed both on host and VP sides.

Next approach involves the usage of semi-hosting, which resides to the forwarding of commonly used system calls (e.g. *read*() or *write*()) to the host PC, in case where no O/S runs on the VP. Representative solution of semi-hosting is the OVP [4] software. Since the OVP supports only functional simulation, it is suitable mainly for performing system modeling at higher abstraction levels, where no timing accuracy is needed. Regarding the communication between Host and VP in OVP, it lacks of standard synchronization mechanism, as long as the *fcntl*() call (used for file locking) is not supported to every CPU model. More specifically, this communication can be realized only via a storage medium, e.g. the hard disk of host PC, which imposes that the simulation speed depends highly on the amount of transferred data. Even though this limitation can be tackled with Unix IPC mechanisms (e.g. named pipes), semi-hosting supports only POSIX regular files.

Table 2.2 Qualitative comparison among VP-to-Hardware communication mechanisms.

	Synopsys HAPS [13]	Xilinx Co-Simulation [14]	Altera Virtual Target [9]	Semi-hosting [5] [4]	Proposed
Abstraction levels	TLM	RTL	RTL	Functional	TLM
Accuracy in VP	Cycle Accurate	Cycle Accurate	Cycle Accurate	Week	Cycle Accurate [6]
Data synchronization	Built-in	Built-in	Built-in	Manual	Built-in
Simulation to VP	Fast	Slow	Fast	Fast	Fast
Supported hardware	FPGA	FPGA	FPGA	Any	Any
Supported CPUs for VP	ARM	Any	ARM	Any	Any
Number of CPUs in VP	Unlimited	Unlimited	Max 2 cores	Unlimited	Unlimited
Communication protocol	AMBA, Raw data	Raw data	Raw data	Raw data	Raw data
Communication mechanisms	Driver-based	Driver-based	Driver-based	Hard disk	Driver-based
Extensible library	No	Yes	No	Yes	Yes
TLM peripherals support	Proprietary Libraries	No	No	No	Yes
Multiple VP frameworks	No	No	No	No	Yes

[5] VP communicates with a host proxy which forwards data to/from the FPGA.
[6] Depending on the ISS.

The last column in this table summarizes the features of the proposed framework, in terms of providing the communication between host and VP. In particular, being a SystemC-based approach, it is a universal solution which supports different abstraction levels (e.g. TLM) and unlimited number of CPUs. Depending on the selected ISS, the Plug&Chip framework could provide even cycle-accurate simulation. Furthermore, it has built-in synchronization mechanisms, while the communication supports a non-intrusive switching at compile time between IPC (Inter-Process Communication) and driver-based connection.

2.2.2. Communication between Virtual Platform and Target Hardware

Although the connection to a hardware board can be easily established through a device driver, the communication mechanism should also enable the VP software to send (or receive) data to (or from) the hardware board. This is the core feature of such a VP to hardware (VP2HW) communication scheme. Table 2.2 summarizes a number of representative solutions that provide such a communication. Similar to previous case, the last column of this table corresponds to the proposed Plug&Chip methodology discussed throughout this research.

The first three approaches discussed in Table 2.2 are commercial products, which provide cycle-accurate hardware-assisted co-simulation with an FPGA device. In particular, the Synopsys HAPS [13] has a TLM interface to the VP software for manipulating the hardware platform. Apart from enabling raw data transfer through physical links (e.g. USB, Ethernet, etc), HAPS also supports a transactor protocol (AMBA), if this communication scheme is available in the final system. Even though HAPS seems to be a promising solution, it is based on proprietary TLM libraries, which make it hardly extensible.

Another approach affects the Xilinx Co-Simulation Flow [14], which establishes a connection between the RTL simulator (i.e. Xilinx ISim [15]) and the target FPGA through a physical link (e.g. USB, Ethernet, etc). Similar to HAPS, the Xilinx flow does not support higher abstraction levels in designing new kernels. Therefore, the IP kernels, as well as the whole platform, have to be developed in RTL. This also imposes that the simulation speed is expected to be lower compared to similar approaches that are based on SystemC models. A relevant solution involves the Altera Virtual Target [9], which similar to Xilinx flow, enables to deploy an FPGA for mapping custom IP kernel(s) described in HDL in systems that contain up to 2 ARM processors.

Next column discusses the advantages and disadvantages of the semi-hosting approach. In this case, the host becomes a proxy between VP and hardware board, as this is the only viable solution to overcome the limitation of VP software to support access to character devices. Even though semi-hosting can be considered extensible, as long as the designer models the target communication protocol, this is rarely efficient due to the lack of support for lower abstraction levels. Moreover, there is no straightforward way to support TLM at semi-hosting.

Regarding the introduced VP2HW communication scheme found in Plug&Chip, it can handle an increased number of processing cores, while the target hardware

is a generic device (e.g. FPGA, ASIC, GPU, CPU, etc), if the appropriate driver is provided. Furthermore, we have to mention that although the test cases discussed throughout this research involve only raw data transfer with the hardware platform, the introduced VP2HW communication library is fully extensible; hence the designer can easily port any other communication protocol.

2.3. Prototyping Methodology

Even though there are plenty of design tools that tackle software (SW) and hardware (HW) problems individually, there are only a few approaches that leverage problems arising in systems that tightly integrate SW and custom HW. This mainly occurs due to the challenges related to system integration that have to be addressed. Even limited, there are EDA approaches which promise to alleviate the integration problem in RTL simulation, emulation and prototyping environments. However, these solutions are often too complex, slow and expensive. Usually, it is the communication link between the host computer and prototyping HW that is mostly constrained.

Towards this direction, throughout this section we propose a framework for enabling product development jointly by SW and HW teams in a way that close interaction is allowed during the development phases. Fig. 2.2 shows the overall flow of the proposed Plug&Chip methodology consisted of three consecutive design stages: (i) system modeling, (ii) rapid virtual prototyping and (iii) system integration. The competitive advantage of this framework is the provided PC-based co-simulation, which trade-offs between speed (functional simulation) and accuracy (cycle-accurate simulation), depending on designer requirements.

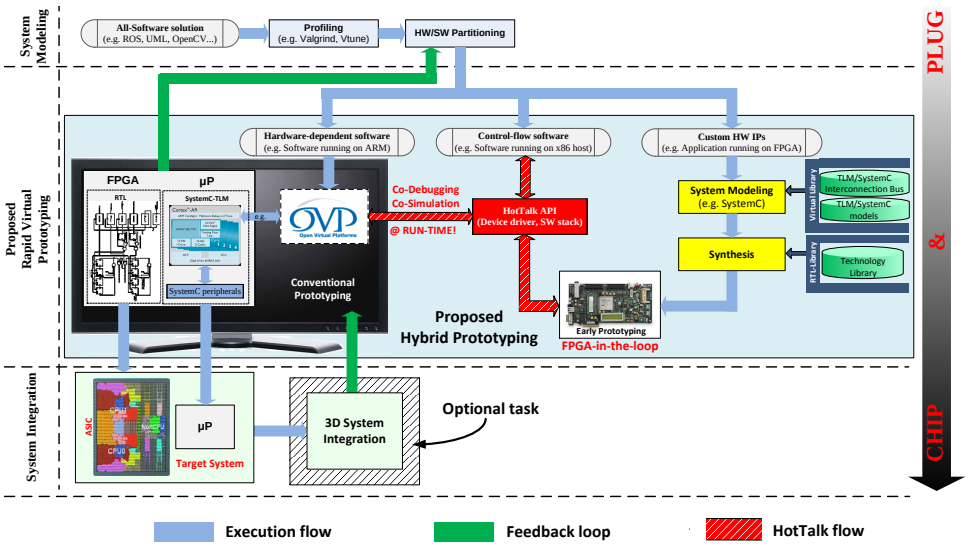


Figure 2.2 The proposed methodology for the Plug&Chip framework.

System modeling describes the stage where the development team provides an abstract description of the system's architecture and starts planning the way that this functionality has to be modeled into HW and SW kernels. Also, during this stage, all the top-level performance constraints are met. Since this is an early design step, the functionality of the final system is described with an "all-software" solution.

Starting from this all-software solution, initially we profile the application's algorithms to determine those kernels that highly affect the system's performance. Different criteria might be incorporated for this task, while the most common ones affect the determination of computationally intensive tasks, the tasks with increased demand for communication (I/O), as well as those tasks that can be executed much faster if we extract their inherent parallelism. For this purpose, a number of software tools can be employed (e.g. Valgrid, VTune, etc). Based on the conclusions derived from profiling task, it is possible to perform a HW/SW partitioning to the pure software implementation, depending on the criteria discussed previously. The output of partitioning step classifies the "all-software" solution to three categories: (i) the HW-dependent software, (ii) the control-flow SW and (iii) the custom HW IPs.

Custom HW IPs are peripherals and hardware accelerators, which provide platform connectivity to off-chip world and acceleration to software functions, respectively. The HW-dependent software refers to the algorithms executed onto the embedded target CPU. While our methodology relies on a PC-based development infrastructure, there is a software stack running on a native host (e.g. x86 compatible), which is responsible for establishing the communication layer between HW-dependent software and custom HW IPs, as well as to provide all the necessary synchronization for the computation tasks. This software stack is referred in Figure 2.2 as "control-flow software". Apart from the development stage discussed previously, the host PC could also be part of the final system. In such a case, the control-flow software includes also the SW that will be executed onto the host PC.

The introduced framework also addresses limitations posed during the profiling step. More specifically, in case where the profiling procedure is performed on a different platform from the actual target system, this might lead to inaccuracies in the derived conclusions. For instance a different platform imposes changes in the Instruction Set Architecture (ISA), the microarchitecture, the compiler, resulting in variances in the executable that is profiled. In order to strengthen the partitioning decisions, HW-dependent information has to be provided, while based on existing solutions, the most accurate profiling information is available after the first design prototype is developed. Typically, once a project has reached this stage, most of the budget has been sunk, which is usually beyond the point of "no return". In contrast, the proposed framework offers the flexibility of incremental HW development and system testing under real world constraints, so that accurate profiling information can also be provided through development stage with a feedback loop.

The second stage of the proposed framework deals with the Hybrid Virtual Prototyping, which is actually the core stage of the proposed methodology. In order to support the interaction between SW and HW development teams, we adopt the

usage of TLM-SystemC models. Different ISSs can be employed for this purpose, however for the rest of this research, the Plug&Chip is based (without affecting the generality of introduced methodology) on OVP [4], since it is a publicly available and easily extensible approach. Additionally, the increased simulation speed provided by OVPSim ensures that complex systems can be modeled in reasonable amount of time (hundreds of millions of simulated instructions per second). As the OVP models are pre-built, they support fully functional simulation of a complete embedded system. Also, since these models are binary-compatible with the simulated HW, the developed software can be executed onto the target (final) system without any modifications. This enables faster iteration for the software development teams.

Similarly, HW developers are also benefited from the adoption of hybrid VP discussed throughout this section. Since this platform is composed of OVP and TLM/SystemC models, it exhibits increased flexibility which in turn alleviates many constraints that designers face during the architecture design. More specifically, the former models (related to OVP) describe the software part of the target system (e.g. executed onto an embedded processor), while the TLM/SystemC models provide the design functionality that has been mapped to custom HW IPs, after system partitioning.

After having a high-level system modeling that meets the design's specifications, we proceed to the HDL development. As long as new IPs are developed, the HW design team is able to incrementally test these IPs by replacing a functionality of the employed SystemC/TLM model with the equivalent HDL prototype mapped onto FPGA boards. The connection between VP and FPGA is established with HotTalk API. More specifically, this API provides the connectivity between the VP prototype and the target hardware, as well as between Host software and VP, through a physical interface found on the host PC (i.e. Ethernet, PCI, USB, etc). Next subsections describe in more detail the functionality of this API.

Offering the HotTalk API, the proposed framework provides a wide class of middleware stack, composed of device drivers on host PC, libraries in OVP and transactors in FPGA, so that designers can efficiently test the entire system from early design iterations down to the final system validation with real-world testbenches, with the minimum possible effort. As mentioned previously, such an incremental design flow provides all the necessary information about meeting the system's specifications, which in turn can be used for performing additional optimizations of the whole system, or re-partitioning the software (through the feedback loop).

The last stage of the proposed methodology deals with the system integration. During this stage, the different cores of target system, including among others the embedded CPU, the reconfigurable fabric, as well as the memory components are integrated to form the target SoC, which can be further optimized if we incorporate a 3-D process technology. Towards this goal, our methodology employs an approach for quantifying with acceptable fidelity the potential gains of using such a technology. In contrast to relevant solutions which are based mostly on academic tools, the evaluation of 3-D stacks discussed throughout this research is performed with the usage of Cadence tools.

The introduced methodology depicted in Fig. 2.2 is automated by a number of

CAD tools. Additional details about the proposed Host-to-VP and VP-to-HW communication libraries, as well as the evaluation of 3-D embedded systems can be found in upcoming subsections.

2.3.1. HotTalk API: Host2VP and VP2HW Communication Infrastructure

This subsection describes in more detail the proposed communication scheme for realizing the communication between the host PC and the VP, as well as between VP and a hardware board (the FPGA for our case study). These two libraries, named Host2VP and VP2HW respectively, form the core part of HotTalk API. In addition, HotTalk API provides an FPGA transactor, i.e. a hardware module mapped on the reconfigurable platform, which realizes the physical link with the FPGA device.

2.3.1.1 Host2VP

Fig. 2.3 gives a functional overview of the tasks implemented in the Host2VP library. This library provides a universal coding style for the host side in order to avoid modifications when the VP is replaced with a real board. Towards this goal, the library provides a high-level interface for realizing four main tasks: (i) open device, (ii) close device, (iii) send data to the device and (iv) receive data from the device. These calls are implemented as generic wrappers which can be adapted to any communication mechanism between the host and the embedded system (either virtual or physical) without imposing any modification to the host's software. In particular, in case of using a real hardware board (case 1 of Fig. 2.3), which is recognized as a character device by the O/S, the I/O calls manipulate the respective O/S system calls ($open()$, $close()$, $write()$ and $read()$ respectively), while the overall communication is performed through the device driver.

On the other hand, the second case of Fig. 2.3 assumes that VP is used instead of a real hardware. In such a case, those I/O calls manipulate the Inter-Process Communication (IPC) mechanisms for realizing the data transfer between host and VP. For this purpose, two FIFO (First-In/First-Out) queues are employed, as it is depicted in Fig. 2.3, each of which is implemented as a shared memory segment. The size of these queues is defined at compile time depending on the connectivity requirements between host PC and VP posed by the target architecture. Specifically, the usage of data packets with increased size leads to reduced IPC overhead in case of massive data transfers, whereas the smaller packet size is preferable whenever there is a limited amount of data to be transferred.

The efficient synchronization for data transfers between the host and the VP is also crucial for the Host2VP library. More precisely, both for the host and the VP, the data synchronization can be realized with the usage of O/S semaphores. Since each semaphore operation is an atomic action, our framework guarantees that no racing conditions will occur. This imposes that if a host has to send and receive data simultaneously, then only one operation will be committed. The second operation will take place only when the currently committed operation is accomplished.

Another feature of introduced library is its applicability to any other VP frame-

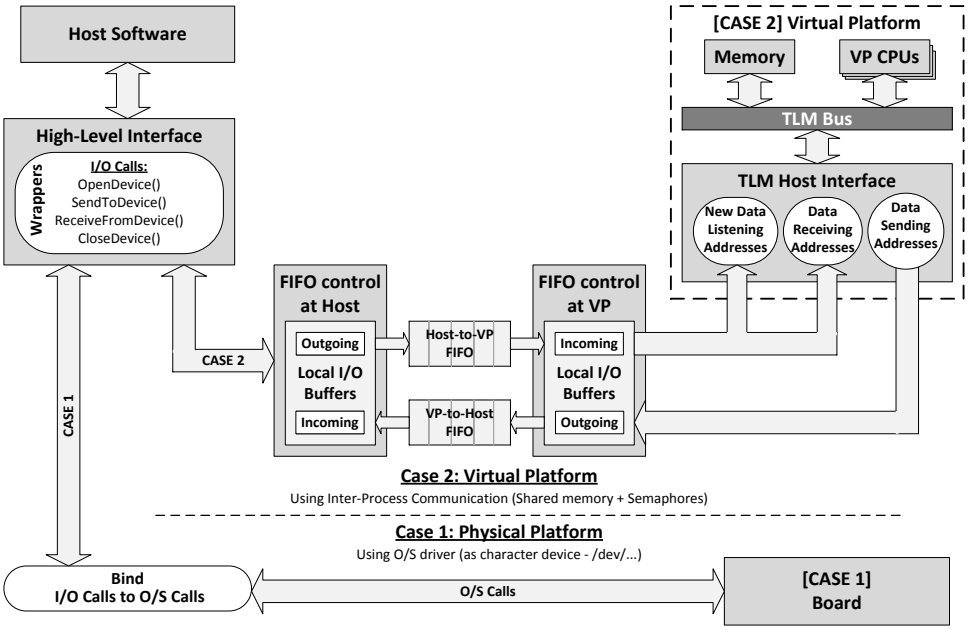


Figure 2.3 Communication mechanism between Host and VP.

work and/or ISS, leading to a universal communication library. For this purpose, the Host2VP library provides an easy-to-use C API, which includes standardized I/O calls for the data manipulation to the VP side. In order the VP side to utilize these calls, the library provides a TLM2.0 VP peripheral, referred as TLM Host Interface, which binds each of these I/O calls with a bus address. Three types of addresses are provided: (i) *New Data Listening Addresses* for checking if new incoming data exist, (ii) *Data Receiving Addresses* for reading the incoming data from host and (iii) *Data Sending Addresses* for writing the outgoing data to host. Hence, the VP software can use each call by accessing the corresponding bus address.

Finally, in order to provide a fully adaptive interface at TLM level, Host2VP is implemented as an hierarchical library, based on a template class, which includes the core methods that realize the TLM transactions. These methods manipulate a set of I/O calls for checking whether new data is available, receiving the incoming data and sending the outgoing ones. With this template class, the designer can develop new TLM Host interfaces with the minimum possible effort.

2.3.1.2 VP2HW

The functionality of VP2HW library, which realizes the communication between the VP and a real hardware board, is depicted in Fig. 2.4. Similarly to the previous case, VP2HW has to support a wide variety of hardware boards. Towards this goal, the target hardware board is recognized as a character device, whereas the communication is performed through the associated device driver. Such a selection

provides the appropriate synchronization for guaranteeing uncorrupted data transfer. Although for the scope of this research the employed device driver establishes the communication between the host and the FPGA through Ethernet, any other protocol could also be used. Furthemore, this library supports raw data transfer between VP and target hardware, through the physical layer (PHY) of Ethernet (there is no requirement for protocol, or service). Additional details about the employed communication scheme can be found in Section 3.1.3.

In addition to this, the VP2HW library supports the communication between the VP software and a real hardware board through a TLM interface connected with the rest of VP as a TLM2.0 peripheral. In particular, the TLM interface exposes to the VP software a number of registers, which trigger the data transfer to and from the hardware board. As Fig. 2.4 depicts, the VP software uses these registers in order to receive (send) the incoming (outgoing) data respectively, after setting the length of the transferred data. Also, a status register indicates if the data transfer between VP and hardware was successful. Whenever the VP software uses the registers for incoming or outgoing data, $getIncoming()$ or $setOutgoing()$ methods are invoked respectively.

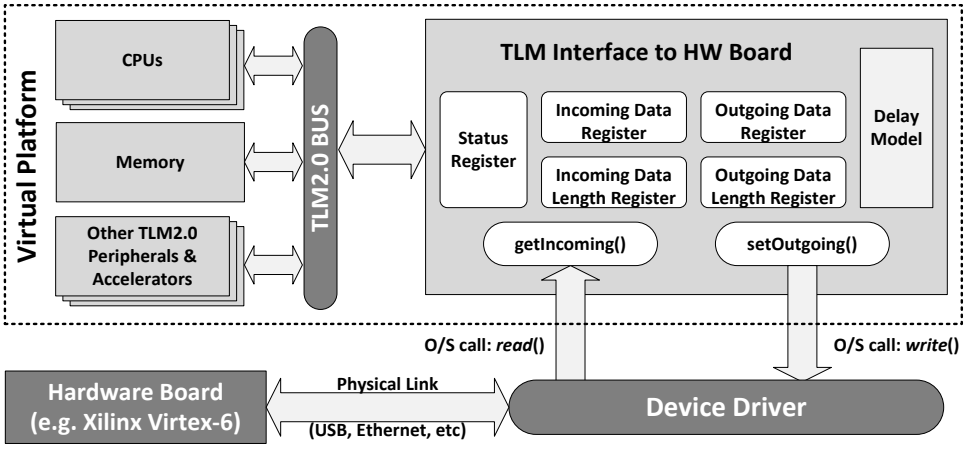


Figure 2.4 Communication mechanism between VP and hardware board.

Another feature provided by the developed VP2HW library affects the delay of hardware board, which does not affect the timing of the TLM transactions. In other words, the designer can define the desired timing resolution. For this purpose, the TLM interface of introduced framework provides a delay model which corresponds to the desired timing accuracy of TLM transactions. Such an approach is important in case the employed hardware device is not the implementation medium of final system. Finally, the developed library is protocol-independent, and consequently any potential/customized communication protocol between VP and hardware can be selected.

2.3.1.3 Implementation of the HotTalk FPGA Transactor

In order to realize the VP2HW communication, the HotTalk API provides a communication protocol consisted of a transactor module mapped onto the FPGA. This transactor handles the incoming or outgoing data, thus establishing the connection between the hardware IP and the communication port (in our case Ethernet). As long as custom HW IPs have been developed, it is possible to verify their functionality under real-world scenarios, using the HotTalk FPGA transactor. This approach was proven that detects bugs and performance issues that cannot be foreseen when solely software testbenches are employed.

The overall communication flow established by the transactor relies on a producer-consumer scheme and it is depicted in Fig. 2.5. More specifically, the testbenches to the host side (either in native code or through a VP) generate the necessary input traces for HW IPs. Then, the device driver assembles input data to protocol frames. The protocol in it's current version is built on top of raw Ethernet frames. In order to maximize the communication throughput, the size of the header packets is limited to 8 bytes, referring to the destination address (6 bytes) and the length of the frame (2 bytes). The rest packet (MTU-8 bytes) is used for data transfer. On the FPGA side, the HotTalk FPGA transactor is responsible for serving the RX requests from host's network interface controller (NIC) (initiated from device driver) to the custom HW IPs. A similar approach is followed for the reverse data direction, in the case that FPGA is forwarding results to the testbench.

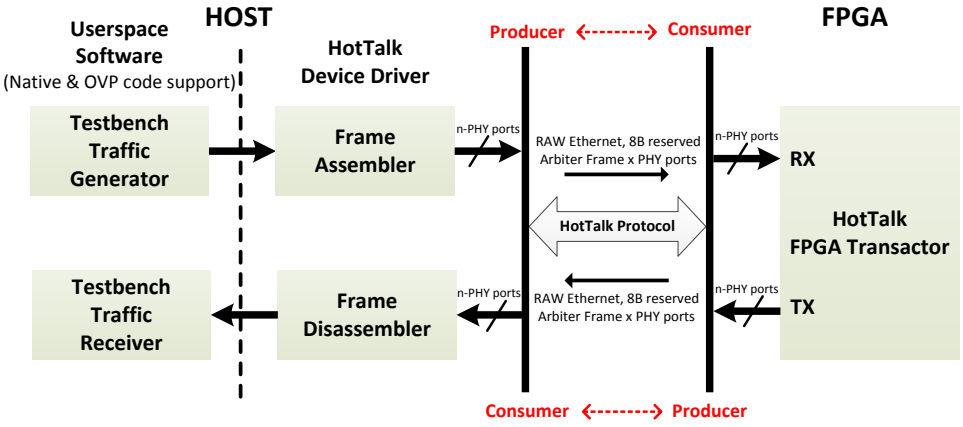


Figure 2.5 The communication flow established in the transactor.

As we have already mentioned, the communication between the developed IP kernels and the VP plays an important role in the overall system's performance. To facilitate the data transfer, we designed a communication scheme depicted in Fig. 2.6, which combines off-the-self and custom-made kernels to implement raw Ethernet connection. This scheme is supported by (i) a communication back-end IP, named ComCore, (ii) an arbiter and (iii) the Design Under Test (DUT).

For this purpose, we utilized the integrated Ethernet physical layer (PHY) chip

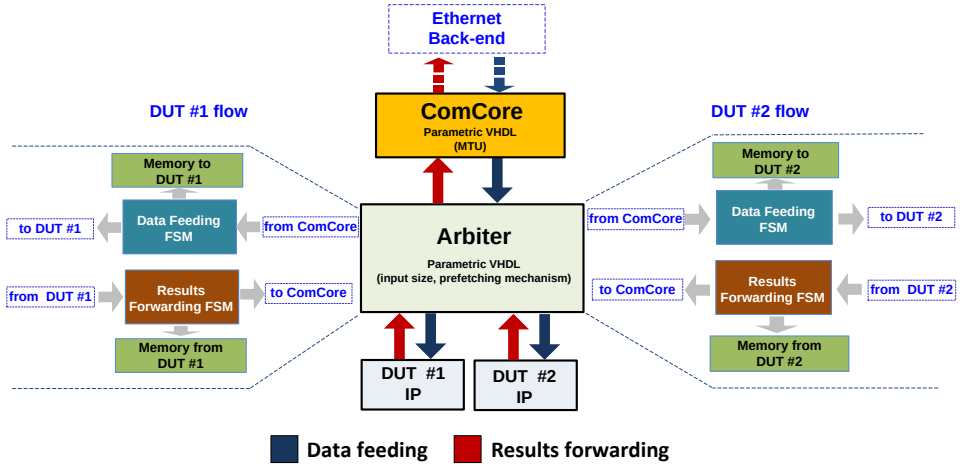


Figure 2.6 Architecture of the employed FPGA transactor.

found in Virtex-6 FPGA board (HTG-V6-PCIE-L240T-2) to perform character encoding, transmission, reception and decoding. Building on top of PHY, we developed a custom Rx/Tx controller based on the Ethernet MAC IP core from OpenCores [16], which implements the CSMA/CD LAN in accordance with the IEEE 802.3 standards. The resulting back-end component, as shown in Fig. 2.6, supports the custom-made kernels used to construct and distribute heterogeneous packets to the various hardware modules of the target system. Specifically, we use "ComCore" for the split/synthesis of large data packets to frames of 1,500 bytes. Moreover, ComCore handles the MAC controller's signaling to provide a simple Wishbone interface to the remaining FPGA modules.

The *Arbiter* component controls the communication channel to avoid conflicts between the processing modules. Among others, this arbiter provides prioritized communication for these modules based on a round-robin polling. In order to support a pipeline operation, the Arbiter pre-fetches packets while the remaining modules process already fetched data. Such a functionality exploits the DMA capabilities of state-of-the-art off-the-self NICs: Rx and Tx operations can be performed in parallel by dedicated Ethernet PHY chips, while at user space, distinct threads can execute independent tasks of the algorithm being scheduled dynamically by the operating system. Additionally, we have to notice that the proposed scheme can support multiple channels of Ethernet communication between the VP and the FPGA, more precisely one channel per DUT, depending on the distinct Ethernet ports found on the target FPGA board. For instance, in Fig. 2.6 an arbiter with two DUTs is depicted. In such a case, the arbiter uses headers to designate the receiving/transmitting module from/to the VP. Regarding the arbiter's architecture, it incorporates two distinct finite state machines (FSMs), each of which is dedicated to transmission and reception procedure respectively. Such a design approach allows independently data feeding and results forwarding, as long as the FIFO memory

dedicated to each FSM is big enough to store the data. Thus, the arbiter could support a parametric pre-fetching mechanism (with configurable amount of pre-fetched data frames), so that the the DUT IP can work on high utilization ratio without waiting the VP2HW communication at every iteration step.

As compared to existing implementation [16], the ComCore developed for the scopes of Plug&Play features batch mode transmission-reception of parametric-length frames, error and collision detection. In conjunction with a Linux kernel driver developed to cooperate with the introduced framework, it also offers recovery from system collisions. On the software side, we developed a device driver to support the communication between the Ethernet Network Interface Card (MII standard compatible NIC) and the VP. The driver provides the interrupt handling of the NIC to enable the asynchronous communication within the system via a transparent interface to the user space. In more detail, the deriver performs read and write operations to a Linux character device file, whereas it was developed as a loadable kernel module for supporting raw Ethernet frames of 1,500 bytes (MTU).

2.3.2. Evaluation of 3-D Embedded Systems

Even though the advantages of 3-D chip stacking seem obvious from a theoretical point of view [2] [1], designers need CAD tools in order to embrace this technology. Even before physical design tools for 3-D become commercially available, path-finding tools will be needed. Such tools will enable designers to explore a number of different 3-D implementation alternatives for their design and come up with the few process technology and design options that are Pareto-optimal. This process typically happens early in the design cycle of a new product, when designers fix high-level decisions like selection of process technology, high-level chip architecture (amount of task-level parallelism), etc.

This subsection describes the part of the proposed Plug&Chip framework dealing with the estimation of performance metrics for SoC designs, when they are physically implemented with adopting the 3-D integration technology. During this step all the platform-dependent decisions, such as the way the system's architecture is partitioned, the IP block-to-layer assignment, the selection of interlayer interconnection technology, etc, are made. By distinguishing this step from the pure physical prototyping step, we can support 3-D stacks comprising heterogeneous layers.

The introduced methodology for supporting fast evaluation of 3-D SoCs consists of three modular steps in order to enable interaction with tools from similar and/or complementary flows. More specifically, the steps of this methodology are summarized as follows:

- *Pre-processing*: Verification of functional integrity for the design and extraction of its IP-XACT description [17]. The IP-XACT is an XML format that defines and describes electronic components and designs.

- *3-D Stack Generation*: Generates the 3-D stack and determines the communication (routing paths) among layers.

- *3-D System Prototyping*: Performs the physical implementation of 3-D SoC and evaluates the derived solution.

2.3.2.1 Pre-processing

The first step in our methodology is depicted schematically in Fig. 2.7. Initially, the architecture's RTL description is simulated under various parameters and constraints (e.g. clock period, on-chip memories organization) in order to verify the system's functionality.

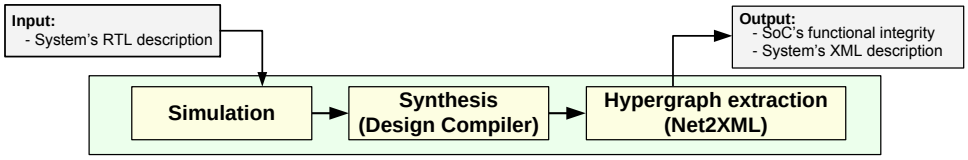


Figure 2.7 Tasks for the pre-processing step.

Then, the desired hierarchy for the target 3-D architecture is determined. Different levels of hierarchy are possible to be handled by our framework, each of which exhibits advantages and disadvantages. For instance, a block-based system's description leads to a coarse-grain solution, whereas a gate-level netlist comes with a finer system implementation. In other words, the fine-grain approach imposes the highest performance enhancement, but it also introduces the maximum computational complexity for performing architecture-level exploration. We have to mention that due to the importance of this selection, careful study about the hierarchy should be applied, because a sub-optimal solution might alleviate the performance enhancements imposed by the 3-D integration. For the scope of Plug&Chip framework, we choose (without affecting the generality of the proposed framework) to maintain the system's hierarchy among heterogeneous modules (e.g. logic, memory), while each module is flattened in order to maximize the performance enhancement.

After defining the SoC's hierarchy, the RTL description is synthesized with *Synopsys Design Compiler*. As long as the design constraints (e.g. timing slacks, DRC's, etc) are met, the output from synthesis is translated to an equivalent XML description. This task is software-supported by our new publicly available tool, named *Net2XML*.

2.3.2.2 3-D Stack Generation

The derived XML description which represents the SoC's netlist after technology synthesis is fed as input to the second step of our methodology, depicted in Fig. 2.8, where we deal with the 3-D stack generation under the selected design constraints.

Initially, the application is partitioned into a number of subsets. Different optimization goals can be considered during this task, such as the minimization of connections between partitions, while respecting some constraints (e.g. like keeping DRAM and logic on different partitions). Previous studies showed that partitioning

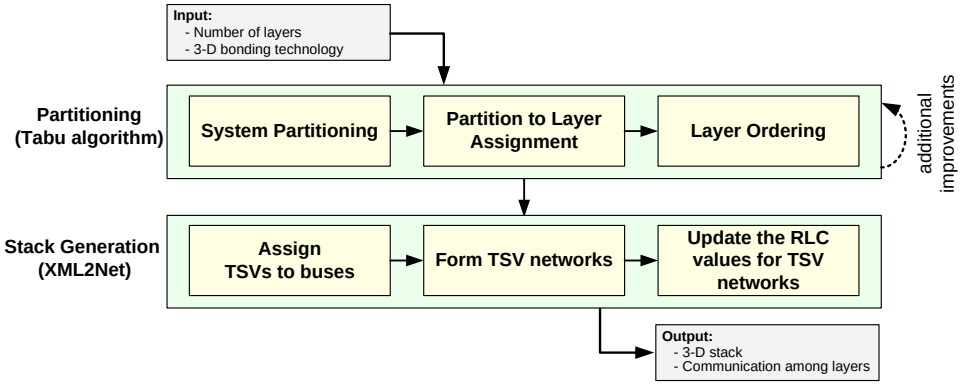


Figure 2.8 Tasks for 3-D stack generation.

algorithm exhibits increased flexibility whenever the number of subsets is higher as compared to the corresponding number of device layers [18].

Next, the assignment of derived subsets to the layers of the target 3-D architecture takes place. During this step, both fabrication (i.e. yield) and cost parameters (i.e. number of TSVs) are taken into account. More specifically, for a given layer, only technology-compatible components can be assigned to, while each of the layers has to exhibit sufficient area utilization. Finally, we build a prototype of the stack by deciding on the order of the die in the stack (which layer goes to the bottom, which one on top, etc) and the choice of 3-D bonding technology. Even though alternative bonding technologies could be evaluated with the proposed framework, such as TSV (or Face-to-Back), Face-to-Face, as well as wire-bond, however, throughout this research we provide results only for the TSV approach, which leads to the maximum performance enhancement [2] [1].

The tasks of system's partitioning, partitioning to layer assignment and layer ordering are software-supported by our publicly available *TABU* algorithm. An initial version of this tool was presented in [19], but for the scope of this research, the algorithm was massively extended. More specifically, initially the tool was developed for supporting exclusively designs mapped onto 3-D FPGAs, whereas the current version is also aware about handling designs for 3-D ASIC platforms.

The output from partitioning procedure provides information about the architecture's functionality assigned to each layer, as well as the required connectivity among layers. This information is appropriately handled by our tool, named *XML2Net*, in order to assign a TSV array to each bus that connects the architecture's components to different layers. Note that whenever a bus needs to be routed in layers i and j, silicon area equals to the area occupied by the TSV array has to be reserved in both layers. Even though our framework can also handle distinct TSVs, throughout this study we select to employ arrays of TSVs because they introduce fewer constraints to routing algorithm. Finally, the TSV arrays that provide bus connectivity between adjacent layers i and j are connected through special-purpose routing paths, named *TSV networks*. As we will discuss later, these networks are

actually implemented with additional metal layers, which exhibit tunable RLC characteristics in order to correspond to those found in TSVs from the selected 3-D technology.

2.3.2.3 3-D System Prototyping

The last step in our framework, depicted in Fig. 2.9, deals with system prototyping to derive the 3-D stack. More specifically, during this step we perform floorplanning, power and ground network generation, placement of physical library cells, clock tree synthesis and global/detail signal routing with the *Cadence SoC Encounter* tool. Since our framework is software-supported by a 2-D commercial flow, it is not possible to modify the functionality of these tools (their source code is not available). Hence, we have to make them aware of the additional flexibility imposed by the third dimension through appropriate design encoding. For this purpose we introduce:

- *Virtual layers*: Our framework assumes that the target 3-D SoC consists of *virtual layers*, each of which contains hardware resources that have to be assigned to different physical layers of the 3-D stack.
- *TSV networks*: These networks correspond to routing paths that provide connectivity between TSV arrays assigned to adjacent layers. Note that during physical implementation, our framework preserves that TSV arrays assigned to consecutive layers are spatially aligned. This is possible by forcing the placement of TSVs to the same relative (x, y) co-ordinates between adjacent layers. The *TSV networks* are actually implemented through additional metal layers inserted to the technology library file, while their total resistance (R), capacitance (C) and inductance (L) values correspond to the TSV's RLC parameters [20].

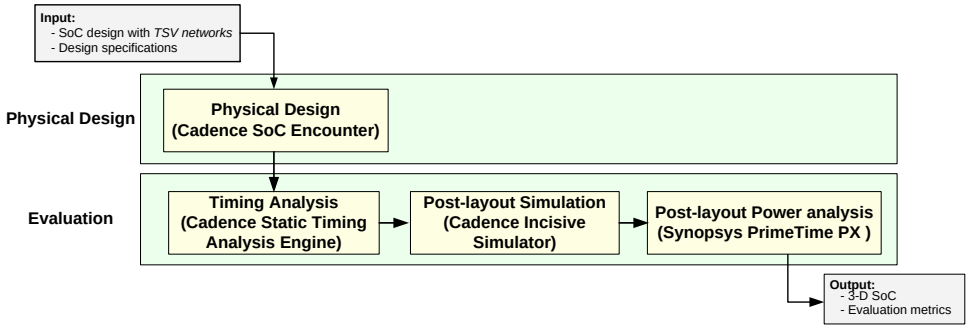


Figure 2.9 Tasks for 3-D system prototyping.

In order to clarify the concepts of virtual layers and TSV networks, Figure 2.10 gives an example, where a design is modeled as a 3-D chip consisted of four layers. More specifically, Fig. 2.10(a) depicts how the design is partitioned to four virtual

layers, as well as the connectivity among these functionalities with five buses (implemented as TSV networks). Similarly, Fig. 2.10(b) corresponds to the same design after performing detailed routing (e.g. with the usage of Cadence nanorouter). As we have already mentioned, the electrical characteristics of these networks are appropriately annotated in order to correspond to those of TSVs, based on the selected fabrication technology. Hence, even though the routing paths of different TSV networks in Fig. 2.10(b) exhibit different wire-length, all of them have exactly the same *RLC* parameters.

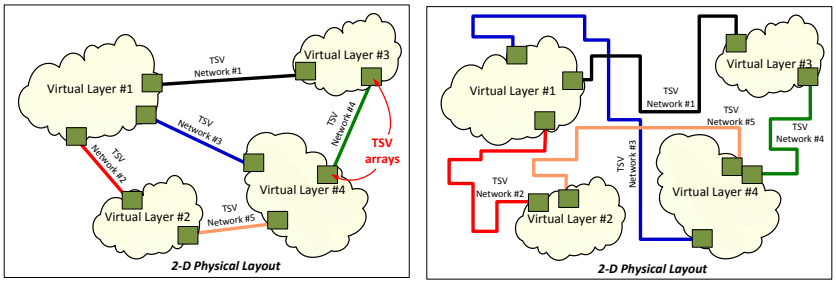


Figure 2.10 Paradigm of evaluating a 3-D design with four layers: (a) model the design with virtual layers and TSV networks and (b) design after successfully routing.

After 3-D physical prototyping, we evaluate the efficiency of the derived solution by applying timing analysis. For this purpose we employ the *Cadence Static Timing Analysis Engine*, while for sake of completeness the analysis is performed both in advance, as well as after clock tree synthesis and architecture's routing. In case the derived 3-D stack does not meet system's constraints/specifications, a number of design optimization may be applied for additional improvements.

2.4. Experimental Results

This section provides a number of experimental results that prove the efficiency of the proposed Plug&Chip framework. For demonstration purposes, we provide this analysis in two complementary steps, similar to those presented in our methodology. More specifically, initially we quantify the efficiency of the derived communication link between host PC and target platform, while then we also provide results about applying the introduced framework for designing a 3-D instantiation of a well-established embedded processor (OpenRISC [21] processor). Furthermore, in order to depict that our introduced framework is also applicable to complex systems, subsection 4.3 provides experimental results for applying Plug&Chip methodology to a project founded by European Space Agency (ESA) targeting to develop in hardware computer vision algorithms for supporting autonomous rover navigation.

2.4.1. Evaluation of Communication Infrastructure

This subsection focuses on quantifying the efficiency of the proposed communication scheme between host PC and VP, as well as between VP and the hardware board. Since the main objective during the development of our framework was to achieve as lower communication overhead as possible, while retaining at the same time the support among others the multiple VP frameworks and the extensibility feature, such an evaluation becomes an important issue for deriving a widely accepted solution in the domain of system virtualization. Additionally, as we are primarily interested on quantifying the communication scheme, ignoring about potential limitations posed by the computational part of the target system, the test-cases presented in the rest of this subsection were appropriately selected in order to introduce the minimum possible overheads in term of wall-clock time.

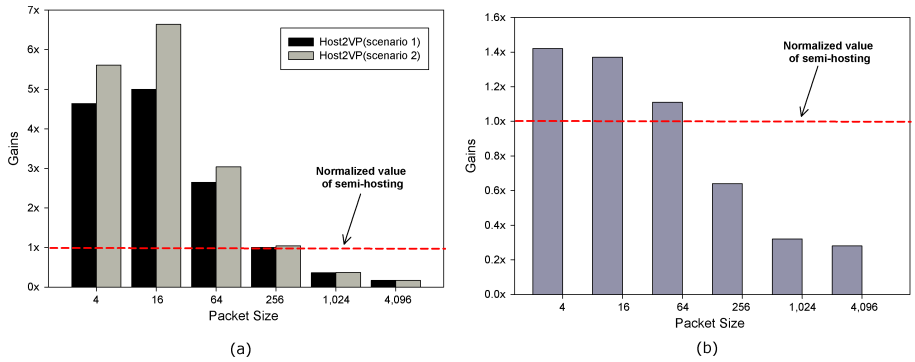


Figure 2.11 Gain in term of wall-clock time for: (a) the proposed Host2VP library and (b) the proposed VP2HW library, as compared to semi-hosting approach [4].

In order to quantify the Host2VP and VP2HW libraries, we have developed two C-based programs which iteratively send and receive data of a 10^6 integers of specific size (ranging from 4 up to 4,096 bytes) between host and VP, as well as VP and target hardware, respectively. The communication overhead for the proposed library is performed by sending a number of 10^6 integers, whereas the results of this analysis are plotted in Fig. 2.11. The horizontal axis in this figure corresponds to the packet size transferred between host PC and VP (Fig. 2.11(a)), as well as between VP and hardware (Fig. 2.11(b)), whereas the vertical ones depict the gains (speedup) in wall-time clock as compared to the semi-hosting approach. At this figure we also plot with dotted line the solution that corresponds to the semi-hosting approach.

Two different scenarios are taken into consideration for the data transfer discussed in Fig. 2.11(a), which are summarized as follows:

- *Scenario 1*: The data transfer is made in segments the size of which is equal to the packet size of Host2VP communication mechanism, as defined at compile time.

- *Scenario 2*: The data is transferred in segments of 4,096 bytes (or 1,024 integers).

Based on this figure we can conclude that for small and medium packet sizes, our solution is up to 5.61× faster than semi-hosting in case of Host2VP, and 1.42× faster in case of VP2HW. Specifically, regarding the Host2VP approach, our library exhibits higher performance for packet sizes smaller (or equal) to 256 bytes, whereas the corresponding packet size for VP2HW is 64 bytes. The limitation of semi-hosting for small and medium packet sizes can be explained by taking into consideration that data transfers with this approach are realized within hard disk (where each data transfer is handled in 4-Kbyte data blocks), as we have already mentioned in Tables 2.1 and 2.2. Hence, by forcing the hard disk blocks to be accessed multiple times, the semi-hosting leads to mentionable overheads. On the other hand, for larger packet sizes, semi-hosting seems to outperform our solution, as less accesses on the hard disk are committed. Despite the previously mentioned analysis, a packet size of 4 bytes corresponds to a more realistic scenario when comparing our solution (TLM-based) against semi-hosting, because the majority of existing processors expose 1-word registers to the bus addresses. In addition to this, the initiator of the specific model employed for our analysis (OpenRISC processor) supports transactions of short length only, even though TLM potentially could support any transaction length.

Fig. 2.12 plots the communication overheads for the two scenarios studied throughout this section, when the packet size is defined constantly equals to 4 bytes. The horizontal axis of these figures denote the number of integers that are sent through the communication link, whereas the vertical axes provide (in logarithmic manner) the time required for this data transfer. This figure indicates the proportional increase of wall clock time with the number of integers that are transferred through the communication link; consequently it is possible to estimate the overhead for any data transfer. The last point indicates also that the gains in wall-clock time discussed previously are independent of the data volume.

Finally, based on Fig. 2.12 we can conclude about the performance efficiency of the introduced communication scheme, as compared to the semi-hosting approach discussed in Fig. 2.11 for the case of the 4-byte packet size. The configuration of HotTalk FPGA transactor significantly affects the performance of co-simulation, as the communication link might be slower than the IP core mapped onto the FPGA. Hence, in order to maximize the efficiency of co-simulation, designers have to employ a suitable (application-oriented) transactor by carefully choosing the size of transaction refill memory (i.e. the amount of data that the arbiter feeds the hardware IP in each transaction). During this selection we also have to take into consideration that larger refill memory imposes additional hardware resources for the transactor's implementation (memory blocks, as well as wider buses for addressing these memories).

In contrast, there is no such monotonic relation between the execution run-time of co-simulation and the size of transactor's refill memory. Regarding the solution introduced in the Plug&Chip methodology, the data rate of the producer-consumer scheme employed by the HotTalk API establishes the most suitable configuration of the transaction's refill memory size per application. For instance, assuming that

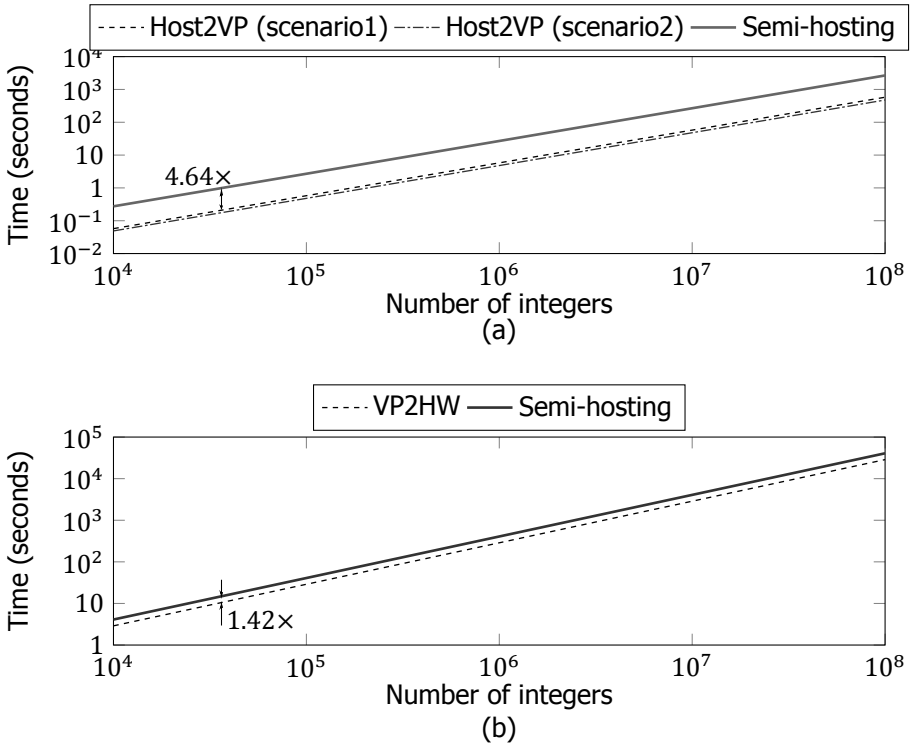


Figure 2.12 Evaluation of communication overhead between: (a) host PC and VP, (b) VP and hardware assuming constant packet size of 4 bytes.

an application's control-path has to process a stream of n bytes and the transactor is configured to load the FPGA with $m < n$ bytes, then a number of iterations equals to $\lceil \frac{n}{m} \rceil$ should be performed, which in turn imposes a degradation in co-simulation speed (since the DUT waits the communication link to deliver the data). DUT utilization refers to the ratio of time which the DUT component is not stalled (due to data transfer) over the total simulation time.

The performance of HotTalk FPGA transactor is quantified with four benchmarks from MiBench suite [22]. The selection of these benchmarks was performed by taking into consideration that the control path for each of them imposes different input/output data rates. Regarding this analysis, the OpenRISC processor was implemented onto the Virtex-6 FPGA board (HTG-V6-PCIE-L240T-2), while the arbiter was connected directly to processor's memory, so that the data input to benchmarks is transferred directly to processor's memory space.

The results of this analysis, which depict how the transactor's configuration affects the overall speed of co-simulation, are summarized in Fig. 2.13. The vertical axes in this figure give the performance of co-simulation in terms of cycles per second (left axis) and the DUT utilization (right axis), while the horizontal axis plots the transactor's memory size. The performance of co-simulation corresponds to the to-

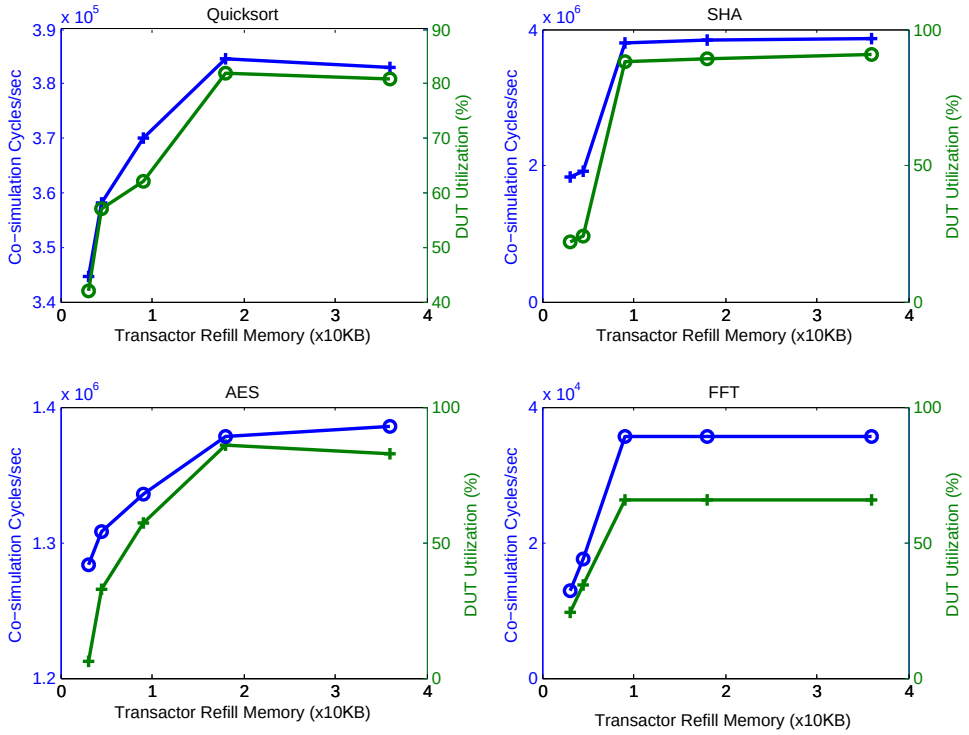


Figure 2.13 Evaluation of the efficiency of introduced co-simulation approach for different benchmarks.

tal cycles for the transactor's (both communication and DUT) execution. Similarly, the DUT utilization defines the percentage of time that the DUT is not stalled due to data transfer.

Based on this figure, we can conclude that the efficiency of the employed transactor in HotTalk API highly depends on the application's inherent requirements for data input/output. In more detail, as we increase the transactor's refill memory, there is a proportional increase to the transactor's efficiency up to a saturation point. For instance, regarding the employed benchmarks, this saturation point is almost 10KB for *SHA* and *FFT* algorithms, while it is 20KB for *Quicksort* and *AES*. Then, the transactor seems to be saturated, and consequently, even if we feed it with a higher rate of data input, this data could not be utilized by the target IP core (DUT).

2.4.2. Evaluating the 3-D OpenRISC

The second part of this section provides a number of qualitative results about the evaluation of digital systems implemented with the usage of 3-D integration process. For this analysis we employ the OpenRISC processor [21] implemented with the usage of TMSC 45nm process technology with 9 metal layers, whereas the parameters affecting the employed TSV library are summarized in Table 2.3.

Fig. 2.14 depicts two different 3-D flavors of OpenRisc processor, as they are

Table 2.3 Characteristics of the selected TSV technology [23].

Diameter	Minimum Pitch	Resistance	Capacitance	Length
1.2um	4um	0.35Ω	2.5fF	4-9um

retrieved after the generation of 3-D stack. Different colors in these diagrams denote blocks that are assigned to different (virtual) layers. More specifically, the first of them, shown in Fig. 2.14(a) corresponds to a min-cut partitioning, whereas the architecture of Fig. 2.14(b) matches to the case where a technology-oriented partitioning is performed. Even though additional 3-D stacks could be derived with $TABU$ algorithm, the solutions depicted in this figure correspond to two representative solutions. Specifically, the min-cut approach leads to the minimum number of TSVs, and hence to yield improvement. On the other hand, the second approach is applicable to designs that include non-compatible technology processes (e.g. logic, battery, RF, sensors, etc) integrated onto a single chip.

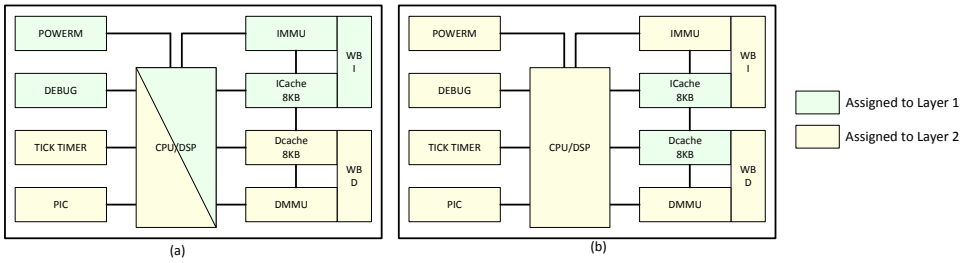


Figure 2.14 Partitioning OpenRISC processors under different constraints: (i) min-cut partitioning and (b) technology-compatible partitioning.

The synthesis of OpenRisc processor is performed with *Cadence Encounter RTL Compiler* under a timing constraint of 10ns (or 100MHz). The derived netlist consists of 13,847 standard cells, 14,541 nets, and 374 I/O ports. Table 2.4 gives some additional technical details about the physical implementation of OpenRisc processor. Based on these results we can conclude that the derived 3-D architectures improves the maximum operation frequency by 1.19× and 1.26× respectively, as compared to the corresponding 2-D system implementation. The physical layouts retrieved from Cadence SoC Encounter for these two architectural approaches are depicted in Fig. 2.15.

2.4.3. Apply Plug&Chip to a Heterogeneous Embedded System - The SPARTAN Project

This subsection describes how we applied our introduced framework to European Space Agency (ESA) initiated project SPARTAN, focusing on the hardware implementation of suitable computer vision algorithms [24]. Rather than similar approaches that tackle this problem solely in software-level, our objective is to develop a number of computer vision (CV) algorithms in ESA compatible VHDL for-

Table 2.4 Metrics about the physical implementation of OpenRISC processor as a 3-D chip.

Characteristics	2-D System	3-D System	
		Solution of Fig. 2.14(a)	Solution of Fig. 2.14(b)
Max. Oper. Frequency (MHz)	95	113	120
Wire-length (μm)	318,750	330,986	323,247
Half-perimeter (μm)	237,791	247,387	234,411
Number of TSVs	0	272	408
Area for TSVs (μm^2)	0.00	1395.36	2093.04
Aspect ratio	1.00	2.00	2.00
Area per layer	149,344 μm^2	149,368 μm^2	149,369 μm^2

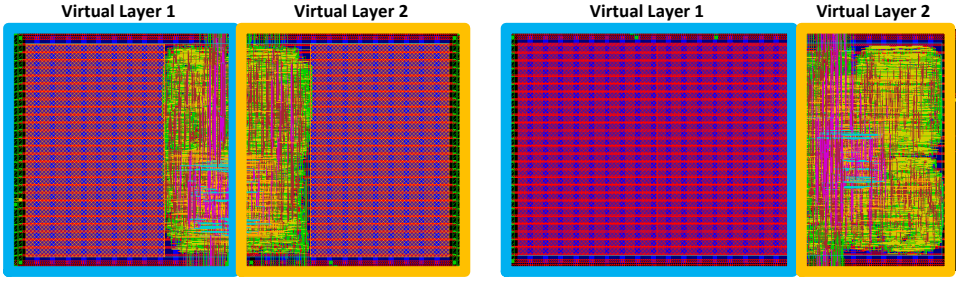


Figure 2.15 Physical implementation of 3-D OpenRISC with the usage of Cadence SoC Encounter for the partitioning discussed in Figs. 2.14(a) and 2.14(b), respectively.

mat, whereas the implementation of SPARTAN system will be demonstrated using a co-design methodology targeting a host CPU and a Virtex-6 FPGA device. More specifically, the CPU provides all the standard operating system services required for proper rover navigation, such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management, whereas the FPGA acts as an accelerator for the computational intensive CV algorithms. Additional details about the architecture of SPARTAN system can be found in [25].

The successful completion of the project imposes that a number of specifications have to be met (i.e. high-performance while maintaining efficiency in terms of energy use, computing power and memory footprint). Starting from an all software solution in C/C++, we identified the CV algorithms (or part of them) that introduce performance bottleneck in systems' execution and have to be accelerated through mapping onto reconfigurable hardware. Since the overall performance of the SPARTAN system (depicted in Fig. 2.16) highly depends on the input signals (e.g. image data), the system exhibits increased amount of dynamism, i.e. its execution cost (e.g., number of processor cycles, memory footprint, energy) and quality metrics (e.g., algorithm's accuracy, system's mean error) are environment dependent (e.g., input data). Hence, during the project, we faced the challenge to estimate the impact of this dynamic behavior at design time.

In order to handle the dynamic nature of the developed system, we adopted

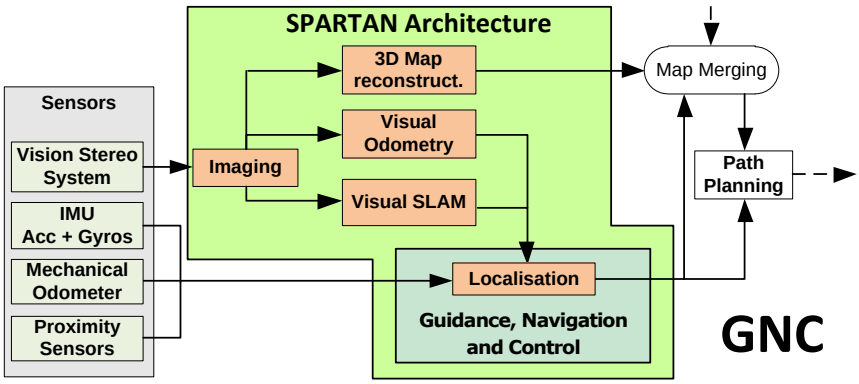


Figure 2.16 Schematic overview of the SPARTAN system.

the concept of system scenarios presented in [26]. According to this approach, the system behaviors that are similar from a multi-dimensional cost perspective, such as resource requirements, delay, and energy consumption, are grouped in such a way that the system can be configured to exploit this cost similarity. The concept of using scenarios in embedded systems domain initiates from the different Run-Time Situations (RTSs), in which a system may run on a given platform. Regarding the SPARTAN project, the parameters of the CV algorithms, as well as those that affect the co-design architecture, result to different RTSs, as they highly affect a quality cost metric (e.g. FPGA resources, CPU frequency, throughput etc). In general, any combination of N cost dimensions may be targeted with this approach. However, the number of cost dimensions in conjunction to all the possible values of the considered RTS parameters usually leads to an exponential number of RTSs. As a result, the design process is being complicated while the run-time overhead of studying all these RTSs is excessive high. A viable solution to this problem relies on appropriately clustering these parameters from an N-dimensional cost perspective into system scenarios [26]. Even though different mechanisms for supporting the prediction of optimum scenario under run-time constraint, as well as for performing scenario switching, are available [26], they are applicable mostly in homogeneous platforms. In contrast, the SPARTAN project was built on top of a co-design CPU-FPGA architecture; thus the concept of using scenarios has to be has to adapted to the employed underline architecture.

After a detailed profiling, we conclude that the most computational intensive CV kernels were the Disparity algorithm (a submodule of 3D Map Reconstruction) and the SURF algorithm (submodule of Visual Odometry) [27]. These algorithms include an increased number of parameters that highly affect the performance of final system. Also, since there is a dependency among the functionalities of SPARTAN system, during the development phase it is important to iteratively control and test

the impact of algorithms parameters against the specifications of the final system.

While for a software only project this testing procedure may be controllable due to the uniform and homogenous implementation platform (e.g. all modules under development are using the same API and shared libraries while running on top of the same OS), this does not affect the SPARTAN project where the implementation platform is a heterogeneous architecture. This limitation is overcomed by apllying the introduced Plug&Chip framework. Specifically this framework provides to the consortium the opportunity to quantify the impact of RTS parameters in CV algorithms both at early design phases, as well as throughout the whole development phase.

Figure 2.17 Evaluation of SPARTAN system using a scenario-based trade-off analysis.

As a proof of concept, Fig. 2.17 gives the SPARTAN trade-offs when four distinct scenarios were studied. While this information is usually available at the final design stages, where the complete system is functional, the Plug&Chip framework evaluates the impact of 16 RTS parameters as long as only two IPs were developed in VHDL. These IPs were tested in true silicon (FPGA), with real-world testbenches, through the Host2VP and VP2HW methods, while the under-development system was emulated on a VP. The trade-offs of Fig. 2.17 aforemention the average execution time per frame and the mean error of rover's Visual Odometry (as compared to the real displacement) versus the average number of detected features. As long as

the number of detected features is increased, the system is able to provide better location estimates and thus the displacement error is reduced. However, such an improvement in accuracy comes with a penalty in execution time, as more features need more time to be processed.

As it is depicted in Fig. 2.17, a Pareto surface of potential exploitation points is derived in the 3-dimensional exploration space. Such an exploration enables the establishment of the optimal configuration RTS-parameters under the project's specification (i.e. compute location estimates faster than 1 second, as it is imposed by the SPARTAN specifications). Thus, while for scenarios 1 and 2 the system can be configured to produce up to 200 features on average, while operating in less than a second and providing high accuracy (in term of error), this does not occur in case scenarios 3 and 4 are employed. In such a case, the design team could appropriately modified the already taken decisions in order to further optimize the entire system and meet the specifications for all the scenarios.

Figure 2.18 Evaluation of SPARTAN system towards the efficiency of introduced co-simulation approach.

We have to mention, that the configuration of HotTalk FPGA transactor significantly affects the performance of co-simulation (as it was already mentioned in subsection 4.1). In order to quantify this performance enhancement, Fig. 2.18 plots the impact of transactor's refill memory to the co-simulation speed, as well as the DUT utilization. Based on our experimentation, we found that up to 30KB there is a almost linear correlation between the number of succeeded co-simulation cycles with the DUT utilization. Note that the DUT utilization is too low since the slow 100Mbps Ethernet link of HotTalk pauses the pipelined operation of SPARTAN

system. However from such an analysis a designer shall calculate in detail the communication overhead of HotTalk FPGA transactor and thus estimate the performance of the final system.

Figure 2.19 represents a snapshot assessment of SPARTAN system at the ROS environment, under the framework Plug&Chip. As shown, the system may be simulated in realistic scenarios during the early design stages, where the algorithms are not completed in their final form. In this instance, the three machine vision algorithms, depicted in pop-up windows, are running in an FPGA device, through the framework Plug&Chip, while all the other motion estimation algorithms of the robotic system (blue and red course lines), as well as the the virtual environment software shown in Figure 2.19 are running on a host computer, through the ROS operating system and the OpenCV library.

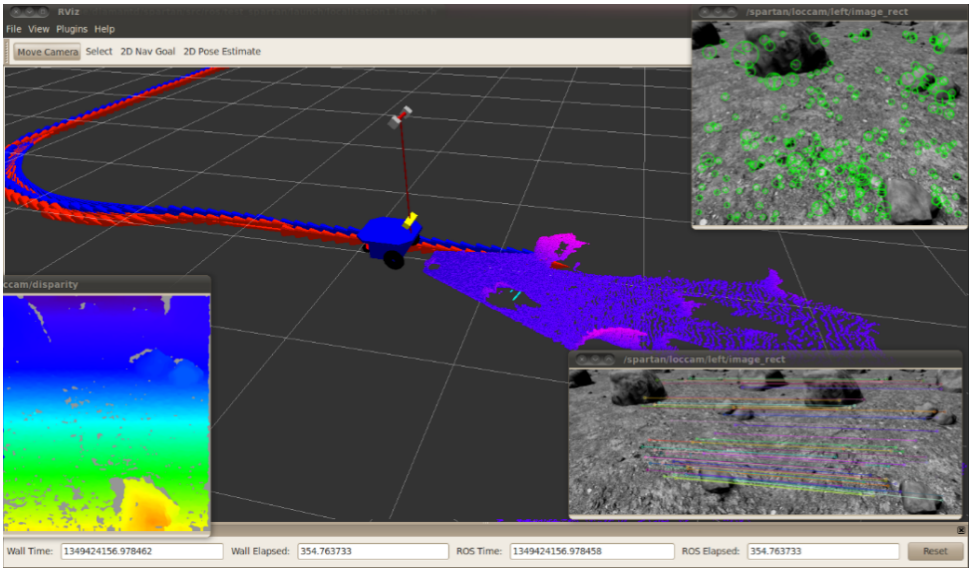


Figure 2.19 Snapshot assessment of SPARTAN system on ROS/Linux environment, using the framework Plug&Chip, on a PC-FPGA co-design platform.

Moreover, Figure 2.20 depicts a snapshot assessment of a car engine control system (Engine Control Unit - ECU). Scope of the system is the motion path control (track control) with the help of machine vision algorithms [7]. In this instance, the SURF machine vision algorithm is executed in a FPGA device, through the framework Plug&Chip, while the rest of the system is running in a virtual simulation environment for the context of automotive racing [29]. This work targets the interest points identification of successive images from the front position of the vehicle, in order to optimize navigation. In this application context the Plug&Chip framework enabled the testing of multiple scenarios of use under-development ECU even with

[7]The system was designed in a virtual prototyping contest for research papers on thesis level, from the company Cadence, for the automotive industry [28] and awarded the 2^{nd} prize.

the lack of real car and environment standards. This early assessment of the SURF algorithm optimization helped in many operating scenarios. Further information on the SURF algorithm can be found at [30].

2.5. Conclusion

This chapter presented a novel framework for supporting rapid, as well as incremental prototyping, of heterogeneous 2-D and 3-D embedded systems. Among others, the Plug&Chip framework provides to designer teams the desired connectivity between the hardware-depended software, the control-flow software, as well as the custom hardware IPs. Such a feature enables starting the development, testing and validation of the embedded software substantially earlier than it has been possible in the past. Experimental results with various testcases, spanning from simple kernels up to heterogeneous HW/SW embedded systems, prove the effectiveness of the introduced framework, as it provides a more efficient co-simulation.

References

[1] V. F. Pavlidis and E. G. Friedman, *Three-dimensional Integrated Circuit Design* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009).

[2] ITRS, *International technology roadmap for semiconductos,* (2012).

[3] Synopsys, *Hybrid prototyping delivers the best of both virtual and fpga prototyping to soc hardware and software teams,* .

[4] OVP, *Open virtual platforms (ovp), online: www.ovpworld.org,* (2013).

[5] R3LOGIC, *R3logic inc.* (2013).

[6] CADENCE, *Cadence 3d-ic solution,* (2013).

[7] T. Borgstrom, E. Haritan, R. Wilson, D. Abada, R. Chandra, C. Cruse, A. Dauman, O. Mielo, and A. Nohl, *System prototypes: Virtual, hardware or hybrid?* in *Design Automation Conference, 2009. DAC '09. 46th ACM/IEEE* (2009) pp. 1–3.

[8] SCE-MI, *Standard co-emulation modeling interface (sce-mi) reference manual,* (2011), http://www.accellera.org/downloads/standards/sce-mi/SCE_MI_v21-110112-final.pdf.

[9] Altera, *Altera virtual target,* (2013), http://www.altera.com/devices/processor/arm/cortex-a9/virtual-target/proc-a9-virtual-target.html.

[10] QEMU, *1st international qemu users forum.* (2011), http://adt.cs.upb.de/quf/.

[11] T.-C. Yeh, Z.-Y. Lin, and M.-C. Chiang, *A novel technique for making qemu an instruction set simulator for co-simulation with systemc,* in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2011*, Vol. I (2011) pp. 288 – 291.

[12] J. Jovic, S. Yakoushkin, L. Murillo, J. Eusse, R. Leupers, and G. Ascheid, *Hybrid simulation for extensible processor cores,* in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012* (2012) pp. 288–291.

[13] HAPS, *Synopsys high-performance asic prototyping systems,* (2013), http://www.synopsys.com/Systems/FPGABasedPrototyping/Pages/HAPS.aspx.

[14] Xilinx, *Xilinx hybrid co-simulation flow quick reference,* (2013), http://www.xilinx.com/tools/feature/14_1_isim_hw_cosim_qrg.pdf.

[15] Xilinx, *Xilinx isim rtl simulator,* (2013), http://www.xilinx.com/tools/isim.htm.

[16] OpenCores, *Ethernet mac 10/100 mbps,* (2013).

[17] IP-XACT, *Ip-xact technical committee,* (2013).

[18] N. Selvakkumaran and G. Karypis, *Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization,* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **25**, 504 (2006).

[19] K. Siozios and D. Soudris, *A tabu-based partitioning and layer assignment algorithm for 3-d fpgas,* Embedded Systems Letters, IEEE **3**, 97 (2011).

[20] I. Savidis, S. M. Alam, A. Jain, S. Pozder, R. E. Jones, and R. Chatterjee, *Electrical modeling and characterization of through-silicon vias (tsvs) for 3-d integrated circuits,* Microelectronics Journal **41**, 9 (2010).

[21] OpenRISC, *Openrisc 1000 project main page,* (2012), http://opencores.org/or1k/Main_Page.

[22] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, *Mibench: A free, commercially representative embedded benchmark suite,* in *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*, WWC '01 (IEEE Computer Society, Washington, DC, USA, 2001) pp. 3–14.

[23] S. Gupta, M. Hilbert, S. Hong, and R. Patti, *Techniques for producing 3d ics with high-density interconnect,* in *Proceedings of the 21st International VLSI Multilevel Interconnection Conference* (2004).

[24] SPARTAN, (2013), http://proteas.microlab.ntua.gr/spartan.

[25] I. Kostavelis, L. Nalpantidis, E. Boukas, M. A. Rodrigalvarez, I. Stamoulias, G. Lentaris, D. Diamantopoulos, K. Siozios, D. Soudris, and A. Gasteratos, *Spartan: Developing a vision system for future autonomous space exploration robots,* Journal of Field Robotics **31**, 107 (2014).

[26] S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. D. Bosschere, *System-scenario-based design of dynamic embedded systems,* ACM Trans. Des. Autom. Electron. Syst. **14**, 3:1 (2009).

[27] D. Diamantopoulos, K. Siozios, G. Lentaris, D. Soudris, and M. Rodrigalvarez, *Spartan project: On profiling computer vision algorithms for rover navigation,* in *Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on* (2012) pp. 174–181.

[28] Cadence, *Cadence thesis contest for automotive embedded systems, cdnlive! emea 2013,* .

[29] SD, *Speed dreams: An open motorsport simulator,* .

[30] G. Lentaris, I. Stamoulias, D. Diamantopoulos, K. Siozios, and D. Soudris, *An fpga implementation of the surf algorithm for the exomars programme,* Workshop on Reconfigurable Computing .

2

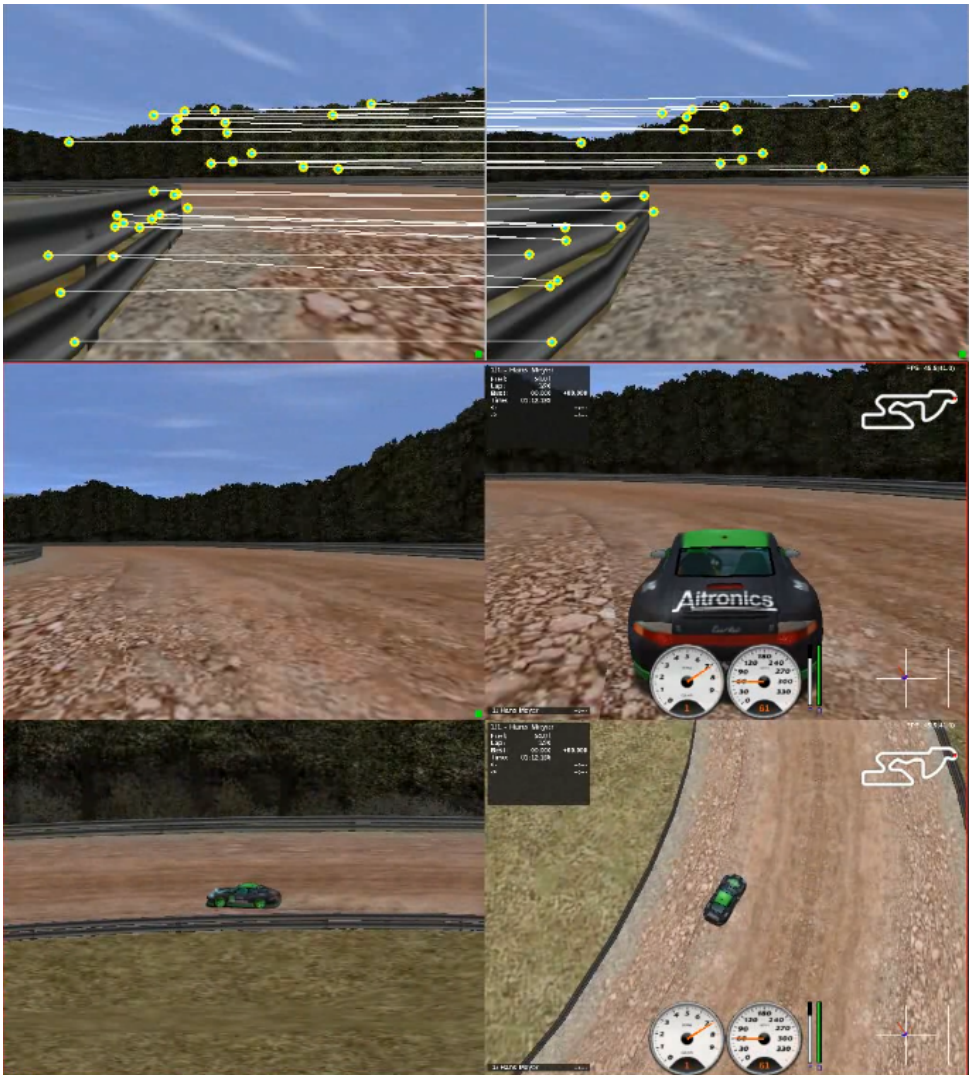


Figure 2.20 snapshot assessment of a car engine control unit (ECU) on Linux host, using the framework Plug&Chip for virtual prototyping of CV algorithms, on a PC-FPGA co-design platform.

3

Cross-Layer Synthesis of Heterogenous Architectures

This chapter presents cross-layer synthesis techniques for heterogeneous application-specific architectures. There are analyzed the peculiarities of the architecture and key contemporary problems of the design of heterogeneous application-specific systems. The proposed methodology targets the optimization of conflicting design metrics required by the modern semiconductor market, such as the energy consumption, the silicon are, the temperature tolerance and the reliability. Also it is introduced a hardware/software co-design flow for heterogeneous platforms. The proposed methodologies are evaluated through the study of four heterogeneous platforms, i.e. a SoC architecture for telecommunications (SDR), a NoC architecture for multimedia applications, a computer vision system for space rover navigation and a SoC architecture designed with three-dimensional integration technology. Chapter 3 is associated with the general methodology of the thesis presented in Section 1.2, through the contribution of the categories of "Application-Specific Platforms - EDA Tools systems 2D/2.5D/3D integration", "Acceleration Datapath Synthesis" and "System Architecture Exploration".

3.1. Thermal and Reliability Aware SDR Architectures

3.1.1. Introduction

Communication has become one of the central uses of computing technology over the years. Architectures that facilitate communication, such as mobile phones and wireless networks have been primary factors in driving the evolution of microprocessors and computer systems. With the evolution of wireless mobile communications, the problem emphasis has shifted to networking protocols and signal processing that are required to sustain the necessary bandwidth of these applications.

In recent years, we have seen the emergence of an increasing number of wireless protocols (e.g. 2G, 3G, GPRS, WiFi, etc) that are applicable to different types of networks.

Software Defined Radio (SDR) technology was created to improve interoperability between different wireless networks, field radios, and devices [1]. SDR technology is comprised of both software and hardware that can be dynamically reconfigured to enable communication between a wide variety of changing communications standards, protocols, and radio links. With this latest SDR technology, system architects are able to create multi-mode, multi-band, and multi-functional wireless devices and network equipment that can be dynamically reconfigured, enhanced, and upgraded through software updates and hardware reconfiguration.

Since the majority of these systems exhibit high-throughput, low-power requirements and short time-to-market, previous studies proposed the usage of System-on-Chip (SoC) architectures to support the efficient implementation of SDR [2, 3]. Meeting the thermal constraints and reducing the temperature hotspots at these platforms are critical tasks in order to design reliable systems. Furthermore, since chip's temperature has significant impact on performance, reliability, power consumption, as well as cooling and packaging costs, it should be carefully optimized at design time. Thermal-aware design is difficult, whereas designing a chip and package for the worst-case power consumption scenario may be prohibitively expensive.

For this purpose, thermal management have recently received a lot of attention by design architects. The goal of thermal management is to meet maximum operating temperature constraints, while tracking timing specifications. Moreover, thermal management can also achieve further temperature reduction in order to improve the reliability degradation of SoCs.

Previous studies shown that thermal stress is tightly firmed to reliability issues [4] . For instance, thermal cycling can be modeled with the Coffin-Manson relation, which relates in an exponential way the number of cycles to failure to the magnitude of thermal cycling [5]. Existing approaches aim to perform thermal management with techniques that come from the power reduction domain.

Typical instantiation of this solution is the usage of Dynamic Voltage and Frequency Scaling (DVFS) [6]. Due to the approximately quadratic relation between supply voltage and power consumption, DVFS-based techniques achieve to provide mentionable savings in power consumption, but they impose slower operation frequency. Moreover, these techniques cannot guarantee that temperature hotspots and/or temperature gradients will be reduced, since there are applied during runtime as a reaction to chip's thermal crisis.

Another technique for providing temperature hotspot elimination, especially at multi-core architectures, is based on load balancing [7]. Even though these techniques have been studied for general purpose parallel computers, they targeted mainly the avoidance of performance bottlenecks rather than thermal issues.

A similar approach is discussed in [8] where tasks are swapped between hot and cool cores in order to control temperature values across the target architecture. However, this approach assumes that threads are transferred among different

cores, which cannot provide focused thermal management (this would be feasible only if functionality transfer is also supported instead of transferring threads). Furthermore, there are available previous works [9, 10] that perform thermal management through compiler optimizations. The main limitation of this approach affects the difficulty to estimate with sufficient accuracy the temperature variations occurred due to revisions of source code. Furthermore, this technique is applicable only to a small percentage of SoCs, since the required architectural details are not always freely available. In order to alleviate these limitations, instruction-level accounting techniques that are based on empirical measures have been employed.

A common drawback among techniques discussed up to now is that they do not incorporate any mechanism for handling thermal history of the cores. This feature provides useful guidelines about the future behavior of the system and can be exploited to improve the results of the migration. In addition to that, existing approaches mainly provide thermal aware application mapping onto SoC devices based on exploration provided through simulation results. These approaches assume that target platform is fixed ignoring about potential improvements achieved through architecture-level optimizations [11]. In this work we propose a new methodology, as well as the software supporting framework, for performing architectural and physical design under constraints posed by temperature hotspots. Specifically, the motivated idea introduced in this research work exploits the selective replication of hardware blocks that exhibit increased power densities. Then, by appropriately assigning tasks onto these replica blocks, it is feasible to alleviate the chip's thermal stress.

The proposed approach aims at temperature optimization, while it can be considered as a proactive strategy that alleviates thermal stress at run-time. The introduced framework does not impose any architectural or compiler modification, whereas it is orthogonal to any other thermal-aware methodology discussed above, since it is based on new architectural schemes to eliminate the consequences posed by temperature hotspots. Thus, existing work on thermal aware application mapping and dynamic thermal management can be used in a modular manner to extend the proposed methodology.

Specifically, we target at the development of an automated design space exploration framework that extracts and evaluates a large number of architectural solutions. Every solution exploits selective block replication. Based on the software supported automatic exploration, we are able to compute higher thermal quality Pareto curves, in contrast to many similar existing optimization approaches that retrieve only a single architecture [8, 9]. Hence, architects can trade-off between the desired level of temperature reduction at hotspots and the resulting timing/area/power overheads. Furthermore, the supporting tool framework provides a considerable speedup at the exploration procedure.

Previous works introduced the usage of parallelism in order to achieve power savings, which in turn lead to temperature reduction [12]. A parallel implementation of a design essentially replicates component(s) of the design such that parallel branches process interleaved input samples. Therefore, the inputs coming into each parallel branch can be effectively down-sampled. An output multiplexer is

needed to recombine the outputs, and produce a single data stream.

The main differentiation of the proposed research approach, as compared to this approach, is that our solution does not assume that replica blocks of the same type are working in parallel. More specifically, in our methodology, only one of the available replica blocks are active at any time. The selection of this active block is based on its thermal condition, as it is described in upcoming sections. The contributions of this research work can be summarized, as follows:

- We show the optimization potential regarding thermal aware exploration by exploiting selective replication of specific architectural blocks.
- We introduce of a novel methodology targeting to provide: (i) elimination of thermal hotspots at SoCs targeting SDR architectures and (ii) alleviation to the temperature gradients.
- Rather than providing only one architectural solution, our methodology retrieves a number of Pareto architectural solutions, each of which trades-off different design constraints/criteria.
- We propose a novel design methodology that is orthogonal to the existing approaches found in relevant literature [9] [7] [6] [8] [10] [13] [14] [15] [16].
- We provide CAD support through developing a software supported thermal aware exploration framework, which is public available for additional extensions through [17].
- We apply the proposed methodology to a real case SoC design consisting of a synthesized LEON3 processor [18].

Experimental results prove the efficiency of the proposed methodology, showing that the selected architecture leads to temperature reduction about 8% (from 380 Kelvin to 363 Kelvin), with a controllable silicon area increase of 15%. As we show latter, such a temperature reduction apart from reduction in cooling cost, also achieves mentionable improvement to the consequences posed by aging phenomena about 14%.

The rest of the research work is organized, as follows: subsection 3.1.2 introduces the underline SDR architecture, whereas subsection 3.1.3 discusses motivational observations that guide us to propose selective insertion of replica blocks. Subsection 3.1.4 describes in a brief manner the micro-architectural enhancements needed for applying selective block replication in existing micro-processor architectures. The proposed methodology is analyzed in detail in subsection 3.1.5, while a number of evaluation results are discussed in subsection 3.1.6. Finally, subsection 3.1.7 concludes the paper.

3.1.2. Target SDR Architecture

During the last years a number of different SDR-based architectures have been developed, whereas a typical instantiation is depicted in Figure 5.15. The front-end is

responsible for converting the signal between the RF domain and an intermediate frequency, and the A/D and D/A components convert the signal between the analogue and the digital domain. In our analysis the baseband functionality is carried out on software running on a System-on-Chip based on LEON3 embedded processor [18].

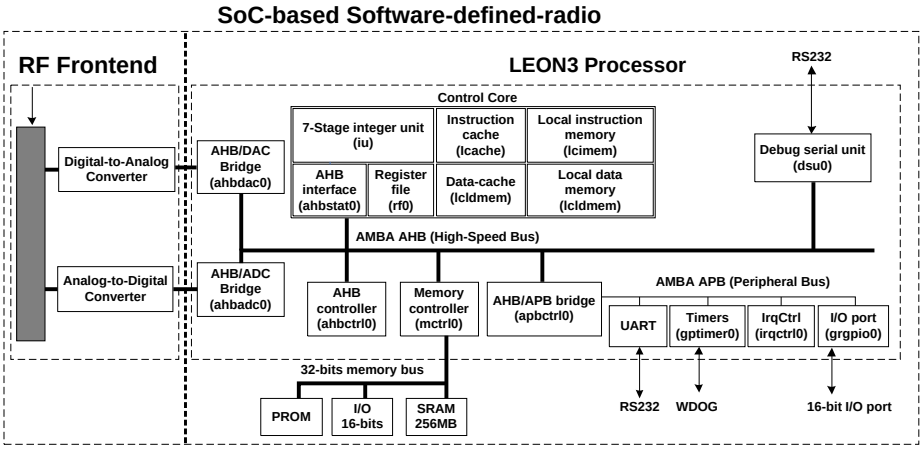


Figure 3.1 The block diagram of employed SoC-based SDR.

LEON3 processor consists of the integer unit, the cache subsystem, the memory management system and the AMBA interface. The instruction unit is fully compatible with the SPARC V8 instruction set, whereas the pipeline consists of 7 stages. The integer unit has configurable separate instruction and data cache (Harvard architecture), whereas the size for each of them is equals to 1Kbyte. Furthermore, the integer unit includes a configurable register file with register window equals to 8. Regarding the L1 caches, they are managed by a cache controller which is interfaced to the system's AMBA AHB bus. The communication to LEON3 peripherals is performed with two bus controllers, referred as AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) controller, respectively. The first of these controllers (AHB controller) is used for the connection of high speed components (i.e. integer unit, memory controller, etc), whereas the second one (APB controller) provides control to the low-speed peripherals (i. e. UARTs, I/Os, etc). Finally, LEON3 processor contains a configurable separate local data (2KByte) and instruction memory (2KByte).

3.1.3. Motivation

In this section we discuss the importance of different hardware blocks to be considered as critical for thermal stress. This problem becomes even more important regarding either high-end processor architectures, i.e. superscalar organizations, or multi-core SoC designs, where multiple hardware components, each of which

with different area and power values, are combined into a single device. Hence, one of the challenges that architects are facing today is to identify the hardware components that higher affect thermal stress.

In order to show how different hardware block thermal profiles affect the thermal stress of the entire IC, Figure 5.14(a) gives the power consumption for the components of the LEON3 processor, when the SDR system is executed.

We select such an embedded processor because it is widely used in numerous commercial and/or research products. However, apart from the selected target platform, the methodology we follow in this research work is also applicable to any other digital architecture.

Since embedded cores usually are designed with low power criterion, many researchers up to now pay effort to reduce maximal temperature values by identifying blocks that dissipate increased power budges. Regarding the LEON3 processor, the local data/instruction memories, the L1 data/instruction caches, as well as the register file are found to be the most power hungry blocks. More specifically, the average power consumption at these blocks, as compared to the total power dissipation, is 57%, 31% and 8%, respectively.

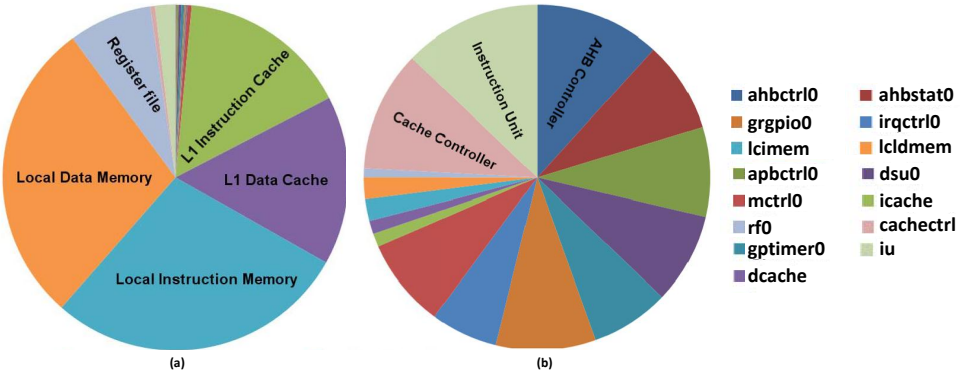


Figure 3.2 (a) Power consumption and (b) Power density pies for LEON3 architecture.

Even though Figure 5.14(a) provides a first order metric about the components with increased power consumption, we show that it is not enough in order to retrieve conclusions about their importance regarding the thermal stress. This occurs since the power metric does not take into consideration the area of underline hardware block, which is especially crucial for thermal spreading. Hence, a more representative metric should be employed in order to evaluate the importance of each core into the chip's temperature values.

A candidate metric for this scope is power density, which denotes the ratio of power consumption for each hardware block per the area occupied by this block. Figure 5.14(b) gives the corresponding power density pie chart regarding the LEON3 architecture. As we can conclude from this figure, the components with increased power densities are not those identified as critical based solely on the power consumption criterion. More specifically, the power density denotes that AHB controller,

instruction unit and cache controller are the blocks with increased impact on thermal stress. These blocks contribute to the total power density about 12%, 13% and 11%, respectively, whereas the five blocks already identified based on power consumption correspond to 5% of total power density. This occurs mainly since the blocks with increased power consumption have also considerable increased area (about 91% of the total architecture's area), which in turn leads to almost negligible power density values for these blocks.

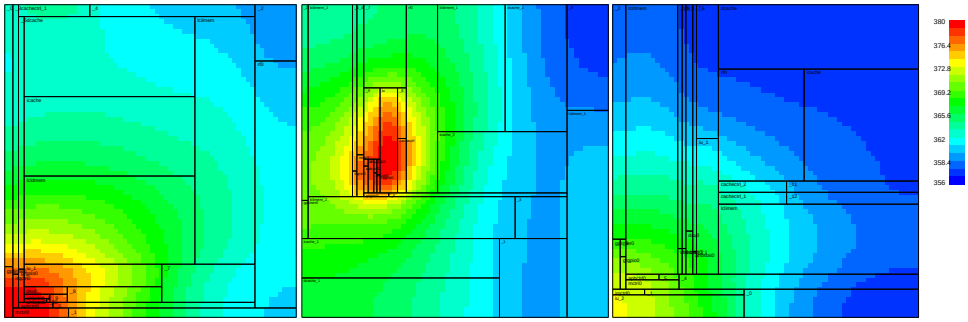


Figure 3.3 Thermal profile for LEON3: (a) without considering replica blocks, (b) with replica blocks (2×local data/instruction memories, 2×L1 data/instruction caches, 2×register file), and (c) with replica blocks (2×instruction unit, 2×cache controller, 2×AHB controller).

Next we depict that the criterion of power density is much more important than the corresponding one about power consumption. For this purpose, Figs. 5.19(b) and 5.19(c) give the thermal profiling as they derived with Hotspot tool [19] about a LEON3 processor running SDR applications (e.g. filters, encoding/decoding, etc), when the five and three most critical components retrieved with the previously mentioned analysis, respectively, are replicated two times. In order to perform this replication of hardware blocks we incorporate the methodology introduced in this research work. We note here that all of these floor-plans were retrieved with HotFloorplan tool [20].

As a reference point for this study we use the thermal map for a LEON3-based SoC SDR architecture when no replica blocks are assumed. This map, shown in Figure 5.19(a), exhibits a temperature hotspot in region where the blocks with increased power densities (AHB controller, instruction unit and cache controller) are floor-planned. This hotspot results in increased temperature value, as compared to the average on-chip temperature, about 7%.

Similar to previous conclusion, the floor-plan that leads to architecture instantiation where components with increased power consumption are replicated (Figure 5.19(b)), also results in thermal stress. This occurs since the replicated components exhibit low power density, and hence they do not have considerable impact on thermal stress. However, we have to mention that this approach exhibits slightly reduced maximum temperature values, as compared to architecture without replica blocks, since it increases device area (the replicated blocks occupy about 91% of the total chip's area). On the other hand, the components with increased power

densities still contribute to thermal stress, as it is shown at Figure 5.19(b).

Note that for shake of completeness, the temperature scaling is constant among for all the thermal maps depicted in Figs. 5.19(a), 5.19(b) and 5.19(c), in order to be clear that only careful replication of blocks with increased power densities can alleviate thermal stress.

3.1.4. Micro-Architectural Considerations

For supporting selective block replication, the processor micro-architecture has to be properly enhanced. We mention that in this research work, we focus mainly on the exploration methodology developed for the automatic evaluation of opportunities delivered through selective replication. Thus, in this section we briefly introduce some micro-architectural considerations that enable the design of processor architectures with replicated components.

In the general case, the data flow and the control flow of the original processor architecture has to be modified towards two directions: (i) enabling mutual exclusiveness between the replicated units, and (ii) permitting run-time management of the replicated resources according to the run-time thermal state of the processor. We focus our analysis on the RISC micro-architecture of LEON3 [18] embedded processor. The block diagram of a SDR system based on LEON3 processor was already depicted in Figure 5.15.

We target on lightweight enhancements in original LEON3's data-path to avoid extensive area, organization and control overheads in respect to the original data-path. For this purpose, we apply selective replication in a coarse grained manner i.e. replicating at the level of instruction unit, rather than at the ALU unit or the instruction fetch level. Furthermore, we avoid replication of the actual memory components (i.e. register file, data cache etc.), since their replication will require proper control mechanisms to establish data coherency among the various replicas.

Although, selective replication in finer granularity than the proposed is a valid design option, we show that coarse-grained component replication can achieve significant temperature reduction and hotspot elimination, which in turn results among others in device improvement against aging phenomena. The proposed approach is not a restrictive one. As shown in Figure 5.14(b), except the register file which is excluded for replication, the rest of the maximum power densities inside a LEON3 processor are distributed among the replicated components, specifically the instruction unit (IU), the cache controller ($CACHE\_CTRL$) and the AHB bus controller (AHB_CTRL).

According to previous analysis, we propose the adoption of a micro-architectural extension similar to the one depicted in Figure 5.16. In the examined case, we assume that each component has been replicated two times (this number is parametric in our methodology and its value is defined by the device architect). Each replicated module is enhanced with operand isolation latches [21] to lower redundant dynamic power by eliminating switching activity in time windows in which the component remains inactive. Leakage optimization techniques, i.e. power gating, can also be applied in an orthogonal manner. However, in the context of this research work, we only account for dynamic power operand isolation techniques, thus

assuming that the inactive component leaks power during its idle time window.

Furthermore, for each replicated component proper pairs of multiplexing and de-multiplexing logic are added to the original data-path, regarding the lightweight control and data flow extension. Specifically, the inputs of each replicated component are driven by the de-multiplexer that properly guides the input data to the active module. Accordingly, the output signals from each replicated component are multiplexed in order to propagate to the next level. We recognized two data-flow paths inside the processor data-path, namely the memory-to-instruction unit and the instruction unit-to-memory data-flow paths. Figure 5.16 depicts the combination of the two data-flow paths by traversing the architecture graph either in a top-down (memory-to-instruction unit) or in bottom-up (instruction unit-to-memory) manner.

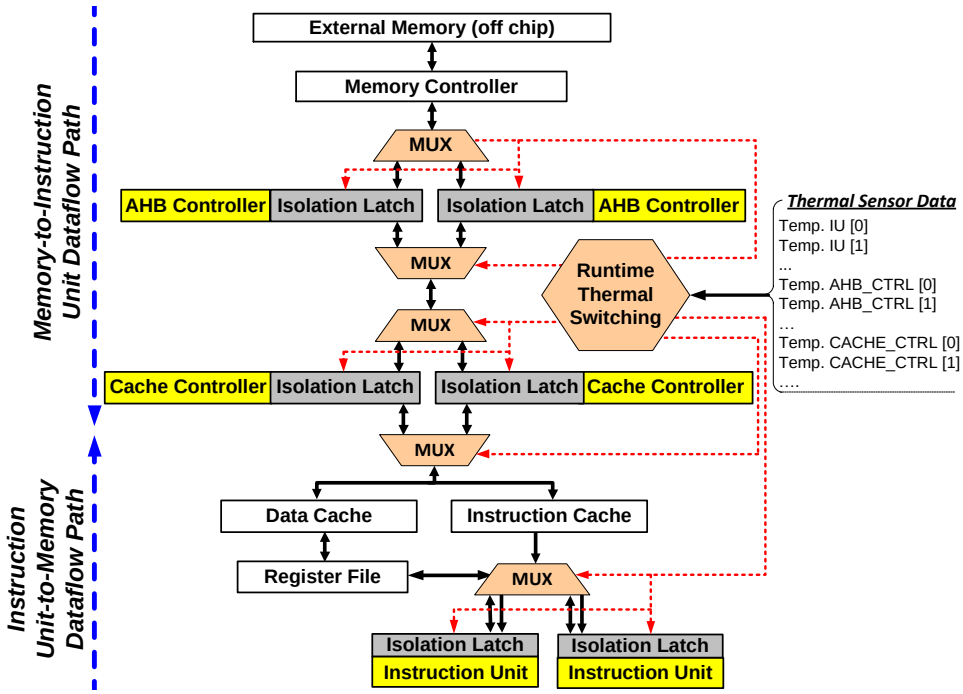


Figure 3.4 Proposed micro-architectural enhancement.

The original LEON3 architecture is also enhanced with a thermal aware runtime controller module for distributing the workload to the available units during runtime. The thermal aware workload distribution is performed by properly issuing the selection signal to the added multiplexing and operand isolation logic. Actually, the same signal configures both the aforementioned components. Since only the selection signals to the extra logic are issued, the thermal aware controller works transparently from the control logic of the rest of the LEON3 architecture. The controller makes the decision which replicated unit to be turned-on/off according

to the thermal state of the processor. It is assumed that runtime thermal data are available, i.e. through thermal sensors.

We consider a reactive scheme of the thermal aware controller. Thus, the controller alters its state whenever an upper temperature threshold, T_{thres} is crossed. As pointed in [22], the runtime temperature threshold is set to lower value than the one estimated during design time exploration to guarantee proper functionality during execution.

The thermal controller reacts to the temperature readings provided by the on-chip thermal sensing infrastructure. $Temp_k[i]$ refer to the temperature value read by the thermal sensor of the replica block i, $i \in \{0, Max_{replicas}\}$ that will be active for execution on the next clock cycle, regarding the unit type k, $k \in \{IU, CACHE\_CTRL, AHB\_CTRL$ The parameter $Unit_k[i]$ is a table of 1-bit registers. Each one of these registers latches the on/off - $Execute/Isolate$ signal for the replica i of unit type k.

The state transitions of the thermal aware runtime controller for a single type of unit, i.e. the instruction unit, are depicted in Figure 5.17. The same control logic is applied to every type of replicated block, k. Since each type of replicated block, k, is managed individually, the overall thermal controller is structured in a modular manner by several control paths like the one depicted in Figure 5.17, each one dedicated to a specific type of block type. If the monitored temperature is lower than the defined threshold, $Temp_k[i] < T_{thres}$, the controller remains in the same state. Each time the monitored temperature for the active replica block, i.e. $Unit_k[i]$, crosses the maximum temperature threshold, $Temp_k[i] < T_{thres}$; the controller enters to an intermediate state. In this intermediate state, the controller maintains its previous configuration for a number of cycles that equals the pipeline stages, $\#PipelineStages_k[i]$ of the specific $Unit_k[i]$. Thus, the controller waits the pipeline of the component to complete its execution, in order to avoid data hazards. In addition, during the intermediate state, the coolest unit among the replicated ones of the same type is extracted using the monitored temperature data, $NextUnit_k := Unit_k[Temp_{Min}]$. When the pipeline completes its execution, the controller issues the steering signals to the extra multiplexors and operand isolators and reconfigures the control and the data-flow of the processor architecture to be performed by the new selected unit (the coolest candidate at the specific time window).

3.1.5. Proposed Methodology

This section describes in detail the proposed methodology for reducing temperature hotspots through selective replication for some hardware modules of the target architecture. Note that throughout this methodology we do not aim at redesigning the whole micro-architecture, but we focus only to critical components. The goals of this methodology are: (i) to provide a proactive thermal-aware approach targeting at micro-architecture designs, and (ii) to support the rapid exploration/evaluation of different architectural selections in term of thermal stress. Note that the architectural modifications applied with our methodology are transparent to the compilation flow (they do not affect existing tools), while they speedup the development of new products, since end-users (e.g. programmers) do not have to

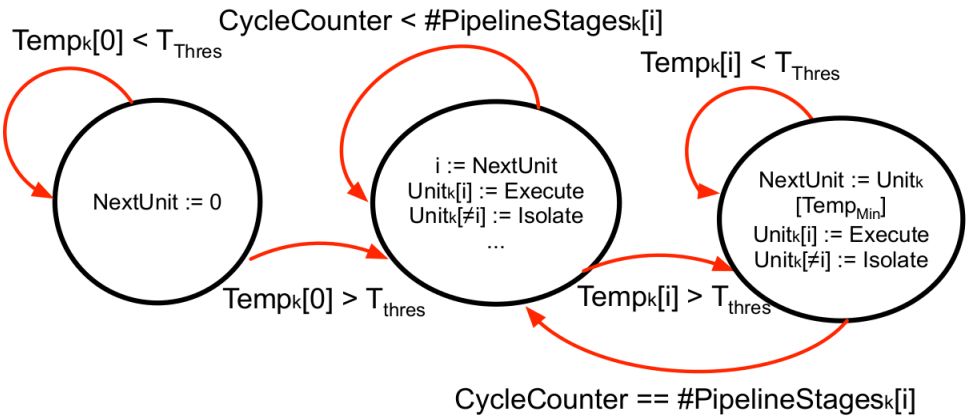


Figure 3.5 Employed thermal-aware runtime controller per replicated block.

consider thermal issues).

This methodology is shown graphically in Figure 5.20. The inputs to this methodology are the description of target architecture in VHDL/Verilog, the technology constraints regarding the selected CMOS technology, as well as the operating conditions, and the affordable area overhand (the higher area, the more replicas can be integrated into the design). Even though this methodology is applied at design time, the upcoming sections also evaluate the on-chip temperature variations during application execution.

3.1.5.1 Architecture Synthesis

The inputs to this replication-aware thermal management methodology are the architecture description (in VHDL/Verilog), as well as the selected CMOS technology. Initially, design is synthesized with Synopsys Design Compiler [23] and then we perform post synthesis simulation (with Cadence Incisive Simulator [24]) in order to extract some metrics about the design. Among others, these metrics include the area occupied by each component of the design, as well as its power consumption based on the switching activity of application executed onto the target architecture (the power consumption is retrieved with PrimeTime PX [25]). In addition to that, from the results of post synthesis simulation, it is feasible to have a first estimation regarding the timing of the design (maximum operation frequency) with the usage of Elmore delay model [26]. Note that both for power and delay study, we employ worst case test vectors as workload to architecture, in order to guarantee that the derived results, as well as the consequence thermal stress, will not be violated for any applications during runtime.

The output of synthesis task is appropriately encoded into an XML format in order to be manipulated by the introduced tools of our framework. The granularity of system's description in this XML format is tunable, since higher detail means a more accurate thermal analysis, but it imposes the maximum computational effort.

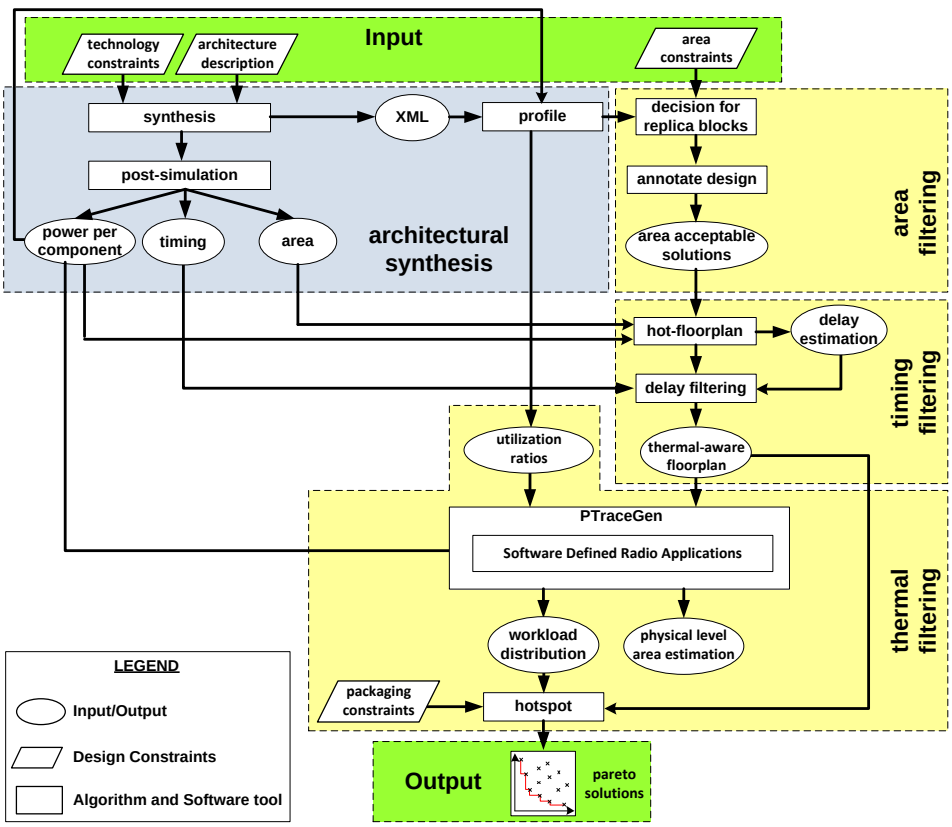


Figure 3.6 The proposed methodology for replication-aware thermal management.

On the other hand, a more coarse grain approach leads to lower computational effort but it also imposes a penalty in term of hotspot elimination. Even though our methodology is applicable at design time, and hence there is almost no performance degradation due to additional computational complexity, however, the increased number of functionalities inside a SoC usually makes the selection of a fine grain description of target system a non desirable approach.

The derived system's description is profiled in order to compute the power density of the functionalities described in XML file. For this purpose, input regarding power and area info, as it was already derived from post synthesis simulation, are employed.

3.1.5.2 Area Filtering

Based on this analysis, it is possible to make a decision regarding which of the architecture's hardware blocks have to be replicated. For this step, the power density for hardware blocks has to be measured. Note that the total number of replication blocks is limited by area constraints posed by designer.

Since we try to alleviate thermal stress mainly at hardware blocks with increased power densities, these are the blocks that should be replicated (as we have shown in Section 3). For this purpose, hardware blocks of the design are sorted in descending order based on their power density values. From our exhaustive exploration study, we found that only a few of the total blocks exhibit increased power densities. Hence, the architectures that contain all the possible combinations among replica blocks are evaluated. As a constraint to this procedure we assume the maximum area overhead, as compared to the one retrieved when no replica blocks are assumed.

At this procedure, an extra architectural parameter needs to be defined . More specifically, apart from the blocks that need to be replicated, we also need to clarify the maximum number for each of these blocks that can be replicated. Since more replicas means better thermal management, in the expense of imposing overheads in area and delay, careful study should be applied. In this study, we evaluate solutions that correspond to maximum number of replica blocks up to five. This selection was based on our conclusion that architectures consisted of more replicas do not lead to additional temperature reduction (due to saturation effect). However, constraints posed by architecture specifications might reduce this number.

After defining the type of replica blocks, as well as how many times they should be replicated, the next tool in our framework performs automatically this task by annotating appropriately the design's description. Apart from the insertion of new (replica) blocks to the design, during this task we have to pay effort to provide the appropriate connectivity through routing infrastructure, as well as to insert the thermal-aware runtime controllers in XML format. Moreover, during this annotation we keep the same connectivity among hardware blocks, while we have also to preserve that all the connections to (and from) replica blocks should be also replicated. This is an important differentiation of proposed solution, as compared to similar approaches found in relevant references [9] [7] [6] [8] [10], since we do not aim at altering the functionality of underline architecture.

The outcome from this step is all the candidate architectures instantiations that meet area constraints. Such a criterion can eliminate from design space solutions that lead to unacceptable overheads in device area due to excessive number of replica blocks. Also, by allowing a designer-defined overhead in this metric it is possible to explore and evaluate different architectural solutions. Regarding our exploration, we set this area overhead to 35%, since otherwise our methodology leads to excessive area penalties.

The employed criterion allows blocks with increased power densities to be replicated more times as compared to blocks with smaller values of power density. This occurs because the area occupied from these blocks is usually smaller, and hence more of them are fit into the given (affordable) percentage of area overhead. Since only one of the replica blocks is active at any time (based on approach discussed in Section 4), such an aggressive replication of blocks with increased power densities lead to (i) minimize maximum temperature hotspots, and (ii) spread more uniformly over the entire architecture.

3.1.5.3 Timing Filtering

Next, we proceed to the second criterion for evaluating the efficiency of derived architecture instantiations that affects the timing constraint. For this purpose, the solutions derived from area filtering are floor-planed with the usage of a thermal-aware floor-planner [20].

The optimization goal during this procedure is to reduce the thermal stress for each architecture in respect to the timing constraint. The alleviation of thermal stress is performed by spreading as much as possible the hardware blocks that contribute more to higher values of on-chip temperature (e.g. modules with increased power densities). Similarly, by minimizing the perimeter of bounding box that surrounds all the modules that are connected with a single bus, it is possible to improve the delay of this bus. Hence, blocks that are connected through bus(ses) have to be floor-planed in spatially close locations.

Regarding our methodology we allow all the blocks of the chip to be "soft" blocks, that is, their aspect ratio can change (in a controlled manner) in each annealing movement of $Hotfloorplan$ tool, but their area is fixed.

The derived solutions are then evaluated in term of delay degradation, as compared to delay estimation retrieved from post-synthesis simulation. For this scope, we use the Elmore delay model [26]. Since we are primarily interested to retrieve a thermal-aware solution, we already knew that a penalty in architecture's performance is affordable. For our study, the timing overhead is assumed to be 14% in order to avoid any mentionable delay overheads.

The output of this step is all those thermal-aware floor-planned solutions which their timing degradation meets the design specification (as it was derived after post synthesis simulation without considering yet any replica block).

3.1.5.4 Thermal Filtering

Finally, in the last task in the proposed methodology, the different architectural solutions are evaluated against thermal constraint. This task is automated with a new tool, named $PTraceGen$, that generates proper power traces onto the examined processor architecture regarding the statistical behavior of the targeted application domain. For this purpose, the outcome from this tool provides detailed information about architecture's units that describe i) the amount of time that a unit is active, ii) the amount of time that a unit is inactive, and iii) the switching activity of these units. The first two parameters (i.e. amount of active/inactive time) refer to the number of execution cycles for a timing window, whereas the third parameter corresponds to the number of transitions between operational states (active/inactive) during the architecture's execution phase.

Specifically, the statistical characterization is performed in a window-based manner (denotes the temporal granularity for performing thermal analysis) by computing the statistical mean values of the primitive operations executed by the processor, i.e. number and type of ALU instructions, memory accesses, cache hits/misses, communication load etc. For this purpose, power traces for all the replica units are generated. These traces describe the power activity for all the architecture's units (replicated or not) within a certain amount of time. Using these statistics the utilization ration per component is extracted in each examined window. The per component utilization ratios are correlated with accurate post-synthesis power measurements to generate the power traces. For the examined power traces, the thermal filtering tool extracts the Pareto frontiers under various design objectives (i.e., power density of the chip, delay, area, max temperature, max thermal gradient, etc).

In this research work, we study a uniform workload incorporating the key elements of base band processing domain, where these optimizations against to thermal stress can be applied. More specifically, the employed applications, obtained from [27] are summarized, as follows:

- *Adaptive differential pulse-code modulation*: ADPCM is a variant of differential pulse-code modulation (DPCM) that varies the size of the quantization step, to allow further reduction of the required bandwidth for a given signal-to-noise ratio.
- *Cyclic redundancy check*: CRC is an error-detecting code designed to detect accidental changes to raw computer data, and is commonly used in digital networks.
- *Fast Fourier transform*: FFT is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse.
- *GSM 06.10*: GSM 06.10 is a digital speech coding standard used in the GSM digital mobile phone system

These telecommunication applications exhibit increased demand for bandwidth requirements. Regarding our architecture, the components with higher power den-

sity values when such kind of applications is executed are the instruction unit, the cache and memory controller, the DSU (debug support unit) and the AMBA AHB/APB bus controllers.

The output of *PTraceGen* tool is a set of workload distributions based on employed applications, as well as accurate area estimation after floor-plan. More specifically, the workload distribution is achieved by appropriately handling the power info per block of the target architecture (component of the selected level of granularity), as they were already derived from post synthesis simulation. This approach guarantees that hardware blocks (replicated or not) dissipate power consumption only the time periods denoted by the application's functionality.

These time periods are retrieved by incorporating info from application's simulation. The employed utilization ratios are averaged over hardware components of LEON3 system in order to determine the active/idle time slots accurately. The output from utilization is fed as input to *PTraceGen* tool in order to redistribute the activity of each component to a timing trace of several time slots. This is achieved by setting each of these components either as active or idle for every time slot.

Furthermore, the area derived in this stage may be different from the one computed during area filtering, due to additional free space inserted to design after floor-plan (which is not occupied by any hardware block) in order to model the white space between hardware components.

The workload distribution, in conjunction to the packaging constraints, are fed to the *hotspot* tool [19] to compute thermal profile of target architecture. For increased accuracy, we used the steady state temperature of each hardware block as the initial temperature values. The default characteristics of *hotspot* version 5 were used for package, whereas the analytic model parameters are summarized in Table 1.

Table 3.1 Thermal characteristics of the employed processors

Parameters	Model value
Sampling interval	20 ms
Die Thickness	0.15 mm
Core Area (no replication)	0.426213 mm^2
Cache area (L1+Local I+D)	0.370561 mm^2
Convection Resistance	0.1 K/W
Convection Capacitance	140.4 J/K

Based on the derived thermal profile, it is possible to evaluate the architecture instantiation in terms of different criteria tightly firmed to on-chip temperature. For the scope of this research work, all the solutions that do not meet the selected thermal constraints (maximum temperature and the temperature gradient) are eliminated from exploration space, whereas typical packaging for embedded processors is assumed [28]. Thus, the output of the proposed methodology contains only the instantiations that correspond to architectural solutions that meet all

the three constraints, namely area, timing and thermal.

3.1.6. Experimental Results

This section provides a number of experimental results derived from the proposed exploration methodology that prove the efficiency of our approach in term of reducing the consequences posed by thermal stress. For this purpose, a LEON3-based design is employed [18], and the introduced methodology is applied to identify the blocks that lead to higher temperature reduction, and hence they should be selectively replicated, as it was already discussed in Section 5.

We have to mention that functionality of the underlined LEON3 processor is not affected by the additional (replica) blocks.

The majority of aging phenomena are tightly firmed to on-chip temperature values. Hence, higher maximum temperatures lead among others to devices having increased failure rates. For this reason, the first criterion employed in our methodology for selecting the architecture of target platform involves to study how temperature values are spatially distributed over the target device.

3.1.6.1 Impact of Selective Replication on Temperature

Figure 3.7 depicts the variation of maximum temperature (in Kelvin) when different instantiations of the target architecture are considered. The axes of this figure denote the normalized operation frequency and the architecture's area, as compared to the corresponding maximum values found among all the candidate architectures, whereas the vertical axis gives the on-chip temperature values. Based on Figure 3.7, it is evident that temperature values vary considerable among architectures with different selection of replica blocks. More specifically, regarding the LEON3 architecture, temperature variations from 354 to 382 Kelvin's, were reported.

The following conclusions can be derived from this figure. More specifically, as we increase the area of target architecture, the temperature values are reduced (almost monotonically). However, this temperature reduction in not constant since just replication of blocks does not guarantee alleviation of thermal stress (as we have already depicted in Figure 5.19(c)). Hence, apart from the number of replica blocks, their properties (e.g., power density, area, power consumption, etc) are also crucial for designing an efficient architecture.

Apart from area, the maximum operation frequency also affects the on-chip temperature values. Based on Figure 3.7, the alternative architectures lead to performance variations up to 14%, which mainly occur due to: (i) additional replica blocks inserted into the design, (ii) the consequent different floor-plans, and (iii) the increased wire-length for connecting these blocks.

Another interesting conclusion might be derived from Figure 3.7. Even thought higher operation frequencies usually result in higher temperatures, this seems to be alleviated when architectures with increased area are assumed. Regarding the LEON3 architecture, the additional area is dominated by blocks with low power density values (as we have already mentioned in Section 2). Hence, by introducing more replica blocks it is possible to improve thermal spreading. However, since

Figure 3.7 Temperature variation for different instantiations of target architectural.

temperature values seem to be more agnostic about the maximum operation frequency, as compared to area overhead, we cannot make safely any conclusion about this.

Based on this figure it is possible to select an architecture that better trades-off design constrains. A balanced design solution under the aforementioned criteria (area, maximum temperature, and delay) is the one that replicates four AHB controllers, three integer units and two cache controllers. This architectural instantiation, mentioned as "Selected architecture" in upcoming figures, belongs to solutions marked as valid during the area, timing and thermal filtering. The selection of this architecture for further evaluation is performed since it belongs to the Pareto front for reliability improvement, as it is discussed in more details in subsection 3.1.6.2.

More specifically, the area and delay overheads for our selected replication-aware LEON3 design are 15% and 7%, respectively, as compared to initial implementation (without considering any replica blocks). Even though these penalties

are not negligible for ASIC designs, we have to mention that they comes with considerable gains in term of maximum temperature value (about 17 Kelvin or 8%), which in turn leads to higher reliability improvements. Furthermore, the proposed methodology for selectively replication of blocks with increased power densities can also be applied to multi-core architectures, where the performance degradation is more affordable.

Apart from the selected architecture, any other architecture instantiation can be chosen without affecting the efficiency of our proposed methodology, if different constraints are applied. Note that our framework mainly indents to enable the thermal improvement of architectures through inserting replica blocks.

In order to show the importance of proper identification for hardware blocks that have to be replicated, Figure 5.12 plots the temperature variation (in Kelvin) versus the power density (W/cm^2) for each instantiation of the selected architecture with replica blocks. We choose to evaluate such a criterion, because power density for existing and upcoming devices becomes a major issue for architects. Researchers have already identified this problem, whereas based on projections it is expected that power density regarding 14nm nodes will be higher than 100 W/cm^2 [29].

Figure 3.8 Results about power density versus maximum temperature.

In order to plot this graph, architectures are grouped into three categories based on their power densities, as following:

- those with area smaller than the 33% of maximum area among all the solu-

tions

- those with area ranging between 33% and 66% of the maximum area among all the solutions
- those with area higher than 66% of the maximum area among all the solutions

Note that this classification with respect to area occupied by different architectures is also applied in upcoming figures (Figures 3.9 and 3.10), since it can provide qualitative comparisons about the importance, as well as the efficiency, of the proposed methodology in term of alleviating thermal stress.

Since different architectures consist of different replicated blocks, their power densities also vary. As we can conclude from Figure 3.9, there is not a straightforward correlation among the occupied area, the overall power density and maximum temperature. More specifically, regarding architectures shown in Figure 3.9 that correspond to increased area (more than 66% of the maximum area), they seem to exhibit the maximal power densities but the lower temperature values. On the other hand, the architectures with smaller area (less than 66% of the maximum area) exhibit reduced power densities and increased temperatures values. This rather strange result is justified by the fact that the type of the replicated blocks is not considered in the previous analysis. Thus, we can conclude that the type of the replication block has a great impact on the thermal behavior of the silicon.

The last conclusion is very interesting since it shows that even architectures with increased power densities can achieve considerable on-chip temperature reduction. This point verifies the argument discussed in subsection 3.1.5, that non optimal replication of blocks leads to similar (or higher) thermal profiles, as compared to initial architecture implementation (without replica blocks).

In Figure 3.9, we have also marked the solutions which correspond to previously mentioned selected architecture, as well as to the original (without replica blocks) LEON3. Based on this, both those two solutions exhibit comparable power densities, but the selected one achieves to reduce maximum temperature value about 17 Kelvin, or 8%.

In order to study the correlation between areas occupied by target architectures and the maximum temperature values, Figure 3.9 plots the corresponding diagram of these parameters. For sake of completeness, we cluster alternative architectures into three groups based on their area (similar to previous figure), while for demonstration purposes we also compute the temperature gradient for each of these clusters.

Based on Figure 3.9, the maximum temperature gradient occurs for architectures with smaller areas, whereas for architectures with area more than 66% of the maximum area is almost constant. If we take into consideration also the smaller delay overhead posed by devices consisted of fewer replica blocks (as it was already shown in Figure 3.7), an additional filtering of derived architectural instantiations can be performed. More specifically, without affecting the generality of the proposed methodology, solutions that correspond to area overheads higher than 66% of the maximum area are assumed not to be desirable (due to additional delay), and hence they are eliminated from exploration space.

Figure 3.9 Results about area versus maximum temperature.

Furthermore, the temperature values for architectures with few replicas (less than 33% area overhead) is about 3× higher, as compared to the remaining solutions. Since our methodology tries to alleviate the hotspots, such high temperature variations usually result in increased cost for packaging and cooling, and hence they are not desirable. These solutions can also be eliminated from exploration space.

This conclusion is very important in order to find the amount of blocks that have to be replicated. On other words, based on Figure 3.9, it is clear that only a few replicas of the blocks with increased power densities should be incorporated, in order to achieve the desired balance between temperature reduction and the consequences area and delay overheads.

3.1.6.2 Impact of Temperature on Reliability

Reliability is defined as the probability that a device will perform its required function under stated conditions for a specific period of time. Predicting with some degree of confidence, strongly depends on defining a number of parameters.

Accelerated life testing employs a variety of high stress test methods that shorten the life of a product, or quicken the degradation of the products performance. The goal of such testing is to efficiently obtain performance data that, when properly

analyzed, provide reasonable estimates of the products life or performance under normal conditions. This induces early failures that would sometimes manifest themselves in the early years of a products life, and also allows issues related to design tolerances to be discovered before volume manufacturing. Both the type of stressor and the time under test are used to determine the normal lifetime. Regarding SoC designs, usually the majority of these aging degradation mechanisms are tightly firmed to on-chip temperature values.

The effect of these stressors can be mathematically determined. Next we model aging acceleration due to thermal stress with the usage of Arrhenius equation (Equation 1). More specifically, this equation models how the age of a product is increased when it operates under higher temperature values, as compared to its normal operating temperature. Figure 3.10 plots how this parameter varies for different architectural instantiations discussed in this research work. More specifically, the horizontal axis in this figure gives the average temperature, whereas the vertical axis shows how the on-chip temperature affects aging degradation.

$$A_f \propto \exp\left[\left(\frac{E_a}{k}\right) \times \left(\frac{1}{T_u} - \frac{1}{T_t}\right)\right] \tag{3.1}$$

where A_f is the acceleration factor, E_a is the activation energy in electron-volts (its value is 0.5 eV for silicon defects), k is the Boltzmann's constant, whereas T_u and T_t are the reference (Kelvin) and the operation temperature during testing.

Based on the values depicted in this figure, we can conclude that the selected architectural instantiation achieves almost the minimum value for A_f parameter among all the candidate solutions. Furthermore, the conventional approach for designing LEON3 architecture exhibits about 14% higher value for this parameter. This mainly occurs due to additional thermal stress introduced by architecture's components with increased power density. However, we have to mention that even this no-replica aware architecture is not the one with the maximum aging degradation, since there exist solutions that correspond to A_f value up to 20% of the selected solution (this occurs since non all the possible combinations of blocks lead to alleviate the thermal stress).

Apart from Arrhenius equation, we also evaluate the different architectures derived during our exploration, under the Time-Depended Dielectric Breakdown (TDDB) [30]. Since oxide breakdown has already been of serious reliability concern in the semiconductor industry because of the continuous trek towards smaller device sizes, such kind of aging phenomenon should be carefully studied. Defects occurred due to TDDB are primarily caused due to the trapping of charges in the oxide that create an electric field, followed by charge flow through the oxide, resulting in a breakdown after sometime. The MTTF due to TDDB phenomenon is described by Equation 2.

$$MTTF = A_0 \times exp(-\lambda \times E_{ox}) \times exp\left(\frac{E_a}{k \times T}\right) \tag{3.2}$$

where λ is a field acceleration parameter, which is temperature dependent.

Figure 3.10 Evaluation in term of A_f parameter for architectures with different average temperatures.

Figure 3.11 plots the variation of Mean Time to Failure parameter for the different architectures, as they derived from our exploration framework. For demonstration purposes, the values at vertical axis were plotted in normalized manner over the corresponding MTTF for the initial architecture instantiation of LEON3 processor (without replica blocks).

Based on Figure 3.11, a number of conclusions might be derived. Among others, as we increase the maximum on-chip temperature values, the MTTF also increases. This is explained due to the tight correlation between aging phenomena and temperature values. Additionally, the selected architecture achieves to improve the MTTF parameter, as compared to initial instantiation of LEON3 processor (without considering any replica blocks), about 14%. We have to notice that in case we increase the architecture's area, through inserting more replica blocks, the MTTF parameter can be further increased. This is due to the fact that architectures with more replica blocks usually occupy more area, which in turn improve thermal spreading. However, as we have already mentioned, such an improvement in MTTF parameter (from 0.86 to 0.81) leads to architectures with unacceptable area penalties (they have eliminated with area filtering as depicted in Figure 3.9).

Figure 3.11 Evaluation of different architectures under TDDB.

3.1.6.3 On designing chip multiprocessors for SDR

This subsection describes the results retrieved of applying the proposed methodology for designing chip multiprocessor architectures. For demonstration purposes, the target multiprocessor consists of four instances of LEON3 (this number is parametric to our framework and can be appropriately tuned based on designer's requirements), while the replica modules among LEON3 processors for a given instantiation of multiprocessor, are the same. As a reference to this study we employ a multiprocessor architecture consisted of the LEON3 which was marked as "selected" in the previous figures. For the following figures, this solution is denoted as "reference solution".

Figure 3.12 gives the thermal profiles for this architecture, as it was retrieved from Hotspot tool. Based on this figure we can conclude that a number of modules per LEON3 exhibit increased temperature values. Note that these hotspots differ from those reported previously at Figure 5.19, due to (i) the thermal diffusion effect, and (ii) the different floorplans for each LEON3 processor.

Next, we will quantify the efficiency of the above solution when this is used in multiple instantiations of a multiprocessor architecture.

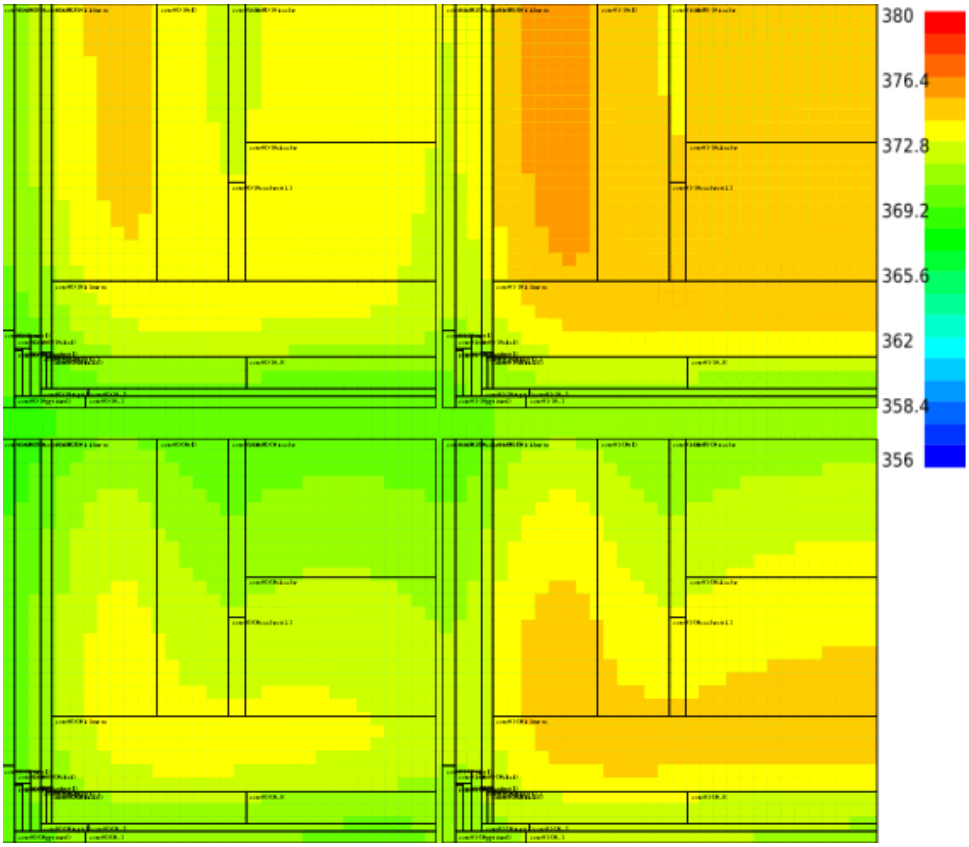


Figure 3.12 Thermal profile for 2×2 CMP LEON3-based architecture.

Figure 3.13 shows the variation of power density as we select solutions with higher area overheads. Based on this figure we can conclude that power density seems that it does not depend on silicon area. We have to mention that during this analysis, the additional area mostly occurs due to the whitespace between hardware components, rather than the area of these replica components.

In contrast to our conclusion about power density, area has a great impact on the maximum on-chip temperature values. This is also depicted at Figure 3.14.

More specifically, based on this figure we can conclude that a controllable area overhead (e.g. 20% increase as compared to the multiprocessor solution composed of LEON3 components selected previously) leads to the reduction of the maximum temperature by almost 0.85× of the previous corresponding value.

Notice that architecture instantiations of the same area exhibit temperature variations due to the different components that are replicated.

Figure 3.13 Normalized power density versus area overhead for multiprocessor LEON3.

3.1.7. Conclusions

In this research work, we propose the adoption of selective replication techniques in order to optimize the thermal behavior of the synthesized micro-processor systems targeting at an SDR system. We developed an automated exploration methodology that permits the thermal aware evaluation of various micro-architectural instantiations.

We show that by using selective replication, we can deliver optimized architectural solutions with minimum thermal stress, affordable delay and user-constrained area overheads. Experimental results have shown a significant reduction of the maximum operating frequency, by 8%, which in turn leads to improvement at maximum on-chip temperature values. Moreover, they have shown that our approach improves by 14% the aging phenomena. These two results show that our approach compares favorably to the conventional design techniques for SoC-based SDR architectures.

3.2. Heterogenous Network-on-Chip Multimedia Architectures

3.2.1. Introduction

Historically, computation has been expensive and communication cheap. However, with the advance of technology scaling, this changed. More specifically, last years

Figure 3.14 Normalized maximum temperature versus area overhead for multiprocessor LEON3.

computation is becoming ever cheaper, while communication encounters fundamental physical limitations such as time-of-flight of electrical signals, power use in driving long wires/cables, etc. In comparison with off-chip, on-chip communication is significantly cheaper; thus the shift to single-chip systems has relaxed many communication problems. Although the shared bus is a simple interface since it is built on well-understood concepts and it is easy to model, on-chip wires do not scale in the same manner as transistors do, and consequently the cost gap between computation and communication is widening. This problem becomes far more savage especially in highly interconnected (multi-core) systems. In contrast, Network-on-Chip (NoC) technology is a relatively new approach that enables not only more efficient interconnects but also more effective design and verification processes for modern MPSoCs [31].

Due to the importance of this interconnection paradigm, researchers spent effort on proposing methodologies and tools for customizing both the router, as well as the entire NoC architecture [32] [33] [34] [35] [36]. Moreover, recently, there are also attempts to employ advanced process technologies for improving further the performance of NoC platforms. Among others, three-dimensional (3-D) integration, which enables stacking of multiple die on the vertical axis and interconnecting them using very fine-pitch Through-Silicon Vias (TSVs), introduces locality along the z-axis enabling on average shorter interconnections between system components

[37].

The existing approach on designing 3-D NoCs imposes networks consisted solely of 3-D routers. Assuming a 3-D mesh topology, these routers, apart from the direct connection to their four neighbors assigned to the same layer, also provide connectivity to vertically aligned routers (upper and lower layers). Even though such a selection leads to "uniform" underline hardware, however, rarely can be though as an efficient solution, since it does not take into account application's requirements for data transfer. Specifically, as a NoC is usually an application-oriented communication infrastructure, careful analysis should be performed for deriving an optimum architecture. Towards this goal, throughout this research work we propose the usage of heterogeneous 3-D NoCs, which better tackles the communication constraints posed by the target applications. By the term *heterogeneous* we refer to architectures that combine a mixture of 2-D and 3-D routers into a single NoC, whereas the spatial assignment of these routers over the target architecture is defined based on application's requirements.

Even though there are some prior works on designing heterogeneous 3-D NoCs [37] [32] [35] [36], all of them are based on abstract models and consequently no useful conclusions about their efficiency might be derived. Specifically, both the NoC's performance, as well as its power/energy dissipation is usually retrieved by counting the number of hops (connections between adjacent routers) that a packet has to traverse in order to be delivered from source to destination nodes, ignoring about parameters related to physical implementation. Also, these approaches do not take into consideration constraints posed by the selected 3-D technology, leading to unacceptable architectural solutions (e.g. they impose an excessive amount of TSVs). On the other hand, the experimentation throughout this work is performed with the usage of a framework based on commercial tools (C-to-Silicon and SoC Encounter) provided by Cadence. Based on our analysis with a number of DSP applications, we found that the introduced heterogeneous 3-D NoC outperforms conventional (i.e. uniform) 3-D NoCs, as it achieves on average 25% higher maximum operation frequency and 39% lower power consumption.

The rest of this research work is organized as follows: subsection 3.2.2 describes the concept of heterogeneous 3-D NoC architecture. The proposed methodology for designing such an architecture, as well as the tool framework for performing rapid evaluation of these NoCs, are discussed in subsections 3.2.3 and 3.2.4, respectively. subsection 3.2.5 presents a number of qualitative and quantitative results that prove the efficiency of introduced solution. Finally, conclusions are summarized in subsection 3.2.6.

3.2.2. Architecture of the Proposed Interconnection Scheme

This section introduces the architectural organization of the proposed communication scheme consisted of a mixture of 2-D and 3-D routers. More specifically, a 2-D router can be used where an incoming routing track is connected to wires on the same layer (F_s = 3). The router's flexibility, denoted as F_S, gives the number of directions to which each incoming wire can be connected. Alternatively, since a 3-D router also supports connections to the third dimension (upper and lower layer,

the value of router's flexibility is equals to five (F_S = 5).

In order to depict the differences between these two baseline routers, we assume an application's task graph (depicted in Figure 3.15) mapped onto a 3-D chip consisted of five layers. The arrows in this figure denote communication links across either the horizontal or the vertical direction between adjacent routers. Based on this example, not all of these routers exhibit similar requirements for data transfer. However, since the design of piece-homogeneous architecture is much easier and more cost-effective, as compared to a irregular platform, it is possible to cluster routers into two main groups: (i) those that have to support connectivity across the vertical directions (upper and lower layers) and (ii) the rest that provide packet routing exclusively inside the same layer. Hence, by replacing the routers that belong to the second group with their equivalent 2-D implementations, as it is depicted in Figure 3.16(a), we expect a more efficient hardware implementation.

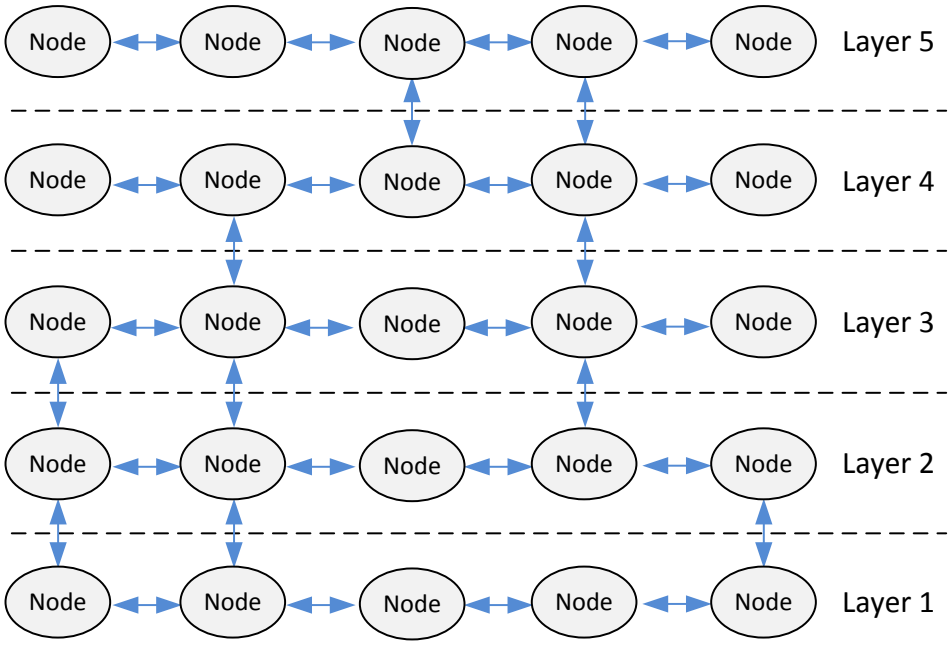


Figure 3.15 Example for an application's communication graph.

One more optimization is feasible, which affects the customization of 2-D and 3-D routers depending on their spatial assignment over each layer. Specifically, regarding the 3-D routers assigned to bottom (Layer 1) and top (Layer 5) layers, there is no demand for providing connectivity to lower and higher layers, respectively, as it is depicted in Figure 3.16(b). Similarly, for the routers assigned to the periphery of each layer, they can be designed with fewer ports, as they have limited number of neighbors. Note that the architectural solution discussed throughout this paper provides the maximum possible customization (similar to Figure 3.16(b)) in order to derive an application-specific heterogeneous 3-D NoC architecture.

Depending on this analysis, apart from the 2-D router (mentioned as *Router*$_0$), we employ three more flavors of 3-D routers:

- *3-D Router*$_1$: It supports connectivity from lower to higher layer. Since an incoming packet can be routed to four different directions (note that the input and output ports could not be the same in order to avoid packet deadlocks), the router's flexibility is F_s = 4.
- *3-D Router*$_2$: It supports connectivity for links that realize connections from higher to lower layers. Similar to previous case, this router has F_s = 4.
- *3-D Router*$_3$: It supports connectivity from lower to higher layer and vice versa (F_s = 5).

3.2.2.1 Designing 2-D and 3-D Routers

The basic component of the proposed architecture is the NoC router. This router was designed in SystemC in order to be highly configurable at compile time. Among others, parameters such as the number of ports, the phit size (word-length), the flit size, the buffer sizes for the input and the stalled packets are configurable, through a package file. Additionally, since our framework aims to support rapid evaluation of heterogeneous 3-D NoC systems, the proposed architectural solution consisted of a mixture of 2-D and 3-D routers (described in SystemC) is fully synthesizable. This is also a key differentiation against relevant approaches targeting to automate the physical design of NoCs.

Figure 3.17 gives the block diagram for the employed 3-D router. The input packets from the interface, as well as the corresponding packets from the attached node, are stored to a buffer of stalled packets. As only one packet is routed in a clock cycle per output port, the buffer of stalled packets (SPB) contains those packets which could not be routed to the router's output ports. Even though a 2-D router consists of fewer ports (there are no ports for connectivity to upper/lower layers), the architectural organization of a 2-D router is similar to the one discussed previously. The only differences between these two architectural instantiations affect the routing mechanism (since there is no connectivity to upper/lower layers), as well as the size of SPB buffer. Specifically, as the 2-D router has fewer output ports, it is more likely for a packet to be stalled in this router, compared to the corresponding 3-D implementation. Hence, the size of SPB buffer for the 2-D router is 30% bigger compared to the corresponding size of a 3-D router.

In addition to that, the SystemC model of our router incorporates a flexible Inter-Router Interface (IRI) to prevent manual architectural modifications in case of a non-valid routing direction (e.g. there is no node attached to a local port, or a router is spatially assigned to the periphery of a layer). The implementation of IRI is based on a set of Read/Write C++ methods, one for each direction, which define the connectivity of router's ports to the neighbor routers. From physical point of view, the IRI provides the appropriate pins for the communication with the neighbor routers, marked as *X_LEFT*, *X_RIGHT*, *Y_UP*, *Y_DOWN*, as well as the

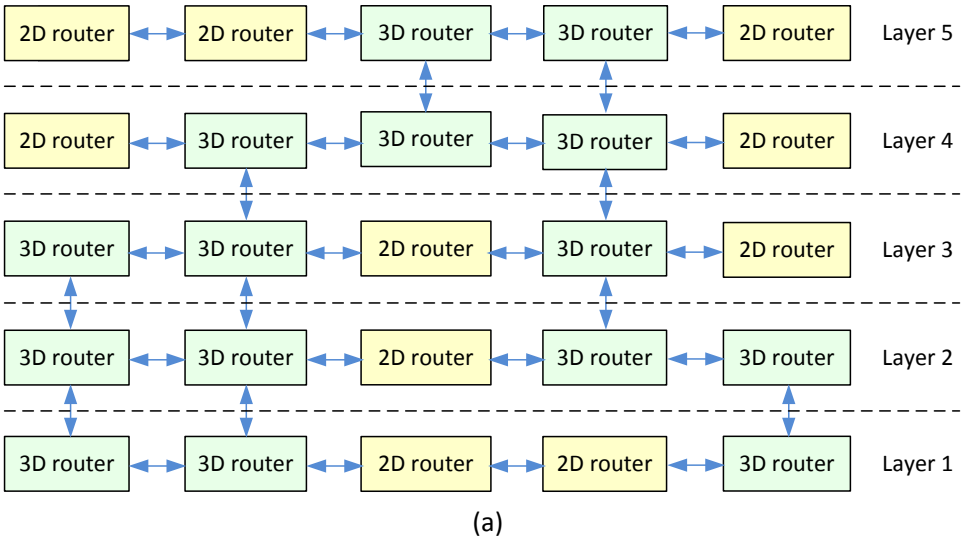


Figure 3.16 Alternative 3-D NoCs for the example discussed in Figure 3.15.

Z_ABOVE and *Z_BELOW* (in case of a 3-D router). In case a router is not attached to a neighbor router at a given direction, then the respective method invalidates the corresponding Read/Write operations.

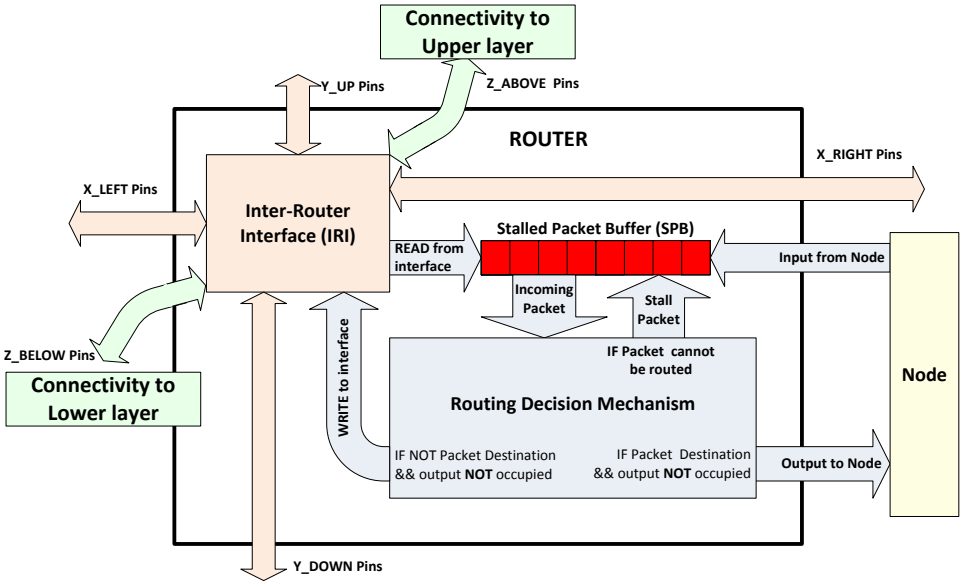


Figure 3.17 Architectural template for a 3-D router.

The selection of the output direction for an incoming packet is defined from the Routing Decision Mechanism based on a modified ZXY algorithm. According to this algorithm, priority is given to the stalled packets that are stored into the router's local SPB buffer. Since the size of this buffer is limited, the routing algorithm incorporates a mechanism for preventing the overflow of SPB buffer. More specifically, in case the packet could not be routed towards its direction with the minimum Manhattan distance and the SPB is full, then this packet is routed to the first unoccupied direction (port) in order to avoid data loss. On the other hand (i.e. when there are available unoccupied output ports), then the z-axis has higher priority compared to the packet routing inside the same layer. Additionally, the employed routing algorithm supports the avoidance of live-locks and dead-locks. In more detail, the avoidance of deadlocks is performed by incorporating the Turn Model [38]. Regarding the livelocks, they may occur in networks with non-minimal routing algorithms, i.e. where packets may follow paths that do not always lead them closer to the destination. For the scopes of this paper, the employed routers avoid livelocks by prioritizing traffic based on hop-counters. For each node a packet traverses, a hop counter is incremented. If several packets are requesting a channel, the one with the largest hop-counter value is granted access. This way, packets that have long circled the network will receive higher priority and eventually reach the destination. Finally, in order to declare the spatial assignment of each router across the grid of 3-D NoC, we also define its co-ordinates as a tuple $< X,Y,Z >$. The information

about the co-ordinates of destination router is also found in the header of each packet, as it is depicted in Figure 3.18. We have to mention that more flexible routing algorithms than the employed modified ZXY algorithm can also be supported, if their functionality is appropriately included into the router's SystemC description.

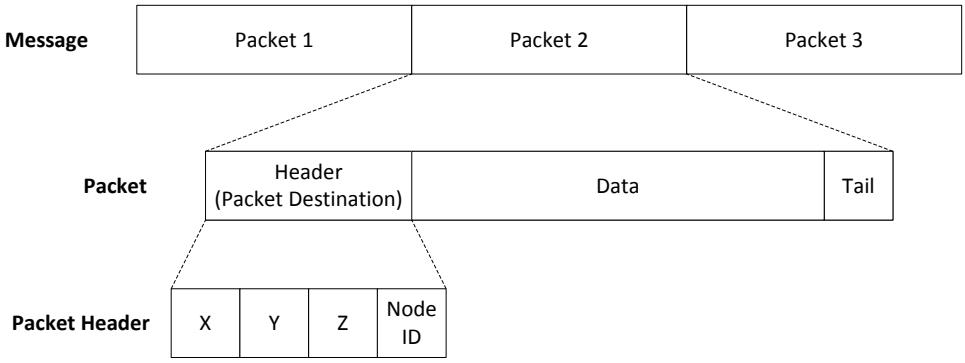


Figure 3.18 Structure of packets for our proposed NoC architecture.

3.2.3. Proposed Methodology

This section describes the proposed methodology for performing rapid evaluation of heterogeneous 3-D NoCs. The steps of this methodology are depicted schematically in Figure 3.19. Initially, a two-step profiling approach is applied to the application's high-level description (e.g. C/C++) for determining the communication bandwidth among functionalities mapped onto different nodes. More specifically, the first step deals with algorithmic analysis of the target application (this task is applied manually) in order to determine the amount of data exchanged between the application's kernels per function call, whereas the second step is based on software tools (for our case we employ the Callgrind tool from Valgrind suite [39]). The scope of this analysis is to find out how many times each function is called. Then, it is possible to calculate with acceptable accuracy the total amount of data sent/received between application's functionalities.

The outcome of this analysis in conjunction to the system specifications (e.g. desired throughput, maximum affordable power/energy dissipation, etc) are fed as inputs to the topology exploration tool. At this step different heterogeneous 3-D NoCs are evaluated in order to derive those topologies that belong to the Pareto-optimal curve. Additionally, as the intra-router communication is far more efficient as compared to the corresponding inter-router links, our framework supports the clustering of multiple nodes into a single router.

The exploration framework takes into account a number of technology-oriented issues, such as the lack of plethora of TSVs (an array of TSVs occupies a significant part of useful silicon area [40]) and the maximum number of layers for the target 3-D platform. By enabling the exploration tool to be aware of these constraints, it is possible to retrieve only the technologically viable solutions. The topology ex-

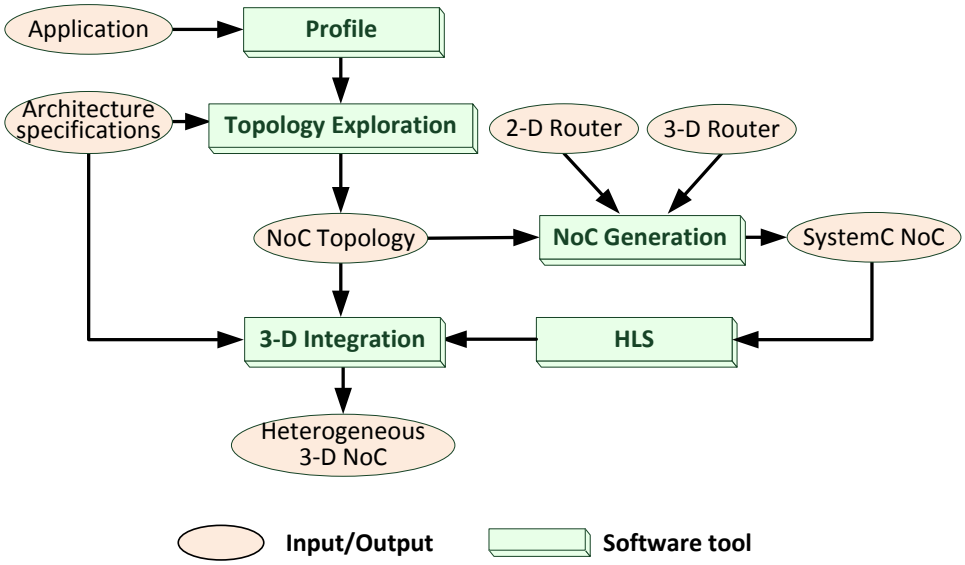


Figure 3.19 Proposed methodology for evaluating 3-D NoCs.

ploration procedure is automated with a software tool initially proposed in [41]. However, for the scope of this work, we have extensively modified the algorithm in order to take into account the additional connectivity constraints posed by incorporating heterogeneous 3-D NoCs (instead of uniform 3-D NoCs supported by the original version). The derived topology defines the spatial location of each router over the NoC's grid, as well as the type (either 2-D, 3-D with connectivity only to upper layer, 3-D with connectivity only to lower layer, or fully 3-D) for each router. By appropriately combining and interconnecting a number of 2-D and 3-D routers, our framework automatically derives a synthezable SystemC model, which describes the heterogeneous 3-D NoC.

After deriving the description of the NoC's architecture, we proceed with the physical implementation onto the selected 3-D technology. First of all, the SystemC is translated to a synthesizable Register Transfer Level (RTL) description with the usage of a High-Level Synthesis (HLS) tool. Regarding this research work, the HLS task is automated with C-to-Silicon compiler [42]. The rest steps of the introduced methodology deal with the architecture's synthesis, as well as the quantification of numerous performance metrics for the target 3-D system implementation. Due to the importance of this step, the next section provides additional details about how we quantify a 3-D design with existing 2-D Cadence tools. This feature is one of the major contributions discussed throughout this paper, since the usage of commercial tools guarantees that the derived results are much more accurate as compared to similar approaches, which are based usually on simplified models.

3.2.4. 3-D Integration

This section describes the introduced framework for performing rapid evaluation of 3-D NoCs. During this step all the platform-dependent decisions are made. Among others, these decisions include the way the system's architecture is partitioned, the IP block to layer assignment, the selection of interlayer interconnection technology, etc. By distinguishing these platform-dependent decisions from the pure physical prototyping step, we can support the design of 3-D stacks comprising heterogeneous layers. Additionally, the introduced 3-D integration task consists of modular steps in order to enable interaction with tools from similar and/or complementary flows. More specifically, the steps of this framework are summarized as follows:

- *Pre-processing Step*: Verifies the functional integrity for the design and extracts its XML description.
- *3-D Stack Generation*: Generates the 3-D stack and determines the communication (connectivity) among layers.
- *3-D System Prototyping*: Performs the physical implementation of 3-D NoC and evaluates the derived solution.

3.2.4.1 Pre-processing Step

The first step in our methodology is depicted schematically in Figure 3.20. Initially, the architecture's SystemC description is simulated under various parameters and constraints (e.g. clock period, on-chip memories organization) in order to verify the system's functionality. Then, we determine the desired hierarchy for the target 3-D architecture. Different levels of hierarchy are possible to be handled by our framework, each of which has advantages and disadvantages. For instance, a block-based system's description leads to a coarse-grain solution, whereas a gate-level netlist comes with a finer system implementation. In other words, the fine-grain approach imposes the highest performance enhancement, but it also introduces the maximum computational complexity for performing architecture-level exploration. Regarding the 3-D stack discussed throughout this research work we maintain the system's hierarchy between routers, while each of them is flattened in order to maximize timing and power savings. These gains are feasible since the employed selection takes into consideration the regularity imposed by the underlined mesh topology.

After defining the SoC's hierarchy, the SystemC description is synthesized with *Design Compiler*. As long as the design constraints (e.g. timing slacks, DRC's, etc) are met, the output of synthesis is translated to an equivalent XML description. This task is software supported by our new public available tool, named *Net2XML*.

3.2.4.2 3-D Stack Generation

The XML description derived from the previous step represents the SoC's netlist after technology synthesis. This description is fed as input to the second step of our proposed framework depicted schematically in Figure 3.21.

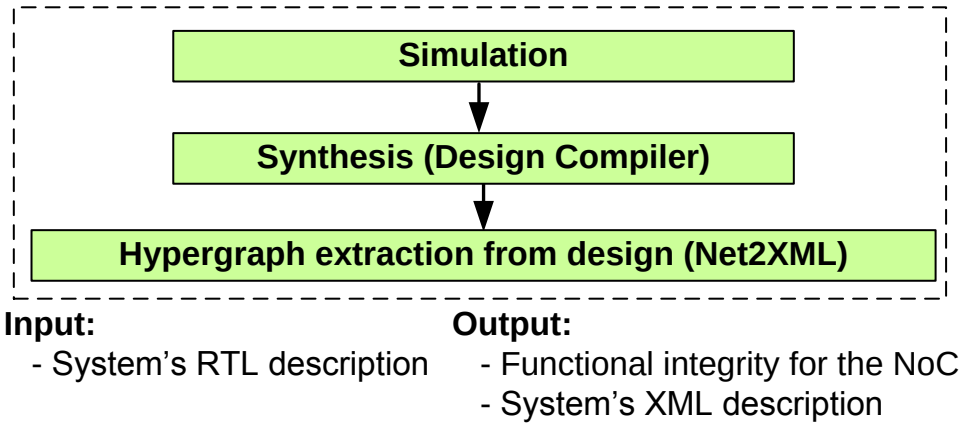


Figure 3.20 Tasks for the pre-processing step.

Initially, application is partitioned into a number of subsets. The goal at this level affects the minimization of connections between partitions, while respecting some constraints, like keeping DRAM and logic on different partitions. Previous studies showed that the partitioning algorithm exhibits increased flexibility whenever the number of subsets is higher as compared to the corresponding number of device layers [43] [44]. The derived subsets are then assigned to the layers of the target 3-D architecture. During this task, both fabrication and cost parameters are taken into account. More specifically, for a given layer, only technology-combatable components can be assigned, while each of the layers has to exhibit sufficient area utilization. Finally, we build a prototype of the stack by deciding on the order of the die in the stack (which goes to the bottom, which to the top, etc) and the choice of 3-D interconnection option. At this step, optimization objectives like total number of inter-die connections, maximum operation frequency, as well as the total system's power consumption are taken into account. Once this is complete, the performance metrics of the stack can be estimated based on high-level models [45] [46] [47] [48].

Another important aspect of the 3-D system prototyping step is the capability of going back and changing already taken decisions. Flexibility is very important, because this methodology is meant to be used for fast search-space exploration. In case the resulting 3-D chip does not meet the system specifications, there is a feedback loop back to the partitioning step to allow designers to modify some of the decisions already made, like using different router, or different block to die assignment options.

The tasks of systems partitioning, partitioning to layer assignment and layer ordering are software-supported by our publicly available *TABU* algorithm. An initial version of this tool was presented in [43], but for the scope of this research work, the algorithm has to be massively extended. More specifically, initially the tool was developed for supporting exclusively designs mapped onto 3-D FPGAs, whereas the current version can also handle 3-D ASIC platforms.

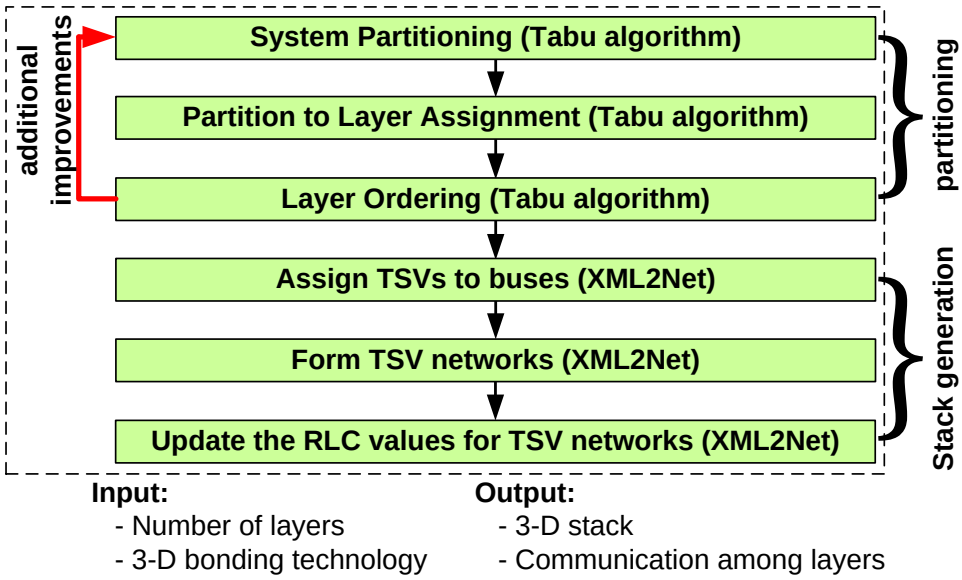


Figure 3.21 Tasks for 3-D stack generation.

Furthermore, rather than similar approaches, which mainly perform a min-cut partitioning [49], our solution provides additional flexibility. Towards this direction, $TABU$ algorithm is aware of a number of physical constraints coming from the fabrication process. For instance, stacks cannot have arbitrary shapes; cubes and pyramids are both manufacturable but other arbitrary shapes may create mechanical instability in the 3-D stack. Also, the selected bonding technology (e.g. TSV, Face-to-Face, etc), as well as the minimum pitch between adjacent TSVs, are taken into account during the partitioning procedure. The tool makes sure that the results respect these basic constraints, by applying high-level constraints on the geometrical properties of the stacks. Regarding the evaluation of the derived solutions, we employ well-established models for estimating wire-length [45], delay [46] and power consumption [47].

The output partitioning procedure provides information about the architecture's functionality assigned to each layer, as well as the required connectivity among layers. This information is appropriately handled by our new tool, named *XML2Net* for assigning a TSV array to each bus that connects architecture's components in different layers. Note that whenever a bus needs to be routed between layers i and j, silicon area equals to the area occupied by the TSV array has to be reserved in both layers. Then, TSV arrays that provide bus connectivity between adjacent layers i and j are connected through special-purpose routing paths, named *TSV networks*. As we will discuss later, these networks are actually implemented with additional metal layers, which exhibit tunable RLC characteristics in order to correspond to those found in TSVs from the selected fabrication process [40].

3.2.4.3 3-D System Prototyping

The last step in our framework, depicted in Figure 3.22, deals with the system prototyping to derive the 3-D stack. During this step we perform floor-planning, power and ground network generation, placement of physical library cells, clock tree synthesis and global/detail signal routing with the *Cadence SoC Encounter* tool. Since it is not possible to modify the functionality of this tool (its source code is not available), we made them aware of the additional flexibility imposed by the third dimension through appropriate design encoding. For this purpose we introduce:

- *TSV networks*: These networks correspond to routing paths that provide connectivity between TSV arrays assigned to adjacent layers. Note that during physical implementation, our framework preserves that TSV arrays assigned at consecutive layers are aligned. This is possible by forcing the placement of TSVs to the same relative (x, y) co-ordinates between adjacent layers. The physical implementation of the *TSV networks* is performed through additional metal layers inserted to the technology library file, while their total resistance (R), capacitance (C) and inductance (L) values correspond to the TSV's RLC parameters.
- *Virtual layers*: Our framework assumes that the target 3-D SoC consists of a number of *virtual layers*, each of which contains hardware resources assigned to different physical layers of the 3-D stack. This enables the usage of existing 2-D physical design tools in order to evaluate the performance enhancement of 3-D chips.

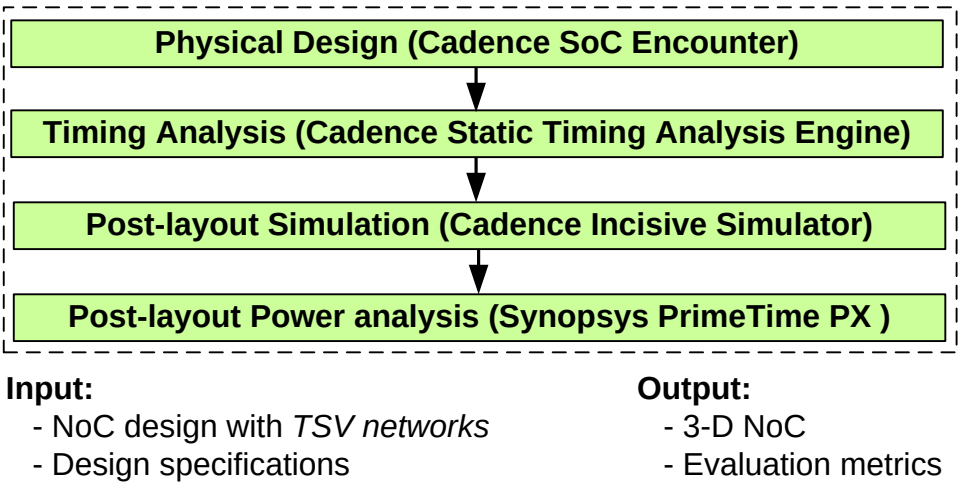


Figure 3.22 Tasks for 3-D system prototyping.

After 3-D physical prototyping, the efficiency of the derived solution is quantified by applying timing analysis. For this purpose we employ *Cadence Static Timing*

Analysis Engine, while for sake of completeness the analysis is performed both in advance, as well as after clock tree synthesis and architecture's routing. In case the derived 3-D stack does not meet the system's constraints/specifications, a number of design optimization could be applied for additional improvements.

Finally, the functional integrity of the physical design is verified by applying a post-layout simulation with *Cadence Incisive Simulator*. For this reason, the delay for all the architecture's routing paths is extracted in SDF format (Standard Delay Format), and then we appropriately annotate the delay values for the *TSV networks*. Note that the annotation of delay values for the *TSV networks* is an important task, since these routing paths has to exhibit the RLC characteristics of the selected TSV technology. For the scope of this research work, the electrical characteristics for *TSV networks* are based on commercially available TSV models [40]. Furthermore, our framework supports the evaluation of 3-D SoCs in terms of power consumption by applying a post-layout analysis with *Synopsys PrimeTime PX* tool. The inputs to this analysis are a trace file that contains signal activities in VCD (Value Change Dump) format, as well as the annotated SPEF (Standard Parasitic Exchange Format) file with extracted parasitic values for all the design's resources.

3.2.5. Experimental Results

This section provides a number of experimental results that prove the efficiency of the proposed solution. For demonstration purposes, four DSP applications are considered as benchmarks. The selection of these applications for evaluation purposes was performed, since their performance highly depends on the data transfer between different storage and processing nodes. Next, we summarize a brief description for the employed benchmark applications.

- Video Object Plane Decoder (VOPD) is a DSP application targeting to high-quality video transition with decent bandwidth performance. The VOPD decoder consists of 16 nodes, including two length decoders, an AC-DC prediction, an ARM processor, two memory components and a VOP reconstructor [50] [41].

- Multi-Window Display (MWD) is another DSP application suitable for implementation with a NoC infrastructure [51]. This benchmark consists of 12 nodes (processing and storage), each of which is assigned to a different router.

- MPEG-4 is a broadly used protocol for audio and video encoding [52]. The employed implementation of MPEG-4 has 12 nodes, including various processing elements, such as a video unit, an audio unit, a RISC processor, a CPU, a binary alpha block and three SRAMs. Since a hardware encoder/decoder consists of many components, a NoC approach is suitable for realizing data transfers.

- The last case study is a multimedia system (MMS) [33] consisted of 25 nodes, including several memories and DSP processors.

Figures 3.23, 3.24, 3.25 and 3.26 give the mapping of the four previously mentioned applications onto the minimum 2-D and the proposed heterogeneous 3-D NoCs, consisted of two layers. The term *minimum* corresponds to the smaller number of routers that are required for performing application mapping, assuming a mesh topology and one node per router. Even though the analysis discussed in this section affects routers with one local port (for attaching the processing, or storage node), the proposed methodology also supports more advanced communication schemes (i.e. clustering multiple nodes to a single router). Similar to topology exploration task discussed in Section 3, the application mapping is performed with an extended version of software tool initially proposed in [41].

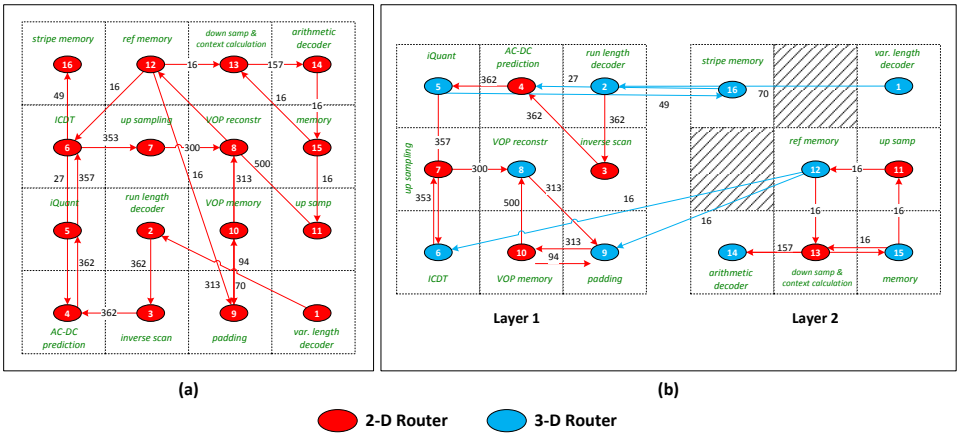


Figure 3.23 Mapping of VOPD application onto: (a) 2-D NoC and (b) the proposed heterogeneous 3-D NoC platform.

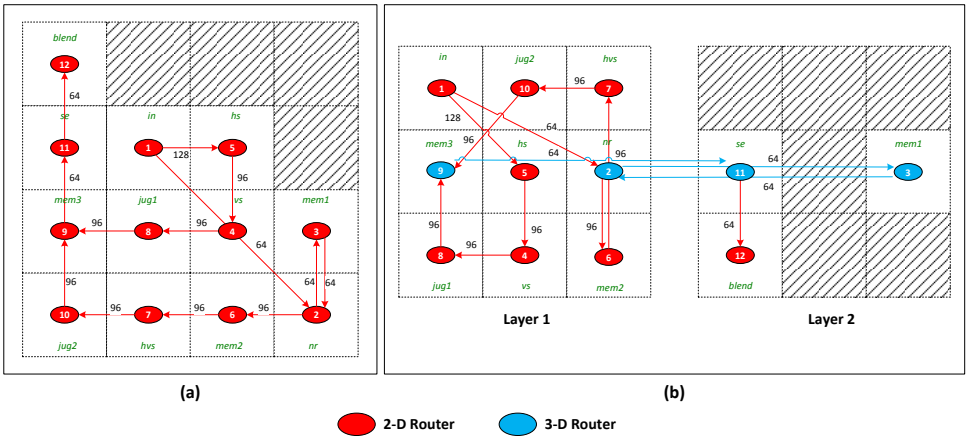


Figure 3.24 Mapping of MWD application onto: (a) 2-D NoC and (b) the proposed heterogeneous 3-D NoC platform.

The red and blue colored circles at these figures encode 2-D and 3-D routers,

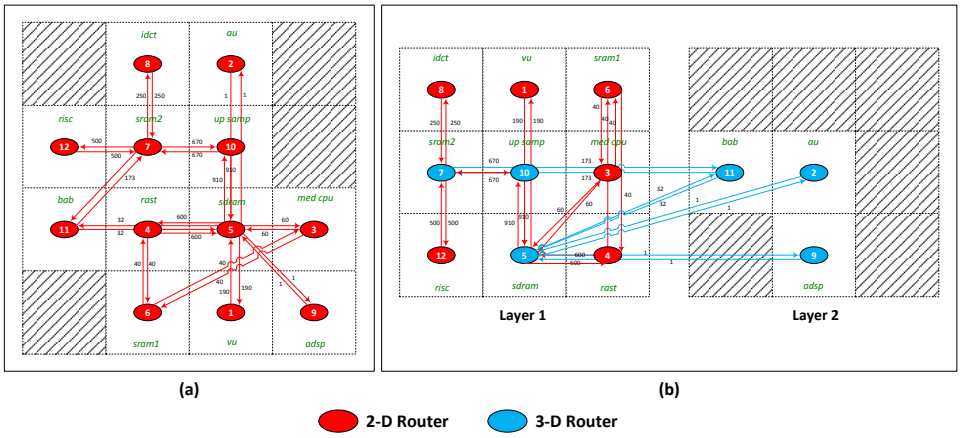


Figure 3.25 Mapping of MPEG-4 application onto: (a) 2-D NoC and (b) the proposed heterogeneous 3-D NoC platform.

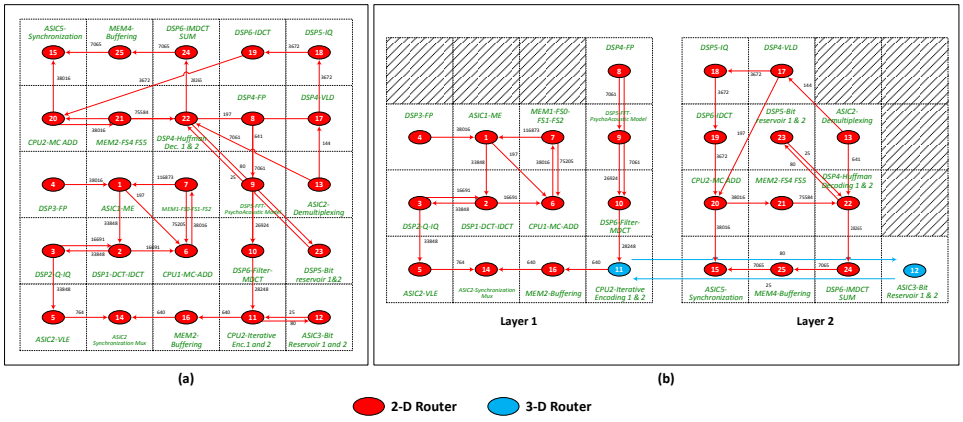


Figure 3.26 Mapping of MMS application onto: (a) 2-D NoC and (b) the proposed heterogeneous 3-D NoC platform.

respectively. Similarly, the directed red and blue colored arrows denote packet transfers between source and destination routers assigned to the same and adjacent layers, respectively. The weights at these arrows correspond to the number of packets that have to be sent from the source to the destination node. Moreover, at these figures, there are also some locations at the grid marked with shadow color. At these locations the mapping algorithm does not assign any node. Hence, for further performance enhancements, it is possible to eliminate the routers from spatial locations without IP blocks.

In advance of quantifying the efficiency of heterogeneous 3-D NoCs, we provide an analysis targeting to evaluate the design of the employed 2-D and 3-D routers as we vary the number of local ports. Note that the case $Node = 0$ corresponds to a scenario, where the router is only connected to the four, or six, adjacent routers,

depending if it is 2-D, or 3-D router, respectively. The scope of this analysis is to depict that SystemC models for 3-D routers exhibit similar efficiency with the corresponding 2-D routers, even though the additional logic required for realizing packet routing in more directions (output ports). More specifically, Figure 3.27 gives the number of equivalent logic gates after synthesizing the two flavors of routers with the usage of TMSC 45nm process technology. We have to notice that the technology synthesis for this experiment was performed by setting identical effort at the C-to-Silicon tool. Based on the results summarized in this figure, we can conclude that, as we increase the number of local ports, there is an almost linear increase to the number of logic gates. Moreover, both the 2-D as well as the 3-D router's implementations requires a similar number of logic gates for a given number of local ports. Two more interesting conclusions can be derived from this figure. Specifically, even though the 3-D router has additional logic for performing packet routing (it has more neighbors compared to the 2-D implementation), however, it incorporates smaller sized SPB buffers, as we have discussed in Section 2. This is also shown in Figure 3.27 for 3-D routers consisted of one, or two, local ports. Regarding higher number of local ports, the complexity of Routing Decision Mechanism becomes more important, and hence the 3-D routers need additional logic gates. Apart from the results summarized in this figure, further reduction to the number of gates is feasible by applying a full-custom designing at the router, instead of employing SystemC models. However, this enhancement is beyond the scope of this research work, as we focus on providing a framework for rapid evaluation of heterogeneous 3-D NoCs with the usage of HLS tool.

Figure 3.27 Evaluation of 2-D and 3-D router in term of equivalent gates.

The different architectural organization between the 2-D and the 3-D routers, both in the routing mechanism and the SPB buffers, is expected to lead to variations

in performance. Towards this goal, the next two figures quantify the efficiency of these routers in terms of latency and energy dissipation for different number of local ports assuming, routers with identical traffic between the 2-D and the 3-D case. Based on Figure 3.28, we conclude that on average the 3-D router exhibits 32% lower latency compared to the corresponding 2-D implementation. This is mainly due to the significant lower (30%) SPB buffer size employed in the 3-D domain. At this figure, we also notice some spikes (non-monotonic improvements) which occur due to the employed HLS tool.

Figure 3.28 Evaluation of 2-D and 3-D router in term of latency.

Figure 3.29 plots the energy dissipation for the different flavors of routers discussed previously. From this analysis, we conclude that 3-D routers outperform the corresponding 2-D implementations. More specifically, the average energy dissipation for 3-D and 2-D routers among the studied benchmarks is 36pJ and 49pJ, respectively. Similar to latency, the 38% additional energy dissipation occurred by 2-D routers is because the 3-D instantiations incorporate smaller SPB buffers. Further improvements are feasible by employing low-power techniques during the design phase, such as power gating and clock gating for those routers that are not utilized, but this analysis is not tackled throughout this research work.

Next, we quantify the physical implementation of different NoC architectures. For this analysis, the four DSP applications discussed previously were appropriately synthesized with the usage of the proposed framework and Cadence SoC Encounter. Three different NoC instantiations are evaluated throughout this analysis, namely: (i) a uniform 2-D, (ii) a uniform 3-D and (iii) the proposed heterogeneous 3-D NoC. The physical implementation was performed with the usage of TMSC 45nm process technology, whereas the parameters for the employed TSV library are summarized

Figure 3.29 Evaluation of 2-D and 3-D router in term of energy dissipation.

in Table 3.2 [40]. Figure 3.30 depicts the floorplan, as well as the layout for one of the studied applications (MPEG-4). At Figure 3.30(a), we highlight the two *Virtual Layers* and the corresponding connectivity between them (plotted with yellow color at Figure 3.30(b)).

Table 3.2 Characteristics of the selected TSV technology.

Diameter:	1.2um
Minimum Pitch:	4um
Resistance:	0.35Ω
Capacitance:	2.5fF
Length:	4-9um

In contrast to application mapping, where the optimization goal was to minimize the number of packet hops, during the physical implementation step we aim to minimize the critical path, in order to derive an efficient communication infrastructure. Hence, the performance metrics for the two flavors of 3-D NoCs studied throughout this research work (uniform and the proposed heterogeneous) differ a lot (due to variations in connectivity imposed by different number of fabricated TSVs).

Table 3.3 provides a number of implementation-oriented parameters for the studied DSP applications, retrieved after successful synthesis and physical implementation. Based on this table, both the 3-D, as well as the introduced heterogeneous solution has aspect ratio 2 (in order to realize onto the same die the two virtual layers shown previously at Figure 3.30). Two more parameters are worth

Figure 3.30 Physical layouts for the introduced heterogeneous 3-D NoC regarding the MPEG-4 application.

mentioning at this table. More specifically, since the proposed heterogeneous 3-D NoC consists of a mixture of 2-D and 3-D routers, it has fewer TSVs as compared to the uniform 3-D NoC implementation. However, such a selection does not impose any penalty in total wire-length, since the proposed heterogeneous approach leads almost to an average 57% wire-length reduction as compared to the case where all the routers are 3-D.

The limited connectivity across the vertical axis found by our proposed architectural solution is expected to lead to a slight increase to the number of packet hops for successful routing between source and destination nodes. For this purpose, Figure 3.31 plots the total number of hops per application for the three alternative NoC topologies. Based on these values we can conclude that the existing way for designing uniform 3-D NoCs, where all the routers provide connectivity to upper/lower layers, is the optimal topology since it reduces the number of packet hops

Table 3.3 Implementation properties for the target applications.

Benchmark		Properties						
		Area(um^2)	Ratio	Wire-length	# of TSVs	# of cells	# of nets	# of IOs
MMS	2-D	991.5×990.3	0.99	10,261,064	0	458,823	462,825	3,251
	3-D	599.3×1,198.2	1.99	11,480,859	704	324,689	326,317	3,251
	Proposed	762.8×1,523.3	1.99	7,864,344	130	534,731	539,864	3,251
MPEG-4	2-D	754.74×753.48	0.99	5,915,764	0	256,980	259,307	1561
	3-D	432.03×861.84	1.99	6,373,170	576	167,911	168,709	1,561
	Proposed	561.7×1,122.66	1.99	3,882,465	192	271,828	275,552	1,561
MWD	2-D	758.9×756.0	0.99	5,566,222	0	262,609	265,219	1,561
	3-D	435.4×870.6	1.99	5,794,055	1,170	171,979	172,762	1,561
	Proposed	551.5×1,101.24	1.99	3,673,350	260	272,823	275,523	1,561
VOPD	2-D	781.3×777.4	0.99	6,156,379	0	283,098	285,961	2,081
	3-D	441.3×885.7	2.00	7,081,633	1,170	178,034	179,182	2,081
	Proposed	585.8×1,170.5	1.99	4,121,908	650	318,237	320,404	2,081

compared to the corresponding 2-D NoC by 29%, on average. On the other hand, the proposed heterogeneous NoC leads to an increase of packet hops against to uniform 3-D NoC about 8.5% on average.

Figure 3.31 Number of packet hops for different architectural solutions: (i) a homogeneous 2-D NoC, (ii) a homogeneous 3-D NoC and (iii) the proposed heterogeneous 3-D NoC.

The previously mentioned reduction of packet hops highlights the importance of employing the 3-D integration paradigm as a viable solution for designing digital circuits with increased demand for connectivity. Even though the conclusion about superior flexibility of 3-D NoCs seems obvious and was also mentioned in the majority of relevant publications, however, these approaches rarely take into account constraints and limitations posed by the fabrication process. Specifically, the usage of 3-D routers impose a TSV assigned to each wire that provides signal connectivity between adjacent layers. Due to the large diameter and pitch of these TSVs, it is clear that an increased number of TSVs leads to area, delay and power overheads.

Figure 3.32 plots the maximum operation frequency for the alternative topologies. A number of conclusions can be derived from this figure. More specifically, the proposed heterogeneous 3-D NoC outperforms the rest implementations for all the studied applications. However, depending on the inherent properties of each application (i.e. the demand for packet transmission), as well as the already derived

application mapping onto target topologies, there are mentionable variations in performance improvement between the introduced solution and uniform 3-D ranging from 1.16× up to 1.43×. One more conclusion is derived from this figure regarding the performance of uniform 2-D and 3-D NoCs, as these two NoCs achieve on average almost the same maximum operation frequency among the studied applications. This occurs because the excessive amount of vertical connectivity usually leads to a saturation, and no additional performance improvement is feasible by introducing additional TSVs.

Figure 3.32 Maximum operation frequency for different instantiations of NoC.

Similarly, Figure 3.33 provides the power consumption for the alternative topologies. From this figure we can conclude that on average the proposed heterogeneous solution, consisted of a mixture of 2-D and 3-D routers, achieves mentionable power savings against uniform 2-D and 3-D NoCs. More specifically, since the 3-D routers have fewer SPB buffers, this leads to power savings for the NoCs that incorporate such kind of routers. Additionally, the shorter routing wire-length found in heterogeneous 3-D NoCs, as we discussed in Table 3.3, also contributes to the smaller power dissipation.

Figure 3.33 Power consumption for different instantiations of NoC.

3.2.6. Conclusions

In this chapter, it is introduced a software-supported framework supporting rapid evaluation of heterogeneous 3-D NoCs. The introduced architectural paradigm consists of a combination of 2-D and 3-D routers that better match to the application's requirements for data transfer. Experimental results with DSP applications prove the effectiveness of such an approach, as compared to existing solutions for designing uniform 2-D and 3-D NoCs, since we achieve on average 25% higher maximum operation frequency and 39% lower power consumption, as compared to the uniform 3-D NoCs.

3.3. Space Critical Systems

3.3.1. Introduction

The exploration of Mars is one of the main goals both for NASA and ESA, as it is confirmed by past and recent activities. The last 15 years there are numerous on-orbit and surface missions to Mars (e.g., NASA's Mars Global Surveyor, Mars Odyssey, Phoenix, Mars Reconnaissance Orbiter, Mars Express) with remarkable results.

One of the most challenging tasks for these missions is the design of autonomous robots. Since the efficiency of these robots is tightly coupled to the accuracy of their decision-making algorithms [53], there is a continues effort towards developing even more efficient computer vision (CV) algorithms.

Even though these solutions become more and more attractive due to their remarkable efficiency in term of accuracy, however, their increased demand both for computational resources and storage make them non-suitable for being incorporated onto space missions. To make matters worst, the majority of these algorithms are implemented solely onto software-level (e.g. Matlab, C/C++), while the employed coding style make them almost imposable to be ported onto hardware.

In this research work we discuss a novel methodology for efficient implementation of CV algorithms targeting to rover navigation for space applications. The introduced solution is part of the SPARTAN (SPAring Robotics Technologies for Autonomous Navigation) project, founded by ESA, as part of the ExoMars mission (scheduled for 2018) [54]. Rather than relevant approaches that tackle the problems of increased computational complexity and execution time solely at software-level, our objective is to incorporate an architecture consisted of a reconfigurable device (FPGA) and a space-oriented central processor unit (CPU). For the scopes of SPARTAN project, the target reconfigurable fabric is a Virtex-6 FPGA platform [55], whereas the processor is a rad-hard CPU with 150MIPS running the ROS system [56] (the hardware specifications was defined by ESA). Additional details about the architecture of SPARTAN system can be found in [57].

Since our target architecture is a heterogeneous platform, we incorporate a hardware/software (HW/SW) co-design methodology, which is software-supported by a number of existing and new tools. A critical task at this methodology is the determination of target implementation medium, either the CPU or the FPGA, where each of the CV algorithms has to be mapped. In order to accomplish this task, we employ a novel profiling methodology which provides some guidelines about the importance of different CV algorithms under various design parameters. More specifically, our selections are based onto the computational complexity, the type of mathematical operations, the memory footprint (both data life-time and maximum storage), the amount of data that has to be transferred among these kernels, the data dependency, etc for each CV algorithm.

Even though profiling is a well-known problem for architectural designers, the main differentiation of introduced solution compared to relevant approaches affects the significant lower execution time. More specifically, rather than profiling algorithms with original data inputs, our methodology employees a technique that guarantees to derive accurate conclusions with the usage of smaller sized images.

Such a feature reduces mentionable the execution time for performing profiling.

The rest of this work is organized as follows: subsection 3.3.2 gives an overview of the SPARTAN system. The proposed methodology is described in subsection 3.3.3, whereas experimental results derived by applying the introduced methodology to SPARTAN system are discussed in subsection 3.3.4. Finally, conclusions are summarized in subsection 3.3.5.

3.3.2. Architecture of SPARTAN System

The goal of the SPARTAN system is to convert visual information from rover cameras into 3D local maps, as well as to perform accurate location estimates useful for the navigation process. Hence, image processing algorithms suitable for 3D map reconstruction and localization are selected (from [58]) and implemented into a parallel processing chain to achieve high performance while maintaining efficiency in terms of computational complexity, memory footprint and energy consumption.

The architecture of SPARTAN system consists of a low-performance CPU (150MIPS) and a Virtex-6 FPGA board. More specifically, the CPU runs the Robotic Operating System (ROS) in order to provide all the standard operating system services required for proper rover navigation, such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management, whereas the FPGA acts as an accelerator for the computational intensive CV algorithms.

The desired functionality of SPARTAN system is encoded with two different operation modes, namely the mapping and localization mode. Whenever the system operates at mapping mode, it produces a 3D map of the environment. On the other hand, at the localization mode, the system is configured to generate location estimates.

As input to the SPARTAN system, we use stereo-vision image information captured from two sets of cameras. The image sizes from these cameras are 1120×1120 (mapping mode) and 512×384 (localization mode), respectively.

Overall the SPARTAN system uses a module for the 3D reconstruction of images [59], i.e., for the computation of disparity map, as well as the coordinates of objects in 3D space, the *SURF* algorithm for feature extraction [60], a motion estimation module that aims to deduce moves in 3D space [61], and a non-probabilistic visual SLAM [61]. The *3D reconstruction* module is used both in mapping and localization mode, while the remaining three modules are employed solely for *visual odometry* in localization mode.

A schematic view of SPARTAN architecture is depicted in Fig. 3.34, whereas the functionality of employed CV algorithms is summarized as follows [57]:

- **Imaging**: Performs image processing for deriving appropriately input to the rest algorithms.
- **3D Map Reconstruction**: Produces the 3D map of the environment.
- **Visual Odometry**: Provides an estimation of the displacement of the rover.
- **Visual SLAM**: Determines the current location of the rover.

- **Localization**: Finds the new spatial location of rover at the map.

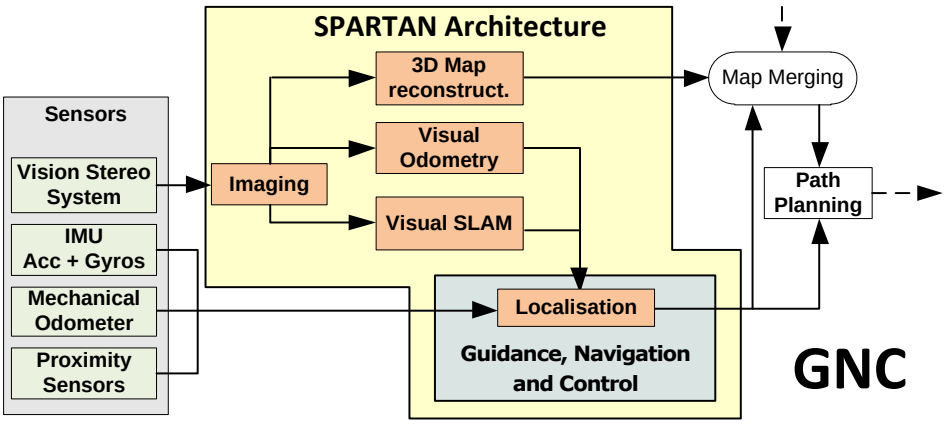


Figure 3.34 Schematic view of SPARTAN mapping and localization mode.

The demonstration of SPARTAN system will be performed with the usage of two scenarios: (i) an Exomars-like rover under the *3DROV* simulator and (ii) with a real rover in an outdoor environment. More specifically, the first scenario involves a simulated rover (Exomars CAD model based) and a Martian surface modeled on the *3DROV* framework. Realistic camera images will feed the vision based SPARTAN algorithms. This scenario allows early testing of the algorithms during the development process, whereas the final demonstration (with the real rover) will prove the effectiveness of proposed architecture.

3.3.3. Hardware/Software Co-Design Methodology

This section describes the proposed methodology for performing HW/SW co-design. Even though the introduced methodology is a general-purpose solution, however, in the context of SPARTAN project it was appropriately tuned to take into consideration both the inherent architectural features posed by our system (FPGA and a low-performance CPU), as well as the constraints related to the increased computational complexity, as they are defined by the ESA's specifications. Fig. 3.35 highlights the main tasks of our proposed co-design methodology.

3.3.3.1 Phase 1: Algorithmic analysis

The first phase of proposed solution, depicted in Fig. 3.36, involves theoretical study of the employed CV algorithms. Algorithm analysis is an important part of our methodology, since it allows theoretical estimations about the resource requirements posed by CV algorithms.

For this purpose, initially we check whether the employed CV algorithms are described in primitive form. By the term primitive, we refer to an algorithmic description without complex functions and libraries. This is very crucial in order to have

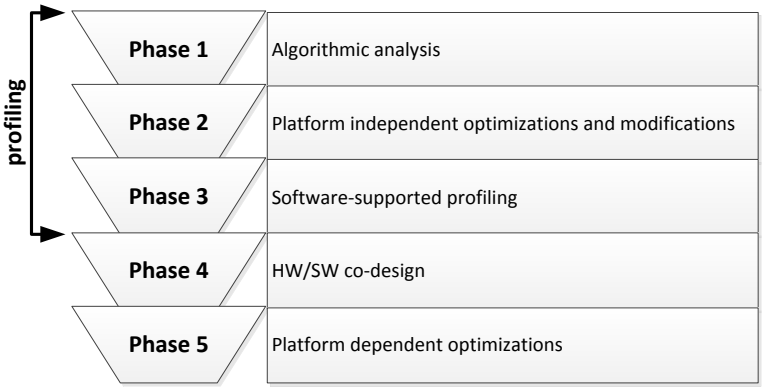


Figure 3.35 Abstract view of the introduced HW/SW co-design methodology.

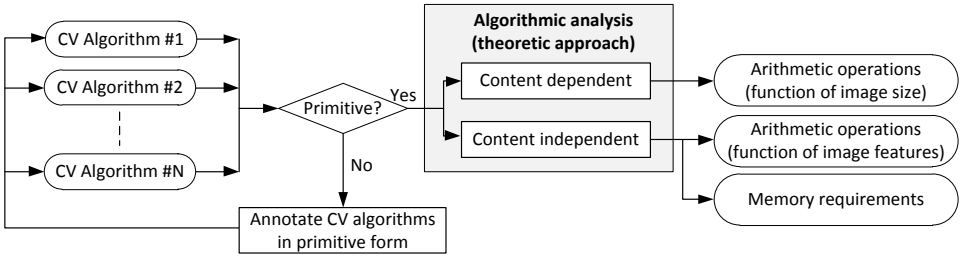


Figure 3.36 Tasks performed during the algorithmic analysis.

accurate profiling results (note that execution time for complex functions in Matlab could be a few seconds, whereas if these functions are executed onto C/C++ might take a long time period). Hence, in case the target algorithms are not described in primitive form, there is a pre-processing step. During this step we pay effort to rewrite these functions into a primitive form without affecting their functionality.

Otherwise (i.e. the algorithms are described in primitive form), we proceed with the algorithmic analysis. Two alternative approaches are feasible for this task. Either to study a content independent, or a content dependent algorithm. More specifically, whenever the complexity of studied CV algorithms does not depend to the content of image (content independent), then theoretical analysis provides accurate results about the number of mathematical operations required for executing these algorithms as a function of image size. Even though the image sizes employed as input to the SPARTAN system are fixed (defined by the project specifications), and consequently, there is no variation to the number of mathematical operations, the algorithmic analysis provides also useful information for applying the extrapolation method (discussed in upcoming subsection).

On the other hand, if we have to study a content dependent algorithm, the theoretical analysis reports the number of mathematical operations as a function of the identified features at images. The algorithmic-dependent profiling method

is the procedure of determining the amount of resources (e.g. time and storage) required for algorithm's execution.

The outcome from this analysis provides valuable conclusions about the computational complexity of CV algorithms, their performance requirements, the memory/storage requirements, as well as the communication load among them. This information can be appropriately handled to gives some potential algorithmic optimizations for the CV algorithms and to provide some guidelines regarding the HW/SW partitioning.

3.3.3.2 Phase II: Platform Independent Optimizations and Modifications

The second phase of our propose methodology applies at algorithmic level a number of platform independent optimizations and modifications. This task is achieved through source-to-source code modifications. Fig. 3.37 depicts the tasks performed at the second phase.

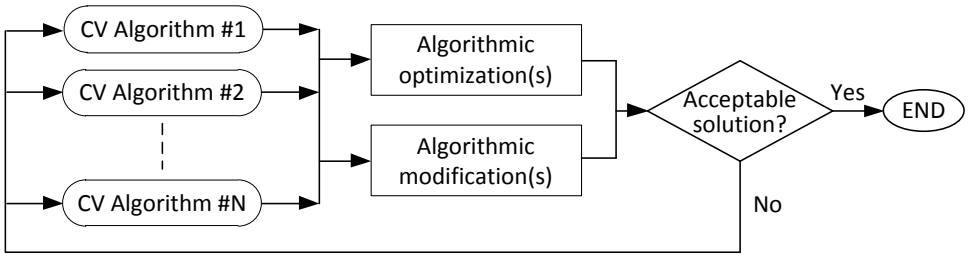


Figure 3.37 Proposed methodology for performing platform independent optimizations and modifications.

As we have already mentioned, the initial version of CV algorithms employed for the scopes of SPARTAN project, were not developed into a hardware-friendly description. Typical example of this limitation is the usage of high-level functions found in Matlab and/or C++, which are very difficult to be implemented onto hardware. Also, they did not take into account issues related to parallel execution (which is provided by target FPGA), as well as the limited amount of on-chip storage found in embedded platforms. Consequently, those algorithms have to be appropriately optimized in order to take into consideration a number of architectural constraints. Note that during this task, we have to preserve the functionality of target CV algorithms. Additional details about the employed algorithmic optimizations can be found at [62].

Apart from these optimizations, during the second phase of introduced methodology, we apply also a number of algorithmic modifications. The scope of these modifications is to reduce considerable the complexity, storage and communication overhead for the employed CV algorithms with an controllable (affordable based on project specifications) penalty to their accuracy. On contrast to algorithmic optimizations, the applied modifications usually affects the functionality of underline CV algorithmic.

Then, the efficiency of CV algorithms is quantified in terms of performance and

the desired accuracy. In case the derived solution does not meet system's specifications, there is a feedback loop for additional improvements. Otherwise, we proceed to the third phase of proposed methodology.

3.3.3.3 Phase III: Software-Supported Profiling

Profiling is an important procedure during HW/SW co-design. The importance of profiling task becomes far more critical whenever the system implementation imposes some form of co-design between non-homogeneous processing cores (e.g. the SPARTAN system incorporates a low-performance CPU and an FPGA board).

Our methodology applies system profiling in different level of abstractions. More specifically, regarding the SPARTAN project, two levels of abstraction were employed: (i) coarse-grain and (ii) fine-grain. Such an approach enables designers to perform rapid evaluation of complex systems, where only a subset of the algorithms have to be studied in detail (under fine-grain analysis). More specifically, even though fine-grain analysis guarantees to find the most accurate and detail results about those kernels that dominate system's performance, however it is difficulty applicable to algorithms with increased code size.

Our software-supported profiling applies initially a coarse-grain analysis to the whole algorithm in order to identify critical kernels (those that dominate system's performance). Then, these kernels are studied with the usage of fine-grain profiling in order to retrieve a number of design parameters that affect systems implementation (e.g. computational complexity, memory footprint, data lifetime, communication bandwidth, area requirements, etc).

Since the profiling of complex systems is a rather difficult task, our methodology uses a number of software tools. Specifically, the coarse-grain analysis is performed with the usage of *Tic-Toc* and *Time* functions, depending on the programming language for each CV algorithm. On the other hand, fine-grain profiling is performed with algorithmic analysis, as well as with dedicated software tools (e.g. Matlab profiler, VTune, Valgrind, etc). This classification is also depicted in Fig. 3.38.

Figure 3.38 Profiling tools.

Next, we summarize the main advantages and disadvantages of the alternative software-supported analysis methods:

- *Tic-Toc*: This method is applicable to CV algorithms described in Matlab language. The output from *Tic-Toc* analysis provides an overview of the execution time for different parts of the algorithm. Since this method is applied manually only to kernels of interest, it's applicability decreases with the code size. Additionally, the *Tic-Toc* method introduces an overhead, which is not easily controllable. For the scopes of SPARTAN project, this overhead was computed based on information published in relevant references (e.g. manual of Matlab).

- *Time*: The *Time* method reports the execution time for CV algorithms developed in C/C++ (e.g. OpenCV) language. Similar to the *Tic-Toc*, this method also introduce an overhead due to the operating system. In order to minimize the impact of this overhead at SPARTAN project, all the CV algorithms were profiled onto the same PC under similar workload.

- *Algorithmic analysis*: This method provides the maximum accuracy, but it is not suitable for increased code sizes because it is applied manually. Furthermore, *algorithmic analysis* does not take into account the overheads imposed by the selected description language (e.g. C++, VHDL).

- *Matlab/C++ profiler*: The last method involves algorithmic profiling with the usage of existing tools (profilers). For the scopes of SPARTAN project, we employ the Matlab's method of source level instrumentation. This method reports for each function the execution time, number of calls, parent functions, child functions, code line hit count, etc. Even though this approach derives the maximum possible information from profiling step, assuming that the algorithm is written in primitive form, however, it imposes the maximum execution overhead.

As we have already mentioned, software-supported profiling is a memory limited procedure, which becomes far more savage whenever the input images have increased size. Regarding the SPARTAN project, the mapping mode incorporates three pairs of stereo images, each of which has 1120×1120 pixels with 200 disparity levels, whereas localization mode uses a pair of stereo images with size 512×384 pixels.

Due to the increased image sizes, if we apply conventional profiling, it will be a timing consuming procedure. In order to alleviate this limitation, our methodology incorporates a technique which is based on smaller size images. This solution can estimate with good fidelity a number of architectural and algorithm oriented parameters, but in significant shorter time period. Additional inputs to this technique are the conclusions derived previously from algorithmic analysis.

Having as input this information, our profiler is able to estimate the memory footprint and the time complexity both for each CV algorithm, as well as for the entire SPARTAN system. These values are fed as input to our extrapolation method

[63] in order to estimate the corresponding values regarding the actual image sizes for SPARTAN system. Since there might be some variations between the estimated values (as they retrieved from extrapolation method) and the actual implementation results, there is an additional step, where we can tune the weighting factors for the employed extrapolation method. Note that the tuning task is performed once, whereas the derived weighting factors are used for the rest projections.

Fig. 3.39 depicts the proposed methodology for performing software-supported profiling of CV algorithms.

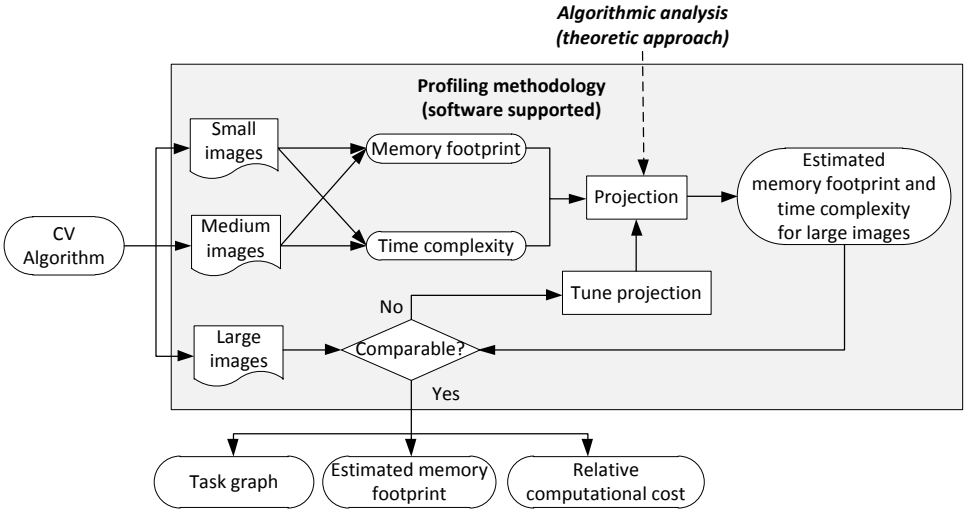


Figure 3.39 Proposed methodology for performing software-supported profiling.

3.3.3.4 Phase IV: HW/SW Co-Design

The proposed methodology for performing HW/SW co-design is depicted in Fig. 3.40. One of the most critical tasks in this methodology affects the algorithm partitioning onto the hardware resources based on the profiling results retrieved previously. As we have already mentioned, the fidelity of profiling conclusions depends on the description (primitive or non-primitive) form of CV algorithms.

Having as input the decisions from partitioning, the next step in our methodology deals with the development of C/C++ and ESA compatible VHDL for the CV algorithms. The developed C/C++ code will be integrated onto a ROS-based PC, whereas the VHDL code will be mapped onto a Virtex-6 FPGA board. The selection of these two hardware platforms is based on ESA's requirements.

The developed algorithms are evaluated with the usage of three different scenarios: (i) each algorithm is quantified as a stand-alone solution, (ii) all the algorithms that realize mapping mode are evaluated against to project specifications for 3D map reconstruction and (iii) the efficiency for all the algorithms that realize localization mode is quantified against to project specifications for location estimations.

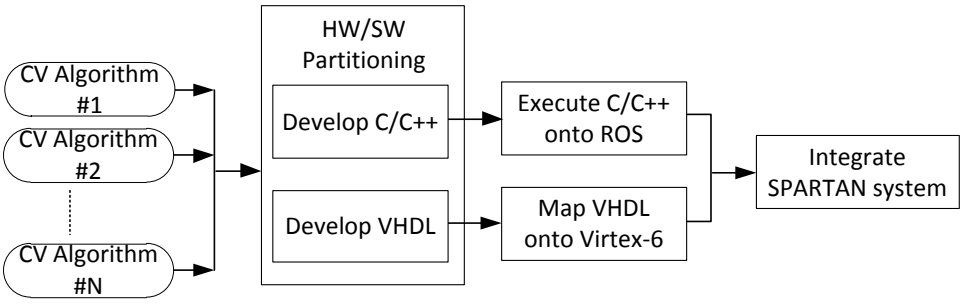


Figure 3.40 Proposed methodology for performing HW/SW co-design.

3.3.3.5 Phase V: Platform Dependent Optimizations

The last phase of our introduced methodology, depicted schematically at Fig. 3.41, applies a number of platform dependent optimizations aiming to further improve the performance of target system. For this purpose, both the efficiency of each CV algorithm, as well as the entire system is compared against to the SPARTAN specifications.

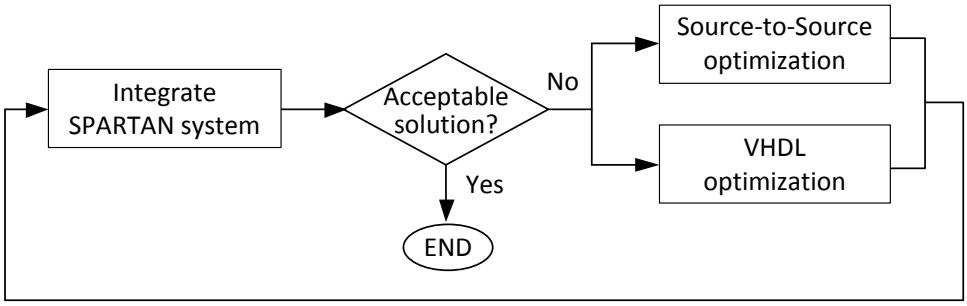


Figure 3.41 Proposed methodology for enhancing the performance of SPARTAN system with platform-dependent optimizations.

Since the consortium of SPARTAN project does not have access to the low-performance processor (150 MIPS) defined from ESA's specifications, the timing evaluation of derived solution was performed as follows: Initially, we measure both the time spent by the kernels running in the CPU T_{cpu} and that spent by the kernels running on the FPGA T_{fpga}. Then, we compute the total execution time $T_{total\_projected}$ after projecting the CPU time into a CPU with 150 MIPS with Equation 1.

$$T_{total\_projected} = T_{fpga} + T_{cpu} \times \frac{150}{MIPS} \tag{3.3}$$

In case the derived decisions about HW/SW partitioning, as well as the existing C/C++ and VHDL implementation of the CV algorithms, meets project specifications, we proceed to the final demonstration. Otherwise, an additional optimization

step is required in order to further improve the performance of entire system. In contrast to previous optimizations, at this phase we apply platform dependent optimizations (e.g. parallelism extraction, pipelining, etc), which take into consideration inherent architectural issues provided by underline platform.

Note that during this phase, it is highly preferable not to alter the already derived decisions about HW/SW partitioning. Otherwise, it is imposed that additional algorithmic kernels have to be developed from scratch onto a new platform (either the CPU or the FPGA), which in turn introduces mentionable architectural modifications, and consequently increased design cost. Furthermore, any potential architectural modification might impose variations to data/signal transfer between hardware platforms, which should be appropriately tackled.

3.3.4. Experimental Results

This section provides a number of results about the first part of our design methodology. More specifically, we discuss the conclusions derived after applying the profiling procedure to CV algorithms employed to the SPARTAN project [59] [60] [61] [64].

3.3.4.1 Overall profiling results

Fig. 3.42 presents the profiling results derived by studying the CV algorithms employed for the scopes of SPARTAN project both for the mapping (Fig. 3.42(a)) and localization mode (Fig. 3.42(b). More precisely, this figure reports the experimental results acquired by (i) the Matlab profiler, (ii) the "Tic-Toc" method and (iii) the algorithmic analysis. The vertical axis at this figure corresponds to the time complexity. For demonstration purposes, this axis gives the relative cost among the different CV algorithms and profiling methods, as a percentage of the total time (assuming distinct operation of mapping and localization modes).

Based on the results summarized in this figure, we can determine the importance of different kernels. More specifically, the *rec3d*, *disparity* and *aggregation* are the most timing consuming kernels at mapping mode, whereas regarding the localization mode the most critical kernel is *aggregation*. These conclusions provide to designers valuable information about the kernels that have exhibit increased performance bottlenecks, and hence they have to be mapped onto FPGA in order to speedup their execution. Additionally, we have to notice, that all the three employed profiling method denote the importance of the same kernels, which also proves our claim that the introduced methodology can derive conclusions with high fidelity.

3.3.4.2 Data Life-Time

Next, we provide some experimental results about the data life-time. These results, depicted in Figs. 3.43(a) and 3.43(b), respectively, for mapping and localization modes, are retrieved with algorithmic analysis. These values were computed by studying the source code assuming sequential execution. Note that during this

Figure 3.42 Profiling results (a) for mapping mode and (b) for localization mode.

analysis we do not take into consideration any potential parallelism, since the initial version of CV algorithms does not provide such feature.

Based on the results summarized in Fig. 3.43, we can conclude that two kernels, named *differences cube left* and *differences cube right*, exhibit the maximum demand for storage (on-chip memory). More specifically, these two kernels store almost 4GB of data, while this information has to be available for five kernels (namely *absolute differences*, *normalized ADs*, *aggregation*, *min disparity search* and *interpolation*).

The excessive amount of storage, which is not available to the SPARTAN system (based on project specifications the SPARTAN system has to incorporate up to 500MB of storage), imposes that we have to apply a number of algorithmic optimizations and/or modifications (as it was discussed in Section III).

3.3.4.3 Estimations about HW/SW Partitioning

Throughout this analysis, we provide also some estimations about the HW/SW partitioning. More specifically, by applying the proposed HW/SW co-design methodology, we found the target platform for the kernels of SPARTAN system. The input to this procedure is the profiling results derived at previous phase. Among others, we take into consideration design parameters that affect the computational complexity of different kernels, their memory requirements (both storage size and data life-time), as well as the communication load. The results from this analysis regarding the mapping and localization mode are summarized in Figs. 3.44 and 3.45, respectively.

Based on this analysis, the kernels that form mapping mode are clustered as follows:

- Kernels with reduced timing complexity: *debayer*, *contrast*, *rectify*, *superpo-*

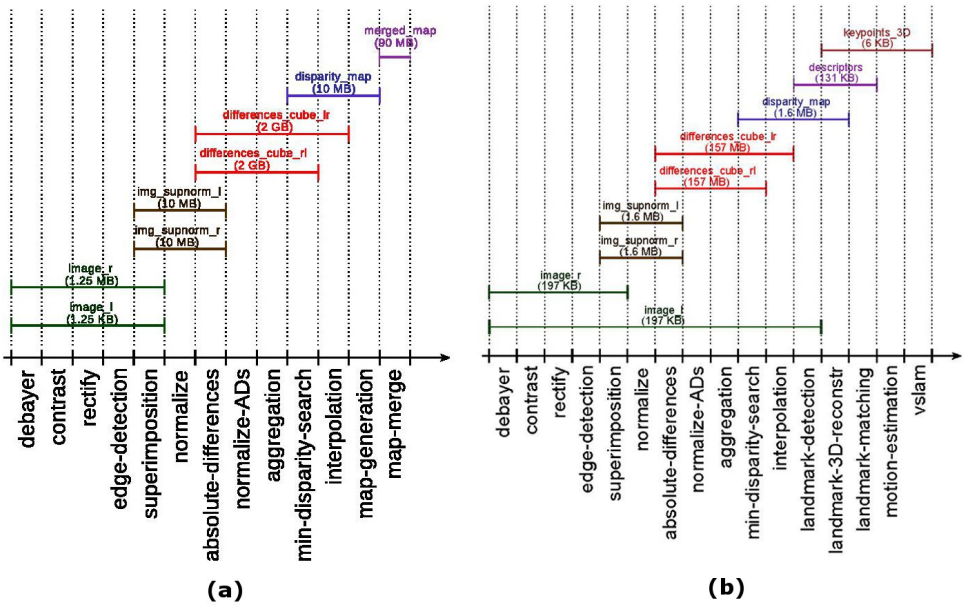


Figure 3.43 Data life-time (a) for mapping mode and (b) localization mode.

sition, *normalize*, *subpixel interpolation*, *mapgen* and *mapmerge*.

- Kernels with medium timing complexity: *edge detection*, *absolute differences*, *normalized absolute differences* and *minimum disparity search*.
- Kernels with increased timing complexity: *aggregation*.

Similarly, regarding the localization mode, the corresponding classification of kernels follows:

- Kernels with reduced timing complexity: *debayer*, *contrast*, *rectify*, *superposition*, *normalize*, *subpixel interpolation*, *landmark 3D reconstruction*, *motion estimation* and *VSLAM*.
- Kernels with medium timing complexity: *edge detection*, *absolute differences*, *normalized absolute differences*, *minimum disparity search*, *landmark detection* and *landmark matching*.
- Kernels with increased timing complexity: *aggregation*.

The timing analysis discussed previously shown that isolating the *aggregation* component during the HW/SW partitioning procedure imposes a significantly large amount of data that has to be transferred from the PC to the FPGA and vice versa (up to 8 GB per kernel). Since this amount of data is not affordable for meeting SPARTAN specifications (for instance entire mapping mode has to be executed at 1

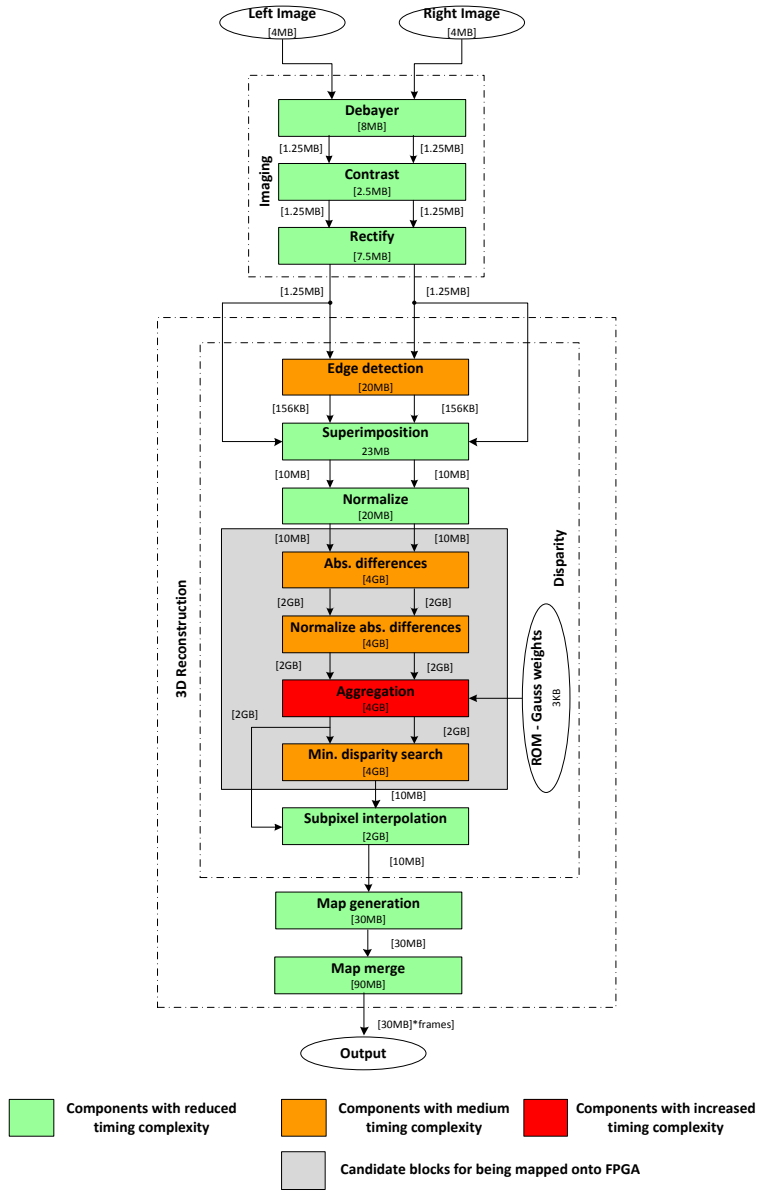


Figure 3.44 Coarse-grain level HW/SW partitioning for mapping mode.

second), a number of alternative clustering options, which reduce significantly the communication load, are also studied.

The outcome from this study is highlighted in Figs. 3.44 and 3.45 with grey color. More specifically, by implementing the *absolute differences*, *normalize absolute differences*, *aggregation* and *minimum disparity search* kernels onto the same

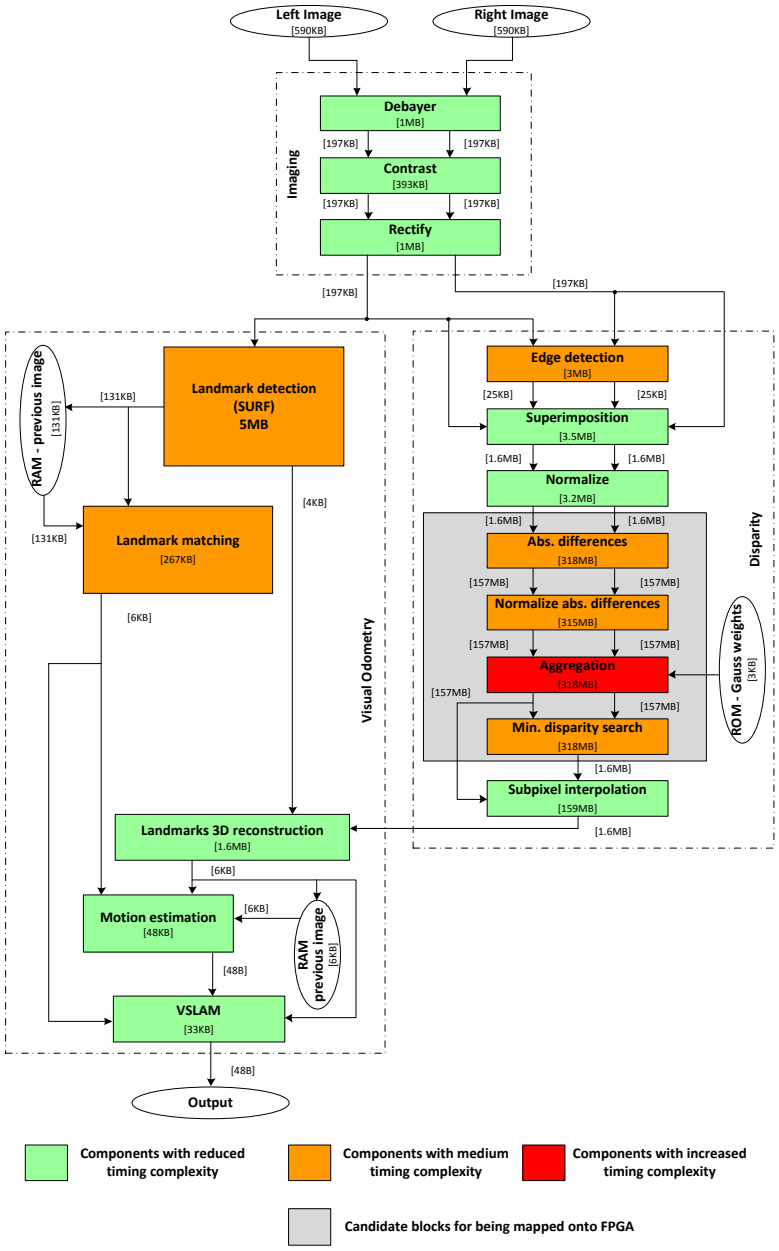


Figure 3.45 Coarse-grain level HW/SW partitioning for localization mode.

platform (FPGA), the derived architecture requires only 30MB of data to be transferred from CPU to the FPGA.

Besides communication, the decisions about HW/SW partitioning depend upon

the efficiency of certain algorithm modifications for reducing the memory (storage) requirements. This is especially crucial for FPGA due to limited on-chip memory. The outcome from profiling, depicted at Fig. 3.44, shows that in case *aggregation* is implemented as a stand-alone kernel, it requires about 4 GB of on-chip memory. In order to avoid such a non-optimal resource utilization, we considered to study the possibility of interleaving their algorithmic steps for computing iteratively each disparity level in advanced of proceeding to the next one. Such an architectural modification enables SPARTAN system to reuse the storage of each disparity level, and hence to minimize the total memory requirements, without affecting the functionality of CV algorithms.

3.3.4.4 Data-flow Analysis

The data-flow analysis for the two operating modes (mapping and localization) are depicted in Figs. 3.44 and 3.45, respectively.

Regarding the mapping mode, none of its kernels could be executed in parallel, since there is a data dependency among them. However, this conclusion imposes that we have to exploit as much as possible the inherent parallelism found inside each kernel in order to meet the timing specifications. Additional performance enhancement is feasible by applying pipelining techniques.

On the other hand, Fig. 3.45 denotes that *disparity* and *visual odometry* kernels are candidate to be executed in parallel, since there is no data dependency among them. Similar to mapping mode, additional performance enhancement is feasible by exploiting techniques like extracting inherent parallelism and applying pipelining.

3.3.5. Conclusions

This research work presents a novel methodology for efficient HW/SW co-design via the partitioning of processing tasks to the available resources of a heterogeneous system, composed of general purpose processors and reconfigurable logic. The proposed systematic methodology examines the computational complexity, the memory requirements (storage size and variables data life) as well as the communication cost, in order to find an optimum architecture of coarse-grained accelerators for speeding-up the overall system execution time. This approach was employed in the design of a heterogeneous system for autonomous space rover navigation through the use of computer vision algorithms. The implemented system meets the standards set by the ESA organization, given the scopes of SPARTAN project [65].

3.4. 3-D Integration for Digital Signal Processing SoC Architectures

3.4.1. Introduction

In recent years, 3-D IC has attracted more and more attention. Along with the technology updates, there are several published works dealing with the 3-D physical design problem. Among others, tools for partitioning, floor-plan, placement and routing for 3-D architectures, have been proposed. These approaches are based almost exclusively on academic tools. On the other hand, the only known commercial framework for supporting the design of 3-D SoCs is provided by R3Logic Corp.[66]. In this letter we introduce a novel framework for supporting rapid evaluation of 3-D SoCs with the usage of existing CAD tools. Such a framework is crucial even before physical design tools for the 3-D domain become commercially available, since it provides a good estimation about the potential benefits from designing 3-D chips.

We note that a part of the design methodology presented in this work is also part of the design flow of the framework *Plug&Chip* presented at Chapter 2 and mentioned in subsection 2.3.2. However the last one work is an optional step of the rapid prototyping methodology proposed by this thesis, while the present work is a systematic study on designing 3-D signal processing architectures.

3.4.2. Proposed Framework

This section introduces the proposed framework for performing rapid evaluation of 3-D SoCs. This framework, depicted in Fig. 3.46, consists of three modular steps in order to enable interaction with tools from similar and/or complementary flows. More specifically, the steps of our framework are summarized as follows:

- *Pre-processing Step*: Verification of functional integrity for the design and extraction of its XML description.
- *3-D Stack Generation*: Generation of the 3-D stack and determination of the communication (routing paths) among layers.
- *3-D System Prototyping*: Physical implementation of 3-D SoC and evaluation of the derived solution.

Initially, the architecture's HDL description (i.e., VHDL, Verilog) is simulated under various parameters and constraints (e.g. clock period, on-chip memories organization) in order to verify the system's functionality. For this purpose, we employ the *Cadence NC-sim* simulator.

Then, we determine the desired hierarchy for target 3-D architecture. Our framework can handle different levels of hierarchy. More specifically, a block-based system's description leads to a coarse-grain solution, whereas a gate-level netlist comes with a finer system implementation. On other words, the fine-grain approach imposes the highest performance enhancement for the 3-D architecture, but it also introduces the maximum computational complexity for performing architecture-level exploration. For the scopes of this letter, we choose (without affecting the

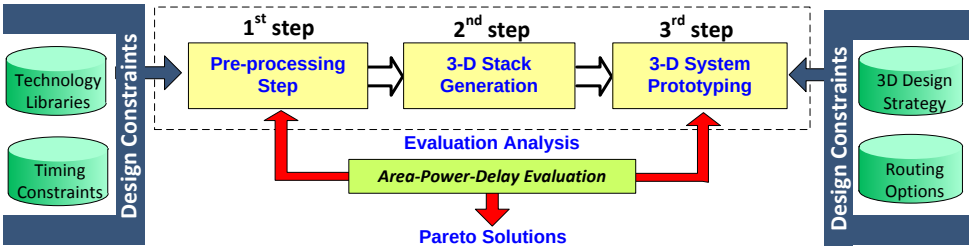


Figure 3.46 Proposed framework for supporting rapid evaluation of 3-D SoCs.

generality of proposed framework) to maintain the system's hierarchy among heterogeneous modules (e.g. logic, memory), while each module is being flattened in order to maximize the performance enhancement.

After defining the SoC's hierarchy, the HDL description is synthesized with *Synopsys Design Compiler*. As long as design constraints (e.g. timing slacks, DRC's, etc) are met, the output from synthesis is translated to an equivalent XML description, which corresponds to the system's hypergraph representation. This task is software supported by our new public available tool, named *Net2XML*. The derived XML description is fed as input to the second step of our proposed framework, which deals with the 3-D stack generation under the selected design constraints.

The second step involves application's partitioning into a number of subsets, each of which is assigned to a different layer of the 3-D architecture. During this task, both fabrication and cost parameters are taken into consideration. More specifically, for a given layer, only technology combatable components can be assigned to, while the layers have to exhibit sufficient area utilization. Then, the layers are appropriately ordered to maximize the performance of derived 3-D stack. This is feasible by assigning to adjacent spatial locations over the z-axis layers with increased interlayer signal activity. The application's partitioning and layer ordering is software-supported by our previously published *TABU* algorithm [43]. Rather than similar approaches, which mainly perform min-cut partitioning [49], our solution provides additional flexibility, since it is aware also about the selected bonding technology (e.g. TSV, Face-to-Face, etc), the desired density of TSVs (per layer) and the shape of 3-D stack (e.g. cube, pyramid, etc). Finally, the derived solutions are evaluated with models for wire-length [45], delay [46] and power consumption [47].

The output from 2nd step provides sufficient information about the application's functionality assigned to each layer, as well as the required connectivity among layers. This information is appropriately handled by our new tool, named *XML2Net*, in order to attach an array of TSVs to every bus that connects architecture's components assigned to different layers. Note that whenever a bus needs to be routed from layer i to layer j, silicon area equals to the area occupied by the TSV array has to be reserved in both layers (we have to preserve that no block will be assigned to TSV's landing area) [37]. Even though our framework can also handle distinct TSVs, throughout this study we select to employ arrays of TSVs because they introduce

fewer constraints to routing algorithm [37]. Then, pairs of TSV array and the block that corresponds to its landing area are connected through special purpose routing paths, named *TSV networks*.

The last step in our framework deals with the system prototyping. More specifically, during this step we perform floor-planning, power and ground network generation, placement of physical library cells, clock tree synthesis and global/detail signal routing with the *Cadence SoC Encounter*. Since the Cadence tools do not support 3-D architectures, we have to make them aware about the additional flexibility imposed by the third dimension through appropriate design encoding. For this purpose we introduce:

- *Virtual layers*: Our framework assumes that target architecture incorporates a number of *virtual layers*, each of which contains hardware resources assigned to different physical layers of the 3-D SoC.
- *TSV networks*: These networks represent routing paths that provide signal connectivity between a TSV array and its corresponding landing area (assigned to adjacent *virtual layers*). Note that during physical implementation, our framework preserves that these two architectural entities are aligned over the z-axis (have the same relative (x, y) co-ordinates at *Virtual layers*). The *TSV networks* are actually implemented through additional metal layers inserted to the technology library file, while their total resistance (R), capacitance (C) and inductance (L) values per unit length are automatically annotated to represent the TSV's RLC parameters [67] [68].

The performance of derived 3-D physical prototype is evaluated by with *Cadence Static Timing Analysis Engine*, while for shake of completeness this analysis is performed both in advanced, as well as after clock tree synthesis and architecture's detailed routing. Then, we verify the functional integrity of physical design by applying a post-layout simulation with *Cadence Incisive Simulator*. For this purpose we extract the delay for all the architecture's routing paths in SDF format (Standard Delay Format) and then our framework automatically annotates the delay values for the *TSV networks*. For the scopes of this letter, the electrical characteristics of *TSV networks* are retrieved from models published at [67] [68]. The evaluation of derived 3-D SoCs can also be performed in term of power consumption by applying a post-layout analysis with *Synopsys PrimeTime PX* tool. The inputs to this analysis are the trace file that contains signal activities in VCD (Value Change Dump) format, as well as the annotated SPEF (Standard Parasitic Exchange Format) file with extracted parasitic values for all the design's resources (logic and interconnect).

3.4.3. Experimental Results

This section depicts how it is possible to employ the proposed framework for designing a 3-D instantiation (with two layers) of a 32-bit LEON3 processor. The LEON3 processor (Fig. 3.47) is a RTL-synthesizable 32-bit processor compliant with the SPARC V8 architecture [69] and configurable through VHDL generics. We instantiated a single-core processor attached as a master to the AMBA Advanced High

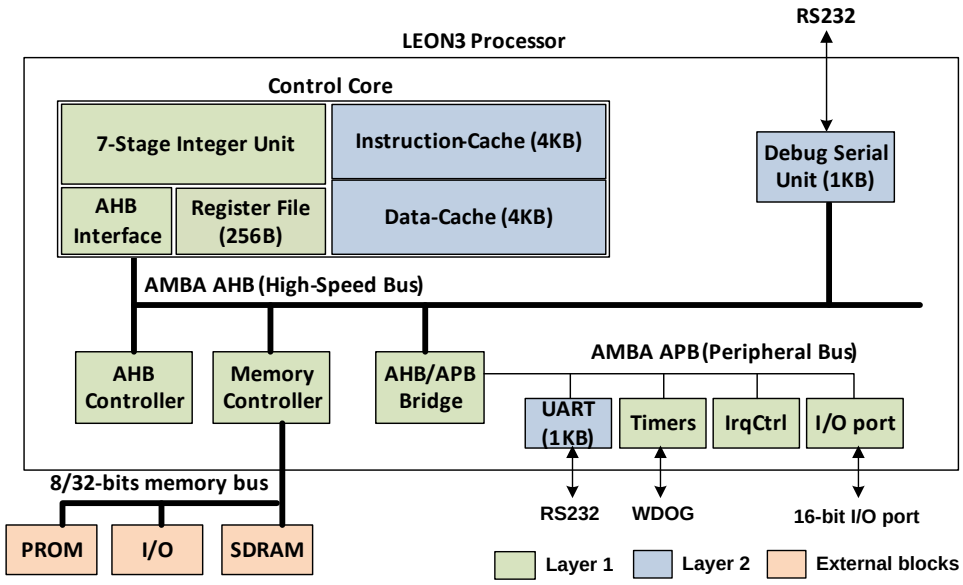


Figure 3.47 Block diagram for the Leon3 processor.

Performance bus (AHB). The processor has 7 pipeline stages, while the internal instruction and data cache include 1 set of 4KB each (Harvard architecture). Since the target architecture is an embedded system, our methodology was tuned to derive a low-power solution. Fig. 3.47 depicts the block diagram of LEON3 processor, as it is retrieved after the 3-D stack generation step. Different colors in this figure denote blocks assigned to different (virtual) layers. Even though additional 3-D stacks can be derived from $TABU$ algorithm, the selected one corresponds to the Pareto optimal solution, i.e. the solution exploiting maximal performance enhancement with the minimal fabrication cost in terms of number of layers and TSVs.

The synthesis of LEON3 processor is performed with *Synopsys Design Compiler* at 130nm CMOS technology under a timing constraint of 4.35ns (or 230MHz). The derived netlist consists of 38,988 standard cells, 42,626 nets, and 110 I/O ports. Fig. 3.48 gives the output from floor-planning, assuming a 3-D device consisted of two layers. In this figure, red and green color dots denote arrays of TSV and their landing blocks assigned to *virtual layer*$_1$ and *layer*$_2$, respectively, whereas the *TSV networks* are depicted with blue color lines. Similarly, red and green color lines correspond to intralayer connections among arrays of TSV and the rest hardware components found in *virtual layer*$_1$ and *layer*$_2$, respectively.

Table 4.5 gives some technical details about the physical implementation of LEON3 processor. Based on these results we can conclude that the derived 3-D architecture reduces total wire-length by 36%, as compared to the corresponding 2-D system implementation. Since throughout this study we focus on designing a low-power instantiation of LEON3 processor, the wire-length reduction is expected to come with considerable power savings, without compromising the performance

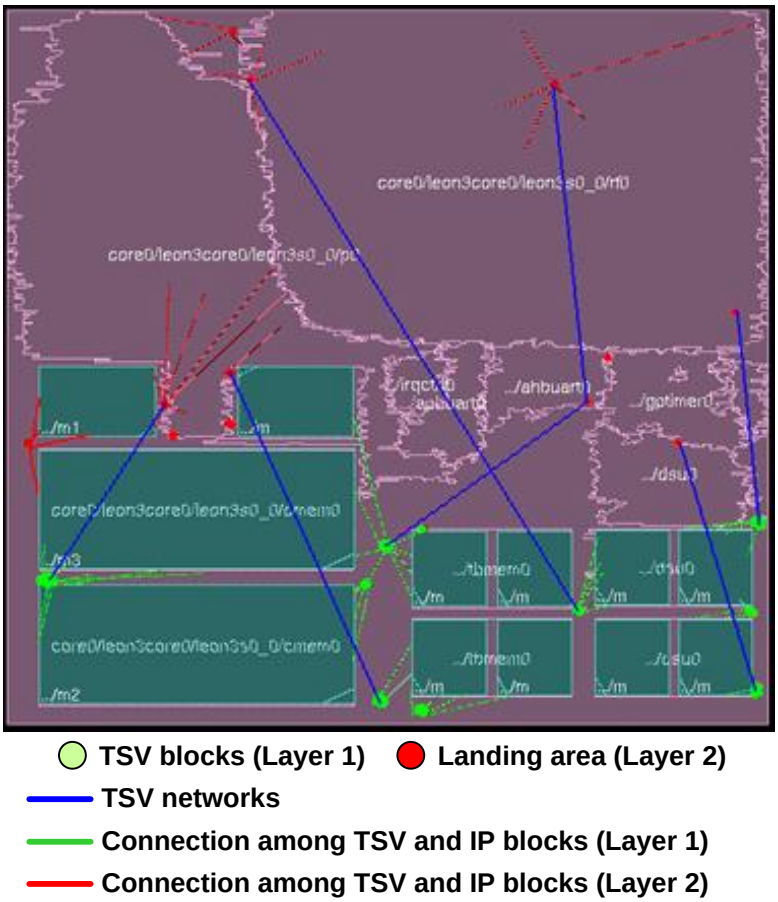


Figure 3.48 Example of designing a 3-D instantiation of LEON3 processor on legacy 2-D Cadence SoC Encounter [24].

Table 3.4 Metrics about the physical implementation.

Characteristics	2-D System	3-D System	
		Layer 1 (Logic)	Layer 2 (Memory)
Wire-length (μm)	855,637	530,443	100,440
Half-perimeter (μm)	823,033	495,010	97,527
Number of TSVs	0.00	817	817
Area for TSVs (μm^2)	0.00	24.7×24.7	24.7×24.7
Aspect ratio	1.00	1.00	1.00
Area per layer (mm^2)	2.89	1.30	1.53
Operation Freq. (MHz)	230	230	
Power consumption (mWatt)	53.96	43.30	

of derived architecture.

For evaluation purposes, the efficiency of derived 3-D architecture is quantified with a number of data intensive benchmarks, which are fundamental kernels in various DSP applications (such as MPEG-4, JPEG, filtering and H.263). In particular, we used five motion estimation algorithms: full search (FS), hierarchical search (HS), three step logarithmic step (3SLOG), parallel hierarchical one-dimensional search (PHODS) and spiral search (SS). It has been noted that their complexity ranged from 60 to 80% of the total complexity of video encoding (MPEG-4) [70]. In addition, we used the 1D wavelet transformation, cavity detector and Fast Fourier transformation (FFT) algorithm. We also incorporated basic benchmarks such as Matrix Multiplication and Bubblesort sorting algorithm.

These benchmarks were implemented in C language and compiled to LEON3 binaries with BCC [71] cross compiler. The binaries were firstly fed to TSIM LEON3 instruction-level simulator [72] for functional verification and resource utilization at host-machine. For instance, Cavity detector algorithm for an input of 64x64 pixels image takes 69 hours simulation time, whereas it requires 58GB of hard disk space for VCD storage and up to 800MB physical RAM, on a host machine with Intel Core2 Duo processor and 4GB RAM. On the other hand this simulation in TSIM environment is completed in less than 3 minutes. Hence, the input data size for each algorithm was appropriately chosen in order to be manipulated by the simulation environment in reasonable time.

Table 3.4 provides also results about the performance and average power dissipation of LEON3 processor. Note that the performance between 2-D and 3-D architectures is constant, since we assumed same throughput. However, the wire-length reduction imposed by the usage of 3-D integration leads to average power savings 20%, as compared to the corresponding 2-D implementation. Table 4.5 provides the simulated time and power savings for every employed application.

3.4.4. Conclusion

A novel framework for supporting rapid evaluation of 3-D SoCs, was introduced. For the scopes of this letter, the proposed methodology was applied to design a

Table 3.5 Simulated time and power consumption under different DSP applications

Benchmark	Clock Cycles	Simulation Time (ms)	Power 2D (mW)	Power 3D (mW)	Gain in 3D (%)
3SLOG	19,748,213	157	59.828	48.405	19.09%
FS	7,368,117	71	43.763	35.260	19.43%
PHODS	13,334,375	120	59.515	48.144	19.11%
SS	7,439,388	72	57.585	46.944	18.48%
CAVITY	23,777,500	216	51.639	42.094	18.48%
HS	9,603,875	93	53.047	42.198	20.45%
MMUL	5,894,285	57	43.763	35.260	19.43%
FFT	8,428,398	91	51.065	40.685	20.33%
WAVELET	6,035,932	56	55.081	43.293	21.40%
BUBBLESORT	5,595,728	46	64.314	50.752	21.09%
AVERAGE:	**10,722,581**	**97.9**	**53.960**	**43.304**	**19.73%**

3-D instantiation of LEON3 processor under low-power constraints. The scope of this study is the development of a methodology and the corresponding design tools to assess the design profit from the adoption of the three-dimensional technology integration. Experimental results with various DSP kernels prove the effectiveness of proposed solution, since it leads to average power savings 19.7% without any performance degradation.

References

[1] T. Ulversoy, *Software defined radio: Challenges and opportunities,* Communications Surveys Tutorials, IEEE **12**, 531 (2010).

[2] J. Glossner, E. Hokenek, and M. Moudgill, *The sandbridge sandblaster communications processor,* in *Software Defined Radio* (John Wiley and Sons, Ltd, 2004) pp. 129–159.

[3] Y. Lin, H. Lee, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, and K. Flautner, *Soda: A low-power architecture for software radio,* in *Computer Architecture, 2006. ISCA '06. 33rd International Symposium on* (2006) pp. 89–101.

[4] A. K. Coskun, T. S. Rosing, K. Mihic, G. De Micheli, and Y. Leblebici, *Analysis and optimization of mpsoc reliability,* Journal of Low Power Electronics **2**, 56 (2006).

[5] V. Gektin, A. Bar-Cohen, and J. Ames, *Coffin-manson fatigue model of underfilled flip-chips,* Components, Packaging, and Manufacturing Technology, Part A, IEEE Transactions on **20**, 317 (1997).

[6] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang, *Thermal vs energy optimization for dvfs-enabled processors in embedded systems,* in *Symp. on Quality Electronic Design (ISQED07), (International Symposium on Quality Electronic Design, 2007. ISQED '07. 8th):pp. 204–209* (2007).

[7] M. Harchol-Balter and A. B. Downey, *Exploiting process lifetime distributions for dynamic load balancing,* ACM Trans. Comput. Syst. **15**, 253 (1997).

[8] J. Yang, X. Zhou, M. Chrobak, Y. Zhang, and L. Jin, *Dynamic thermal management through task scheduling,* in *Performance Analysis of Systems and software, 2008. ISPASS 2008. IEEE International Symposium on* (2008) pp. 191–201.

[9] D. Atienza, P. Del Valle, G. Paci, F. Poletti, L. Benini, G. De Micheli, and J. Mendias, *A fast hw/sw fpga-based thermal emulation framework for multiprocessor system-on-chip,* in *Design Automation Conference, 2006 43rd ACM/IEEE* (2006) pp. 618–623.

[10] M. M. Sabry, J. Ayala, and D. Atienza, *Thermal-aware compilation for system-on-chip processing architectures,* in *Proc. of 20th ACM Great Lakes Symposium on VLSI (GLSVLSI'10)* (2010).

[11] M. Monchiero, R. Canal, and A. Gonzalez, *Power/performance/thermal design-space exploration for multicore architectures,* Parallel and Distributed Systems, IEEE Transactions on **19**, 666 (2008).

[12] J. Rabaey, *Low Power Design Essentials*, 1st ed. (Springer Publishing Company, Incorporated, 2009).

[13] H. Li, P. Liu, Z. Qi, L. Jin, W. Wu, S.-D. Tan, and J. Yang, *Efficient thermal simulation for run-time temperature tracking and management,* in *Computer Design: VLSI in Computers and Processors, 2005. ICCD 2005. Proceedings. 2005 IEEE International Conference on* (2005) pp. 130–133.

[14] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, *Temperature-aware microarchitecture: Extended discussion and results,* in *In Proceedings of the 30th Annual International Symposium on Computer Architecture* (2003) pp. 2–13.

[15] D. Brooks, V. Tiwari, and M. Martonosi, *Wattch: a framework for architectural-level power analysis and optimizations,* in *Computer Architecture, 2000. Proceedings of the 27th International Symposium on* (2000) pp. 83–94.

[16] D. Brooks and M. Martonosi, *Dynamic thermal management for high-performance microprocessors,* in *High-Performance Computer Architecture, 2001. HPCA. The Seventh International Symposium on* (2001) pp. 171–182.

[17] Proteas-Software-Repository, (2013), http://proteas.microlab.ntua.gr/ksiop/software.

[18] Gaisler, *Leon3,* ().

[19] W. Huang, K. Sankaranarayanan, K. Skadron, R. Ribando, and M. Stan, *Accurate, pre-rtl temperature-aware design using a parameterized, geometric thermal model,* Computers, IEEE Transactions on **57**, 1277 (2008).

[20] K. Sankaranarayanan, S. Velusamy, M. Stan, C. L, and K. Skadron, *A case for thermal-aware floorplanning at the microarchitectural level,* Journal of ILP **7** (2005).

[21] A. Raghunathan, N. K. Jha, and S. Dey, *High-Level Power Analysis and Optimization* (Kluwer Academic Publishers, Norwell, MA, USA, 1998).

[22] F. Zanini, D. Atienza, and G. De Micheli, *A control theory approach for thermal balancing of mpsoc,* in *Design Automation Conference, 2009. ASP-DAC 2009. Asia and South Pacific* (2009) pp. 37–42.

[23] S. Inc., *http://www.synopsys.com,* ().

[24] Cadence, *Cadence inc.* .

[25] P. PX, *http://www.europractice.stfc.ac.uk/vendors/primetime_px_brief.pdf,* .

[26] S. Sapatnekar, *Rc interconnect optimization under the elmore delay model,* in *Design Automation, 1994. 31st Conference on* (1994) pp. 387–391.

[27] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, *Mibench: A free, commercially representative embedded benchmark suite,* in *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on* (2001) pp. 3–14.

[28] S. Canumalla and P. Viswanadham, *Portable Consumer Electronics: Packaging, Materials, and Reliability* (PennWell Corporation, 2010).

[29] ITRS, *International technology roadmap for semiconductos,* (2012).

[30] J. Srinivasan, S. Adve, P. Bose, and J. Rivers, *Lifetime reliability: toward an architectural solution,* Micro, IEEE **25**, 70 (2005).

[31] G. De Micheli and L. Benini, *Networks on chips: technology and tools* (Academic Press, 2006).

[32] F. Gebali, H. Elmiligi, and M. W. El-Kharashi, *Networks-on-Chips: Theory and Practice*, 1st ed. (CRC Press, Inc., Boca Raton, FL, USA, 2009).

[33] J. Hu and R. Marculescu, *Energy- and performance-aware mapping for regular noc architectures,* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **24**, 551 (2005).

[34] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, *Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip,* in *Design, Automation and Test in Europe Conference and Exhibition, 2003* (2003) pp. 350–355.

[35] I. Anagnostopoulos, A. Bartzas, and D. Soudris, *Application-specific temperature reduction systematic methodology for 2d and 3d networks-on-chip,* in *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, Lecture Notes in Computer Science, Vol. 5953, edited by J. Monteiro and R. van Leuken (Springer Berlin Heidelberg, 2010) pp. 86–95.

[36] A. Richard, D. Milojevic, F. Robert, A. Bartzas, A. Papanikolaou, K. Siozios, and D. Soudris, *Fast design space exploration environment applied on noc's for 3d-stacked mpsoc's,* in *Architecture of Computing Systems (ARCS), 2010 23rd International Conference on* (2010) pp. 1–6.

[37] V. F. Pavlidis and E. G. Friedman, *Three-dimensional Integrated Circuit Design* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009).

[38] C. J. Glass and L. M. Ni, *Fault-tolerant wormhole routing in meshes without virtual channels,* Parallel and Distributed Systems, IEEE Transactions on **7**, 620 (1996).

[39] V. suite, *http://valgrind.org,* .

[40] S. Gupta, M. Hilbert, S. Hong, and R. Patti, *Techniques for producing 3d ics with high-density interconnect,* in *Proceedings of the 21st International VLSI Multilevel Interconnection Conference* (2004).

[41] S. Murali and G. De Micheli, *Bandwidth-constrained mapping of cores onto noc architectures,* in *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, Vol. 2 (2004) pp. 896–901 Vol.2.

[42] C. S. Compiler, *http://www.cadence.com/products/sd/silicon_compiler/pages/default.asp* .

[43] K. Siozios and D. Soudris, *A tabu-based partitioning and layer assignment algorithm for 3-d fpgas,* Embedded Systems Letters, IEEE **3**, 97 (2011).

[44] K. Siozios, A. Papanikolaou, and D. Soudris, *A method and tool for early design/technology search-space exploration for 3d ics,* in *Proceedings of the IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)* (2008) pp. 359–364.

[45] S. Das, A. Chandrakasan, and R. Reif, *Calibration of rent's rule models for three-dimensional integrated circuits,* Very Large Scale Integration (VLSI) Systems, IEEE Transactions on **12**, 359 (2004).

[46] T. Okamoto and J. Cong, *Buffered steiner tree construction with wire sizing for interconnect layout optimization,* in *Computer-Aided Design, 1996. ICCAD-96. Digest of Technical Papers., 1996 IEEE/ACM International Conference on* (1996) pp. 44–49.

[47] J. M. Rabaey, A. P. Chandrakasan, and B. Nikolic, *Digital integrated circuits*, Vol. 2 (Prentice hall Englewood Cliffs, 2002).

[48] D. Velenis, M. Stucchi, E. Marinissen, B. Swinnen, and E. Beyne, *Impact of 3d design choices on manufacturing cost,* in *3D System Integration, 2009. 3DIC 2009. IEEE International Conference on* (2009) pp. 1–5.

[49] N. Selvakkumaran and G. Karypis, *Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization,* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **25**, 504 (2006).

[50] V.-D. Ngo, H.-N. Nguyen, and H.-W. Choi, *The optimum network on chip architectures for video object plane decoder design,* in *Parallel and Distributed Processing and Applications*, Lecture Notes in Computer Science, Vol. 4330, edited by M. Guo, L. Yang, B. Di Martino, H. Zima, J. Dongarra, and F. Tang (Springer Berlin Heidelberg, 2006) pp. 75–85.

[51] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, *Noc synthesis flow for customized domain specific multiprocessor systems-on-chip,* Parallel and Distributed Systems, IEEE Transactions on **16**, 113 (2005).

[52] I. E. Richardson, *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia* (John Wiley & Sons, 2004).

[53] M. Di Marco, A. Garulli, S. Lacroix, and A. Vicino, *Set membership localization and mapping for autonomous navigation,* International Journal of robust and nonlinear control **11**, 709 (2001).

[54] *Esa nasa exomars programme,* .

[55] Virtex-6-FPGA-Family., *www.xilinx.com/products/silicon-devices/fpga/virtex-6.html,* .

[56] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, *Ros: an open-source robot operating system,* ICRA workshop on open source software **3**, 5 (2009).

[57] K. Siozios, D. Diamantopoulos, I. Kostavelis, E. Boukas, L. Nalpantidis, D. Soudris, A. Gasteratos, M. Aviles, and I. Anagnostopoulos, *Spartan project: Efficient implementation of computer vision algorithms onto reconfigurable platform targeting to space applications,* in *Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), 2011 6th International Workshop on* (2011) pp. 1–9.

[58] R. lab Duth., *http://robotics.pme.duth.gr,* .

[59] L. Nalpantidis, G. Sirakoulis, and A. Gasteratos, *A dense stereo correspondence algorithm for hardware implementation with enhanced disparity selection,* in *Artificial Intelligence: Theories, Models and Applications*, Lecture Notes in Computer Science, Vol. 5138, edited by J. Darzentas, G. Vouros, S. Vosinakis, and A. Arnellos (Springer Berlin Heidelberg, 2008) pp. 365–370.

[60] H. Bay, T. Tuytelaars, and L. Van Gool, *Surf: Speeded up robust features,* in *Computer Vision – ECCV 2006*, Lecture Notes in Computer Science, Vol. 3951, edited by A. Leonardis, H. Bischof, and A. Pinz (Springer Berlin Heidelberg, 2006) pp. 404–417.

[61] L. Nalpantidis, G. C. Sirakoulis, and A. Gasteratos, *Non-probabilistic cellular automata-enhanced stereo vision simultaneous localization and mapping,* Measurement Science and Technology **22**, 114027 (2011).

[62] D. F. Bacon, S. L. Graham, and O. J. Sharp, *Compiler transformations for high-performance computing,* ACM Comput. Surv. **26**, 345 (1994).

[63] S. Cabay and L. Jackson, *A polynomial extrapolation method for finding limits and antilimits of vector sequences,* SIAM Journal on Numerical Analysis **13**, 734 (1976).

[64] ROS, *The robot operating system,* .

[65] I. Kostavelis, L. Nalpantidis, E. Boukas, M. A. Rodrigalvarez, I. Stamoulias, G. Lentaris, D. Diamantopoulos, K. Siozios, D. Soudris, and A. Gasteratos, *Spartan: Developing a vision system for future autonomous space exploration robots,* Journal of Field Robotics **31**, 107 (2014).

[66] R3LOGIC, *R3logic inc.* (2013).

[67] 3d-performance Lancaster University, *Tools for design space exploration of 3-d integrated circuits,* Http://3d-performance.lancs.ac.uk.

[68] A. Jantsch, M. Grange, and D. Pamunuwa, *The promises and limitations of 3-d integration,* in *3D Integration for NoC-based SoC Architectures* (Springer, 2011) pp. 27–44.

[69] S. M. Inc., *Sparc v8 architecture,* ().

[70] V. Bhaskaran and K. Konstantinides, *Image and Video Compression Standards: Algorithms and Architectures*, 2nd ed. (Kluwer Academic Publishers, Norwell, MA, USA, 1997).

[71] Gaisler, *The bcc cross-compiler,* ().

[72] Gaisler, *Tsim simulator,* ().

4

Computer-Aided Design Tools for Reconfigurable Platforms

This chapter presents the proposed design tools for emerging reconfigurable platforms. It introduces a characterization and classification technique for clustering the input applications according to their intrinsic characteristics, during the phase of the synthesis. Also, a novel multi-objective genetic algorithm is proposed for the FPGA placement problem. The algorithm utilizes the aforementioned classification information in order to explore effectively the search space and thus, provide placement solutions that trade-off design metrics on the Pareto front. Chapter 4 is associated with the general methodology of the thesis presented in Section 1.2, as to its contribution to the category "Reconfigurable Platforms - EDA Tools FPGAs".

4.1. Introduction

Over the past decade reconfigurable architectures, and more specifically Field-Programmable Gate Arrays (FPGAs), have revolutionized the way that digital systems are designed and built due to their inherent re-programmability feature. In addition, modern FPGAs have become efficient alternatives to Application-Specific Integrated Circuits (ASICs). For instance, the recent Xilinx Virtex-7 devices designed at 28nm contain almost 2 million logic cells.

Even though the previously mentioned architectural selections meet the requirements of logic-hungry FPGA designers, they further complicate the work allocated to the FPGA CAD tools [1]. Although, a number of EDA (Electronic Design Automation) tools that provide automated application implementation onto hardware platforms are now available, their execution still imposes an increased run-time overhead. This problem becomes more evident taking into account that the capacity, in term of logic resources, steadily increases at the rate anticipated by Moore's Law. Hence, the EDA tools must synthesize, place and route more logic blocks and interconnection networks for every new platform. However, given the increasing

complexity of applications mapped onto FPGAs, it is expected that physical design tools, i.e. placement and routing (P&R), will be extensively stressed to deliver highly optimized solutions within practical run-time budgets.

This occurs mainly due to the fact that processor performance improvements have not been tracking FPGA capacity growth since the mid-2000s, as shown in Fig. 4.1.

Figure 4.1 CPU speed versus FPGA logic capacity [2].

Application implementation onto FPGA platforms can take hours, or even days, depending on the complexity of these designs. One of the most time-consuming steps in the FPGA CAD flow is application's placement. A good quality placement is essential to the overall designs quality, since it influences among others the interconnect delay, the congestion, the wirelength, as well as the power consumption [3]. While there exists a lot of previous research on placement algorithms for improving application's maximum operating frequency, power/energy dissipation and the wiring area occupied by a circuit, very few of them have as their primary goal the minimization of the tool execution run-time. Compile time has recently been recognized as an important issue for FPGAs [4], whereas there are designers that are willing to afford a reduction in the quality of results (e.g. a penalty in performance) in exchange for a high-speed compilation [3]. Moreover, as the capacity of FPGA devices and the size of designs grow, there is a great interest for performing fast application's implementations onto reconfigurable platforms.

To keep run-time in check, the two main companies offering high-capacity FPGAS, Xilinx and Altera, have been continuously optimizing their CAD tools. Even though such a selection alleviated the run-time pressure, it is unlikely that such algorithm engineering efforts can be sustained at the rate required by several more

generations of Moore's Law. Continuous technology scaling without comparable scaling of execution run-time for application implementation onto FPGA devices is expected to lead to a run-time crisis. This crisis among others manifests itself as a reduction in productivity and the corresponding increase at the engineering costs. Based on relevant research approaches, there are three ways to reduce the execution run-time of CAD tools, which can be classified as follows:

- Discourage flat compilation of the entire design, and instead force users to compile partitions of their designs incrementally and assemble the partitions. Even though this approach mitigates execution run-time, it imposes an increased design complexity, whereas it also does not allow optimizations to be applied between partitions.

- Pay effort to find faster single-threaded algorithms, which can achieve mentionable execution speedup with a little, or no, sacrificing at quality of derived application implementation [3] [5]. This selection leads to mentionable speedups, however, it is not widely accepted as it cannot follow the exponential growth in FPGA logic cell counts.

- Develop novel parallel algorithms, to take advantage of the existing and upcoming multi-core processors [6] [7] [8] [9] [10] [11] [12] [13]. With the current market trend of increasing the number of CPU cores rather than designing faster CPU cores [1], the usage of parallel CAD algorithms promises to alleviate the run-time crisis. These algorithms allow the capacity of FPGA platforms, as well as the number of working processor cores, to scale simultaneously. Towards this direction, both Xilinx and Altera have started to implement parallel flavors for their CAD algorithms that offer mentionable execution speedups. This is also the approach studied throughout this manuscript.

The functionality of the majority placers is based on simulated annealing [14] [6] [7] [11] [6] [10] [7]. These approaches in order to derive the best possible quality, sequentially perform swaps of random logic block locations, accepting non-greedy moves with exponentially decreasing probability based on the current temperature of the anneal and the *delta* that would occur in the overall cost function [15]. The temperature at simulated annealing is reduced following a cooling schedule. At the outset of this cooling schedule, virtually all moves are accepted. On the other hand, as the end of the cooling schedule is reached, only moves that would improve the overall cost function are accepted. Previous works (e.g. [14] [16]) shown that simulated annealing placers produce promising results if appropriate cost functions and sufficiently slow cooling are employed.

Throughout this research work we introduce a novel placer based on genetic algorithm. The main contributions of this work can be summarized as follows:

- We propose a new parallel genetic algorithm (GA) for application placement onto FPGA devices. The proposed algorithm utilizes the concept of mixability in order to improve the quality of the derived placement solutions.

- We analyze the internal structures (elitism-based selection, mixability-driven mutation, path-driven crossover, self-adaptive mutation) defined for tailoring the genetic algorithm to the application placement problem.
- We present a coarse-grain yet efficient OpenMP parallel implementation of the proposed genetic algorithm, by identifying interdependent tasks across the GA operators and by defining the thread-oriented heap memory structure.
- We propose a software-supported exploration methodology for tuning the parameters of the GA engine according to the input application's features.
- We provide an extensive experimental evaluation regarding the efficiency and the scalability of the proposed framework in comparison to existing state-of-the-art placement tool.

More specifically, our placer performs a more effective search space exploration, whereas its inherent parallelism is exploited by the underlying multi-core architectures for reducing the execution run-time. The introduced approach supports either fast application placement (with significant lower run-time overhead), or provides superior applications placements (e.g. with higher operating frequency, lower power consumption, etc) at the comparable execution run-time. More specifically, in the first case, our solution leads on average to 67× faster execution, whereas the average increase in term of maximum operating frequency in the latter approach is 16%. As a reference to our study, we employ the state-of-the-art academic placer for FPGAs [14] [15], as well as our previous work introduced in [17].

The rest of this chapter is organized as follows: Section 4.2 describes the related work, whereas section 4.3 introduces the proposed framework for application implementation onto reconfigurable architectures with the usage of a genetic algorithm. Since the efficiency of this algorithm highly depends on the customization of its own parameters, Section 4.4 describes the employed methodology for tuning the parameters of evolutionary placement. Quantitative results that prove the effectiveness of the introduced framework against to state-of-the-art approaches is presented in Section 4.5, while conclusions are summarized in Section 4.6.

4.2. Related Work

Over the last decade researchers have developed methodologies, algorithms and tools aiming to accelerate the task of application implementation onto FPGAs with the usage of parallel placers. Table 4.1 summarizes a representative number of these approaches. The second column of this table reviews if their selections during application implementation are performed having as goal to improve the maximum operating frequency. The last two columns give the underlying hardware for their execution and the achieved speedup, respectively. Next, we discuss in more detail some of the most representative solutions found in relevant literature.

A parallel timing-driven algorithm based on simulated annealing for Quartus-II framework is discussed in [9] [19]. Even though this approach evaluates numerous

Table 4.1 Overview of existing parallel placements.

Reference	Timing-Driven	Hardware Targeted	Speedup
[6]	No	Sequent Balance 8,000 (8-proc.)	6.4× on 8 processors
[10]	No	VAX 11/784 (4-processors)	2.3× on 4 processors
[8]	No	6 Nat.Sem. 32,016 processors	4× with 5 processors
[12]	No	Hypercube multiprocessors	8× with 16 processors
[18]	No	Networks of machines	3.3× on 16 processors
[11]	No	FPGAs	500×-2,500× over CPUs
[9]	Yes	Multiprocessors	2.1× on 4 processors
[13]	No	MPPAs	1/256 less swaps with 1,024 cores
[7]	No	GPU	10× on NVIDIA GTX280
[19]	Yes	Multiprocessors	2.4× on 8 processors
[2]	Yes	Multiprocessors	161× using 25 processors

moves in parallel, these moves are serially committed in order to achieve a serially equivalent placement. Authors claim that such an approach achieves a speedup ranging between 2.1× and 2.4×, for 4 and 8 processing cores, respectively, whereas the quality of derived application placement is comparable to the corresponding one retrieved with the usage of a serial version. A distributed annealing algorithm targeting to a systolic architecture is discussed in [11]. The functionality of this algorithm relays on restricting the swap range of each block to its 4 immediate neighbors. Even though such a selection is easily parallelizable, the lack of opportunity to apply a more aggressive scenario during placement leads to 36% quality degradation in the final circuit. An extension of work discussed in [11], showed that there is a quality improvement of 5% compared to simulated annealing approach, whereas it still offers mentionable speedups.

A bottom-up netlist clustering aiming to reduce the problem's size, so that a smaller and more easily solvable placement to be applied is obtained in [20]. A similar approach is followed in [21], where a min-cut algorithm divides the placement of the overall netlist to smaller placement problems, so that these problems can be solved in parallel. However the derived placement quality is degraded. A fast timing-driven placement for reconfigurable platforms based on a macro-floorplanner, is discussed in [4]. Even though the author reports an improvement in execution run-time compared to relevant approaches, the employed benchmarks are quite small. Hence, non-valuable conclusions about the efficiency are extracted. Authors in [22] propose an algorithm for application placement onto FPGAs, which trades-off the quality of derived solutions (in terms of total wirelength) and the execution run-time. Another algorithm involves the usage of Tabu search in order to speedup the execution run-time of placement algorithm [5]. Authors in [2] presented a parallel timing-driven algorithm which performs high and scalable speedup compared to VPR [14], but comes with mentionable loss of quality (both timing and wirelength). An alternative approach assumes the incorporation of dedicated hardware for accelerating the simulating annealing problem [11]. The idea of this work is to introduce parallelism to the placer, but the specialized hardware makes it prac-

tically non-applicable for real designs. Finally, authors at [23] propose the usage of genetic algorithm at VPR placer. However, the experimental results provided in this work, showed that such an implementation does not outperform placements retrieved with conventional (e.g. simulated annealing) algorithm.

4.3. The Proposed Design Framework

The proposed framework for performing application placement and routing (P&R) onto reconfigurable platforms is depicted in Fig. 4.2. This framework is similar to existing academic and commercial flows (e.g. [14]), whereas its inherent modularity allows to easily adapt tools with the same functionality between alternative flows. More specifically, throughout this work we replaced the existing placer found in NAROUTO framework [24] with the proposed algorithm, named *GENESIS*.

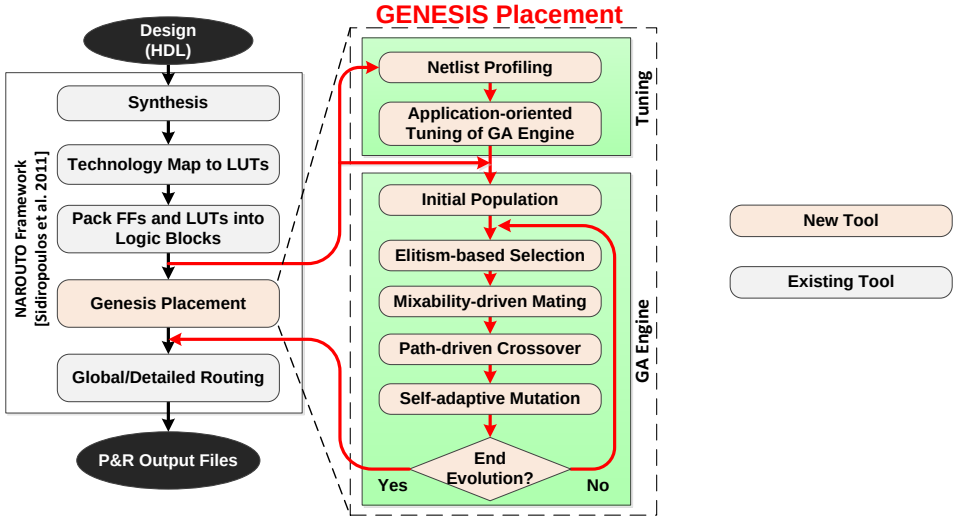


Figure 4.2 Proposed framework for parallel application placement.

Synthesis is the first step in our CAD tool flow. During this step, the application's description in HDL (Hardware Description Language) is converted into gate-level logic functions and flip-flops. Next step involves the technology mapping aiming to translate the nelist derived from synthesis step, into a nelist consisted of K-input LUTs and flip-flops. Technology mapping algorithms strive to minimize specific objectives, such as area, delay, power or any combination of them. In this work the task of technology mapping is performed with the SIS optimizer [25].

Then, we cluster the derived nelist in order to pack the LUTs and flip-flops created in technology mapping, abiding to the constraint that no more than N LUTs, or N flip-flops, can be packed to form each cluster. This output of clustering task is a set of BLEs (Basic Logic Elements). Although structurally similar, different vendors have adopted their own terminology to describe a cluster. For instance, Xilinx uses the term CLB (Configurable Logic Block) and Altera uses the term LAB (Logic

Array Block) to describe a cluster. Different optimization goals could be taken into consideration during the clustering task. For instance, the minimization of delay [15] and power consumption [26], are typical approaches. These optimizations requires the understanding that BLEs contained within the same cluster communicate through intra-cluster connections, which are faster than inter-cluster connections, also known as the routing network. Hence, the size of cluster, mentioned as N, highly affects the efficiency of application's implementation. A larger N value allows more BLEs to share intra-cluster connections. However, careful tradeoff analysis must be considered as the intra-cluster connection latency degrades with increased cluster size. In this work we assume that our target FPGA consists of 4 BLEs per cluster (i.e. N=4), while the task of netlist clustering is performed with the timing-driven Versatile Packing (T-VPACK) tool [14].

The placement step involves assigning each CLB in the circuit to a unique physical location on the FPGA chip. This algorithm is realized with the proposed genetic algorithm, named *GENESIS*. Rather than relevant approaches which are executed sequentially, the new placer exhibits inherent parallelism, which can benefit from multi-core processors. Moreover, the proposed solution supports automated tuning of the algorithm's parameters depending on the inherent features posed by the target application, for further performance improvement. The *GENESIS* algorithm provides application placement that aims to minimize a combined objective that takes into consideration the Half-perimeter Wirelength (HPWL), also known as the bounding-box cost, the application's delay and its power consumption. Decisions made by the placer are highly influential to the overall solution quality, since CLBs are locked in the placed locations after this stage. Additional details about the introduced algorithm are provided in upcoming sections.

The last step in the proposed CAD tool flow is routing that finds the suitable paths for all of the interconnect signals utilizing the programmable routing switches of the FPGA. The employed routing is performed with the usage of VPR's routing tool, which is a single-step router based on the PathFinder algorithm [15]. Finally, the placed and routed application's netlist is evaluated by performing timing closure and extract timing and power metrics that quantify the design quality.

4.3.1. GENESIS Evolutionary Placement

This section describes in more detail the employed placer, named GENESIS. The placement task results in a physical assignment of all logic blocks and I/O pads on the target FPGA, which minimizes one or more objectives found in the cost function (wirelength, speed, power dissipation). Since the placement is an NP-hard combinatorial optimization problem, there is no algorithm that can provide an optimal solution in polynomial-time. Thus, many heuristics have been proposed, in an attempt to obtain near-optimal solutions in a reasonable amount of time. We adopt the usage of genetic algorithms (GA), which were already applied to numerous optimization problems. GAs incorporate a random, yet directed, search for locating the global optimal solution. Since the search is not biased towards the local optimal solution, these algorithms are superior to gradient descent techniques. In contrast to random sampling algorithms, which mainly pay effort to retrieve a valid solution

through random selections, GA can direct the search towards relatively prospective regions in the search space [27], leading among others to significant lower execution run-time without affecting the quality of derived solutions. Furthermore, we show in Subsection 4.3.2 that the GA can be efficiently parallelized, thus being possible to take advantage of modern multi-core CPUs in order to improve performance compared to the state-of-the-art academic tool VPR 4.30 [14].

Genetic algorithms are motivated by the theory of evolution. They have been designed as general search strategies and optimization methods working on populations of feasible solutions. A genetic algorithm aims at producing near-optimal solutions by letting a set of strings [1], representing random solutions, undergo a sequence of transformations governed by a selection scheme biased towards high-quality solutions. An index of merit (fitness value) is assigned to each individual chromosome, according to a defined fitness function. Usually the fitness value is the value of the objective function or some scaled version of it. A new generation is evolved by a selection technique, in which there is a larger probability of the fittest individuals being chosen. The selected chromosomes are used as parents for the construction of the next generation, while new generation is derived as a result of reproduction operators applied on parents.

Our solution adopts the concept of the mixability theory regarding the role of sex in evolution, proposed by [28]. According to this theory, the evolvability [2] of a population can be enhanced by the ability of alleles [3] to perform well across different combinations. Authors explore how it is possible for sex to increase the population mean fitness while, at the same time, may break down highly favorable combinations of genes, which impedes the increase in fitness. As shown, "the ideal module is also a *good mixer* in the sense that it is transferred as a whole by recombination and is therefore more likely to maintain its individual contribution to fitness across different backgrounds" [28]. We apply the concept of mixability at the construction of GENESIS operators *crossover* (subsection 4.3.1.3) and *mating* (subsection 4.3.1.2).

To present a motivation example of our approach we use the following example: Let the architectural parameters of the search space in GA to be encoded in the form of a chromosome-like structure, as it is depicted in Fig. 4.3. More specifically, each chromosome consists of two parts for encoding the architecture's CLBs and I/Os, named the *CLB locus* and *IO locus*, respectively, whereas the assignment of netlist's CLBs and I/Os to a *locus* is defined as *allele*. The physical locations both of CLB's and IO's on the FPGA are indexed through ascending order numbering sequences (as it is depicted in the left-bottom part of Fig. 4.3).

Each distinct solution (placement) of the GA is represented with an *individual* ($Ind(1), Ind(2), \ldots, Ind(N)$), whereas a group of *individuals* constitutes a *population*. In order to quantify the efficiency of these *individuals* we incorporate a fitness

[1] Each string represents a chromosome.

[2] Evolvability refers to the adaptation of a population's genetic operator set over time. It is the ability of a population to generate adaptive genetic diversity, and thereby evolve through natural selection.

[3] Each allele represents a specified set of alternative values for each gene. Each chromosome consists of a large number of genes, each uniquely located on the chromosome.

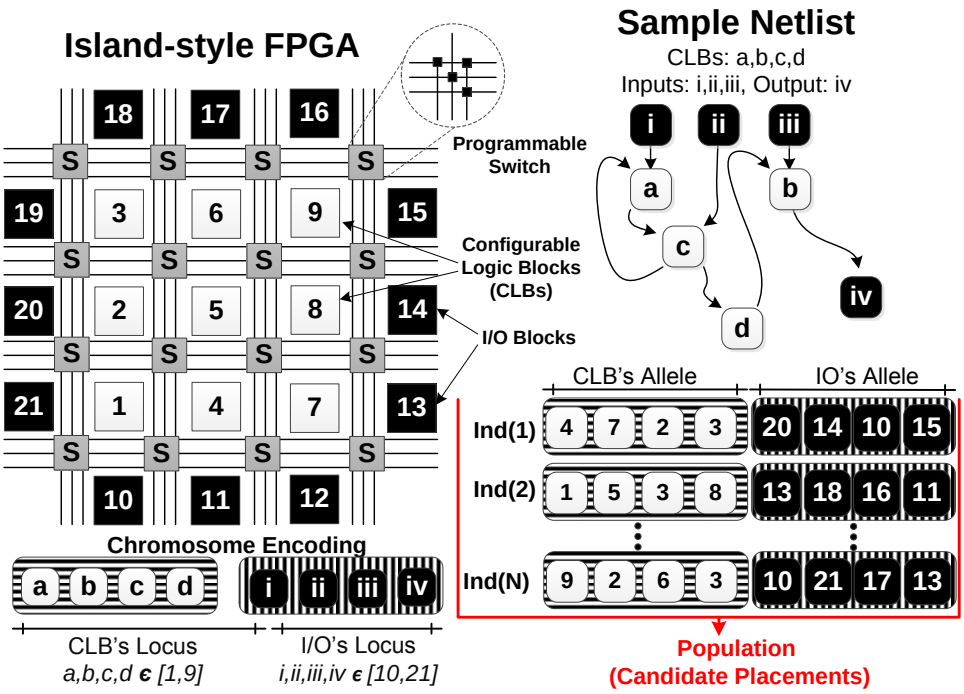


Figure 4.3 The employed representation of FPGA with a chromosome structure and an example of netlist encoding.

function. For the scopes of this research, the GA is tuned to derive placements that optimize timing and wiring cost. Equation 1 gives the fitness value for *individual* i.

$$Fitness(i) = \alpha \times T_{cost}(i) + (1 - \alpha) \times W_{cost}(i) \tag{4.1}$$

where T_{cost} denotes summation to delay cost over all paths of the application's netlist and the W_{cost} corresponds to the total wirelength over these paths. Regarding the timing cost, it is computed based on the Elmore delay model, whereas the wiring cost is retrieved from the Half-Perimeter WireLength (HPWL) model [14] [15]. The α parameter is used to weight the optimization importance of each of the two costs (timing, wiring). Since the parameter α affects the Quality of Report (QoR) we explore its effect on the employed benchmarks through a heuristic exploration analysis (4.4.2). The fittest solution has also the lowest fitness value, since the metrics of delay cost and wiring cost are decreasing while QoR is improved.

Given the previous GA representation, we create a population of FPGA placements (individuals) for the MCNC *alu4* benchmark (14 inputs, 8 outputs, 1,519 CLBs). The individual i is identified by the genotype CLB_jIO_k, which represents the physical locations of CLBs and IOs in the FPGA array, in which the respective CLBs and IOs of *alu4* netlist have been placed to. The combination of six different CLB placements ($j = 6$) and six different IO placements ($k = 6$) formulates n=36 distinct

solutions, which are depicted in Fig. 4.4. While genotype CLB_1IO_1 has maximum fitness, there is a sense in which allele CLB_6 performs best overall among the CLB alleles across different genetic contexts, a metric which called *mixability*. Authors in [28] have shown that evolvability improved when alleles of the same gene compete with each other based on how well they perform on average (i.e. CLB_6) rather how well they perform in any one specific combination (i.e. CLB_1IO_1). This conclusion motivate us to construct reproduction operators which result in placements with enhanced QoR (low fitness) on average, rather than on specific combination.

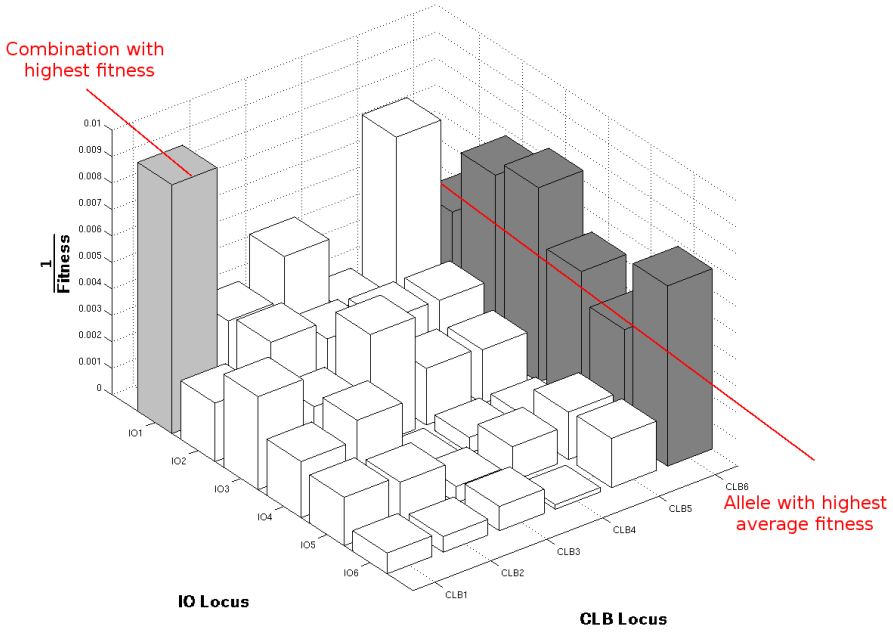


Figure 4.4 The *alu4* fitness landscape with six alleles per locus. Alleles $CLB_{1:6}$ in locus CLB and $IO_{1:6}$ in locus IO give 36 genotypes CLB_jIO_k with fitness $f_{j,k}$, represented by the height of the bars.

However the problem that arises in such an approach, is the level of granularity that classifies these combinations. In case of timing-driven placement, the goal is to assign netlist's CLBs/IOs to physical FPGA CLBs/IOs so that the critical path[4] is minimized, leading to the maximum operating frequency of the design. Every path of a netlist is a connection of edges on the netlist's graph representation node, so that information can flow from a source to a sink node during one cycle. At the FPGA case, sources are the input pins and the flip-flop output pins while sinks are

[4]The critical path of a design refers to the delay of its longest path and determines its clock period. The critical path is determined by the number of levels of logic, the internal cell delay, the wire delay, the cell input capacitance, the cell fanout and the cell output drive strength.

the output pins and the flip-flop input pins. Fig. 4.5 depicts a design test case in order to present these definitions.

During HDL synthesis, several optimization techniques are performed so that an HDL description can be efficiently mapped onto physical FPGA resources. While HDL compilation results in fine-grain netlist representation (generic gate and flip-flop models), as depicted in Fig. 4.5(b), the last synthesis steps deal with the packing of this fine-grain functionality to the FPGA primitive cells, i.e. the slices. As described in Section 4.3 we use T-VPack tool in order to pack LUTs and registers to a netlist of slices. The goal of this task is to maximize the utilization of CLBs, while minimizing the number of inter-cluster connections on the critical path. This packing procedure creates dependencies among slices belonging to different paths, but still have been packed to the same CLB in order to satisfy the capacity requirement of T-VPack tool. Regarding the example depicted in Fig. 4.5(b), in case that $FF1, FF2, LUT3$ and $LUT4$ are packed into the same CLB ($CLB2$), then the placement of $CLB2$ affects both the wirelength of $Path2$ and $Path3$.

While existing placement algorithms seek to find optimal (i.e. with minimum critical path) placements for CLBs, throughout this research we propose the usage of reproducing operators in GA which take into account the distribution of CLBs and IOs within paths, given a packing solution. To support such an approach we firstly define the metric $dependability$, denoted by symbol ξ, as follows:

Definition 1 *Given a design netlist with n-nodes (CLBs, IOs) on timing graph (e.g. Fig. 4.5(c)), k-edges (nets between two nodes) and m-paths ($m <= k$), so that for every path $p_{i=1:m}$ there is a set S_{p_i} containing all the nodes belonging to path p_i, then for every node N_j the metric dependability, ξ_j, is defined as:*

$$\xi_j = \sum_{path\ i=1}^{m} f(i,j),\ f(i,j) = \begin{cases} 0, when\ node\ j\ \in\ S_{p_i} \\ 1, when\ node\ j\ \notin\ S_{p_i} \end{cases} \tag{4.2}$$

The metric ξ_j for every node j represents a quantitative value of the placement importance for this node. The proposed placement algorithm takes into consideration the $dependability$ metric in order to converge quickly towards global optimum. We analyze the use of this metric throughout the construction of transformations steps of the GA during evolution.

Another metric we use in the construction of the proposed transformations steps is the average sub-fitness value of paths, $\bar{f_p}$. While the fitness metric f quantifies a single placement of entire netlist, the path sub-fitnsess f_{p_i} quantifies the placement of the single path i, based on the sum of the bounding box of its nets.

Definition 2 *Given an FPGA architecture with l-physical CLBs and IOs, a design netlist with n-nodes (CLBs, IOs) on timing graph (e.g. Fig. 4.5(c)), k-edges (nets between two nodes) and m-paths ($m <= k$), so that for every path $p_{i=1:m}$ there is a set R_{p_i} containing all the edges belonging to path p_i, and a placement $\Pi(\hat{n} \Rightarrow \hat{l})$ the metric average paths sub-fitness value, $\bar{f_{p_\Pi}}$, for placement Π, is defined as:*

$$\bar{f_{p_\Pi}} = \sum_{path}^{m} \sum_{p_{i=1}} f_{p_i \Pi}, \tag{4.3}$$

where for each path p_i the metric path sub-fitness value, f_{p_Π}, for placement Π, is defined as:

$$f_{p_i \Pi} = \sum_{edge}^{R_{p_i}} \sum_{e_{j=1}} \left[bb_x(j)) + bb_y(j) \right], \tag{4.4}$$

where for each net j, bbx(j) and bby(j) denote the horizontal and vertical spans of its bounding box, respectively.

We also define the $diversity$ metric to quantify the similarity between two placements. This metric is used by the mutation operator (4.3.1.4) and it is defined as follows:

Definition 3 *Given an FPGA architecture with l-physical CLBs and IOs, a design netlist with n-nodes (CLBs, IOs) on timing graph (e.g. Figure 4.5(c)) and j placements $\Pi_{\{1,j\}}(\hat{n} \Rightarrow \hat{l})$ then for every node N_j the metric diversity, $\Delta_{\Pi_{\{1,j\}}}$, is defined as:*

$$\Delta_{\Pi_{\{1,j\}}} = \sum_{node}^{n} \sum_{i=1} g(i,j),\ g(i,j) = \begin{cases} 0, when\ \Pi_1[node_i] = \Pi_2[node_i] = \cdots = \Pi_j[node_i] \\ 1, when\ \Pi_1[node_i] \neq \Pi_2[node_i] \neq \cdots \neq \Pi_j[node_i] \end{cases} \tag{4.5}$$

The transformations on the individuals of a population constitute the recombination steps of a GA. These transformations are performed by four operators in a way that implicitly good properties are identified and combined into a new population. This new population (also called offspring) hopefully has the property that the fitness value of the best individual (representing the best solution in the population) and the average fitness value of the individuals are better than in previous populations. The process is then repeated until some stopping criteria are met. The four basic operators of a genetic algorithm when a new population is constructed are selection, mating, crossover and mutation. In subsections 4.3.1.1, 4.3.1.2, 4.3.1.3 and 4.3.1.4 we describe in more detail these operators. Table 4.2 summarizes the symbols and definitions used for the rest section.

4.3.1.1 Selection

Through selection a new temporary population is generated, where each member is a replica of a member of the old population. The individuals are sorted based on their fitness value. According to a predefined probability, a percentage of the population will remain intact in the next population, a technique referred as elitism. The intended effect of this operation is to improve the quality of the population as a whole. However, no genuinely new solutions and hence no new information is

Figure 4.5 (a) Baseline FPGA architecture. (b) HDL-to-CLBs Synthesis. (c) Graph representation of a design circuit. (d) FPGA physical placement.

created in the process. The generation of such new placements is handled by the crossover operator, explained in subsection 4.3.1.3. The elitism percentage p_e is an important parameter of the quality of placements, as highlighted by Figure 4.6. While no-elitism (0%), or high elitism rate (75%) express high fitness, as expected, it seems that the elitism percentage value highly affects the algorithm's efficiency, i.e. using p_e=25% the GENESIS algorithm, applied to MCNC circuit *s38417*, converge to a local optimal at generation 219, while at the same generation, the algorithm configured with p_e=50% provides a solution with higher (worse) fitness metric of 16%. The algorithm is described in pseudo-code in Algorithm 1.

4.3.1.2 Mating

The mating operator selects two individuals in order to create a new one, whereas this task is repeated until all the individuals have been mated. Different approaches are possible to be employed during this step. The common way of mating parents consists of taking a parent from the mating pool and selecting its mate by choosing randomly one of the remaining parents. The mated parents are then removed from the mating pool. Restricted mating techniques, such as fitness sharing [29], evolving agents [30], tabu genetic algorithm [31] and seduction [32], which do not select a mate uniformly at random, have been proven to improve the performance

Table 4.2 Parameters of the proposed GA.

Symbol	Description
$P(t)$	Population: The set of total placements at time t of evolution.
n_p	Population size: The total number of placements.
g_p	Population age: The total number of GA generations.
$E(t)$	Elitism pool: The subset of $P(t)$ of the fittest individuals during evolution.
p_e	Elitism percentage: The percentage of the population constitutes subset E, $p_e \times n_p = size(E)$.
$M(t)$	Mating pool: The subset of $P(t)$ of individuals that form pairs for new offsprings.
p_{mat}	Mating percentage: The percentage of the population constitutes subset M, $p_{mat} \times n_p = size(M)$.
$C(t)$	Crossover pool: The set of $M(t)$ after applying crossover operator. $size(M) = size(C)$
b_m	Mutation probability: The probability applying mutation to an individual i of population $P(t)$.

(t denotes the time step of evolution, i.e. the i^{th} generation of GA)

ALGORITHM 1: Elitism-based Selection

Input: Population on time t: $P(t)$
Input: Elitism percentage p_e
Output: 1)Fitness based Sorted Population on time t: $P_s(t)$
Output: 2)Elitism Pool on time $t+1$: $E(t+1)$

```
1 initialization;
2 for i = 1 to n_p do
3 |   f(i)=ComputeFitness(Individual_i);
4 end
5 P_s=SortPopulation(P, *f);
6 for j = 1 to (p_e × n_p) do
7 |   E_j(t+1) ← P_s(Individual_j);
8 end
```

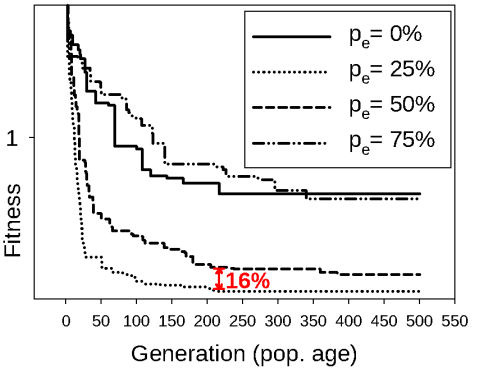


Figure 4.6 GENESIS elitism rate impact on quality for MCNC circuit s38417 (6406 CLBs, 29 Inputs, 106 Outputs), n_p=100, g_p=500, $mutation\_rate$=25

of the GA. In GENESIS algorithm, we incorporate mating preferences based on mixability theory, already discussed. As shown in Algorithm 2, the pairs are selected so that they expose minimum difference in path sub-fitness value (f_{p_Π}) for all paths, among any other combination pairs. Thus, at line 15 in Algorithm 2, individuals i and j are mated together to mating pool M, when the corresponding placements have paths with similar placement quality, i.e. they "compete with each other based on how well they perform on average rather than how well they perform in any one specific combination".

4.3.1.3 Crossover

After the selection of mates using the mating operator, follows the information exchanging between the chromosomes of mates, for all individuals in mating pool M. Usually for each pair, the crossover operator is applied with a certain probability by choosing a position randomly in the chromosome and exchanging part of the chromosome referenced by this position using a replacement policy, i.e. the tails

ALGORITHM 2: Mixability-driven Mating

Input: Population on time t after Selection: $P(t)$
Input: Elitism pool on time $t+1$: $E(t+1)$
Input: Placement of every individual i on time t: $\Pi_i(t)$
Input: Mating percentage p_{mat}
Output: Mating Pool on time $t+1$: $M(t+1)$

```
for i = 1 to (p_mat × n_p) do
    for k = 1 to m paths do
        Path_Fitness_Array[f_{p_k Π}]_i = Compute_path_sub-fitness_value(Individual_i, path_k);
    end
end
for i = 1 to (p_mat × n_p) do
    min ← (+∞)
    for j = i + 1 to (p_mat × n_p) do
        Current_path_sub-fitness_diff = 0;
        for k = 1 to m paths do
            Current_path_sub-fitness_diff += abs(Path_Fitness_Array[f_{p_k Π}]_i - Path_Fitness_Array[f_{p_k Π}]_j)
        end
        if (Current_path_sub-fitness_diff < min) then
            min = Current_path_sub-fitness_diff;
            M(t+1) ← P_s(Individual_i) ⊕ P_s(Individual_j);
        end
    end
end
```

of the two strings (this is the simplest version of a crossover). The effect of the crossover is that certain properties of the individuals are combined to new ones or other properties are destroyed. A high quality netlist placement onto an FPGA requires a high quality placement of netlist's paths. In order to avoid the destruction of high-quality paths, we guide the crossover operator to be applied on random paths rather than random placements, i.e. instead of exchanging the placement of random CLB_i between two mates, we replace the placement for every CLB, IO belonging to the random $path_j$. Since a high number of overlaps may occur using such an approach, we adopt the usage of $dependability$ metric ξ in order to overcome this problem. More specifically, every time that an exchange of a path between two mates forces a netlist CLB/IO to be placed at an FPGA physical CLB/IO location which is reserved by another netlist CLB/IO, then the physical CLB/IO location is assigned to the netlist CLB/IO with higher $dependability$ value. This policy verifies that the high ξ-value CLB/IO will retain a placement so that it affects the quality of fewer paths compared to the low ξ-value CLB/IO. This description is also explained through Algorithm 3.

ALGORITHM 3: Path-driven Crossover

Input: Population on time t after Mating: $P(t)$
Input: Mating pool on time $t+1$: $M(t+1)$
Input: Placement of every inividual i on time t: $\Pi_i(t)$
Input: Dependability vector $\hat{\xi}$ for every node (CLB/IO) of netlist.
Output: Crossover Pool on time $t+1$: $C(t+1)$

```
1  for i = 1 to size(M(t+1)) step 2 do
2      mateA = M(t+1)_i; mateB = M(t+1)_{i+1};
3      rpath = rand(1, n paths);
4      /* TryExchangePath(MateA, MateB, rpath); */
5      for k = 1 to m nodes of rpath do
6          foreach (MateA, MateB) do
7              if (Π[node_k] overlaps with Π[node_k'] ) /* node_k' has been mapped to
                 Π[node_k'] at P(t) of the other mate*/ then
8                  if (ξ̂[k]_i < ξ̂[j]_i) then
9                      Assign node_k' to free node or exchange with low-ξ node;
10                 else
11                     Assign node_k to free node or exchange with low-ξ node;
12                 end
13             else
14                 ExchangePath(MateA, MateB, rpath); /* no overlap exists */
15             end
16         end
17     end
18 end
```

4.3.1.4 Mutation

While previous operators are constructed using a conservative methodology regarding the freedom of chromosome elements to randomly assigned on the search space (elitism-based selection, mixability-driven mating, path-driven crossover), there is a high probability of premature convergence of the GA to sub-optimal solution. In order to prevent this, our algorithm incorporates also a mutation operator with some predefined probability over the population. During mutation, a path is randomly chosen and its nodes are assigned to random spatial locations. The purpose of mutation is to avoid getting stuck in local minima and also to assure the exploration of new solutions of the search space. The problem arises from this operator is how to define the probability threshold. At one hand, too low mutation probability may cause convergence into a local minimal while on the other hand too high frequent mutations will prevent the convergence of the GA, resulting into a random walk. On GENESIS algorithm we provide a novel self-adaptive mutation operator which exhibits dynamic mutation probability. While there are existing works on controlling the mutation probability at run-time (i.e. [33], [34]), most of them often use the fitness metric deviation from generation to generation in order to adapt the probability. We extend this approach by guiding the selection of probability based on the similarity of placements on population. This idea derived from our initial exper-

imental analysis, where while the GA placer (prior to dynamic mutation) converged to a solution, we noticed placement similarities among individuals. Based on the metric $diversity$, which represents a qualitative metric of placement similarities, we increase the mutation probability when $diversity$ is decreasing and vice versa. Thus we use an inversely proportional activation function f_d to accommodate this feature. This operation is described by Algorithm 4.

ALGORITHM 4: Self-adaptive Mutation

Input: Population on time t after Crossover: $P(t)$
Input: Crossover pool on time $t+1$: $C(t+1)$
Input: Placement of every inividual i on time t: $\Pi_i(t)$
Input: Dependability vector $\hat{\xi}$ for every node of netlist.
Output: Population on time t: $P(t+1)$

```
1  for i = 1 to n_p do
2      Δ_P=CalcDiversity(P(t));
3      b_m= f_d(Δ_P);
4      if (rand(0,1) < b_m) then
5          rpath = rand(1, n paths);
6          for k = 1 to m nodes of rpath do
7              /* Random FPGA CLB for netlist CLB and FPGA IO for netlist IO */
8              /* Mutation of node_k */
9              Π[node_k] =rand(1, m nodes);
10             if (Π[node_k] overlaps with Π[node_k'] ) /*node_k' has been mapped to
                 Π[node_k'] at P(t) of the other mate*/ then
11                 if (ξ̂[k]_i < ξ̂[j]_i) then
12                     Assign node_k' to free node or exchange with low-ξ node;
13                 else
14                     Assign node_k to free node or exchange with low-ξ node;
15                 end
16             end
17         end
18     end
19 end
```

4.3.2. GENESIS Coarse-grain Parallelism Engine

The motivated idea at the proposed GA-based placement is to split the whole problem into a number of distinct sub-problems, which can be solved simultaneously using multiple processors. Such a divide-and-conquer approach is applied using a coarse-grain method: the operators of *selection*, *mating*, *crossover* and *mutation* are grouped into non data-dependencies tasks, which are executed in parallel. This technique is feasible because the operators that are applied on a subset of population are independent from those applied to the rest population, thus there is no requirement for inter-thread communication during this phase[5]. Specifically, each operator works on several subsets of individuals found in the GA population. Thus,

[5]An exception occurs for a function within selection operator, which sorts the individuals based on their fitness value.

in a coarse description, parallelization is performed as follows: The processing work of each subset is assigned to a different parallel thread, which forms the tasks to be executed in parallel within each GA's era.

Such a coarse-grain parallelism approach in placement problem is efficient due to the improved instruction and data locality within each iteration of GA operators. This technique is also improved from task parallelism, as all shared data of population can be exchanged in a deterministic producer/consumer style, eliminating the possibility of data races and thus shortening the sequential part of GA algorithm.

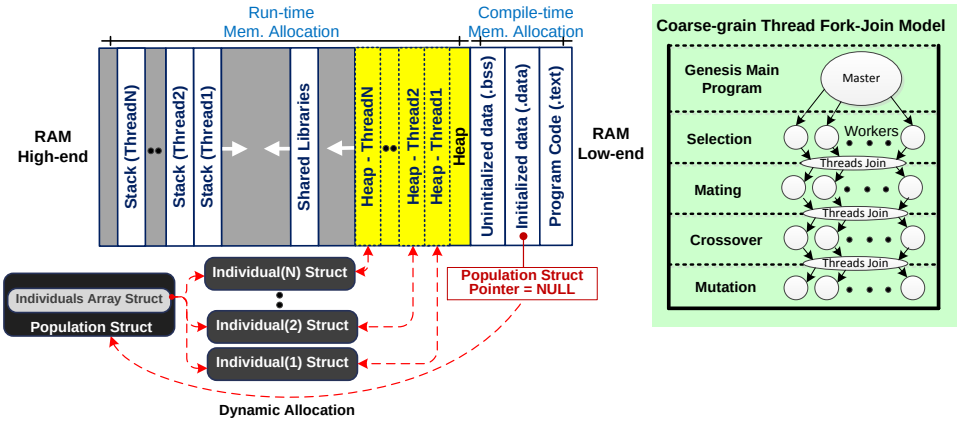


Figure 4.7 Thread-oriented GA memory structure and the corresponding fork-join model.

The overall parallelization approach is depicted in Fig. 4.7. Regarding the implementation issue, we employ the OpenMP API to express task and data parallelism into the proposed GA. In order to provide an efficient shared memory access per thread, we created an individual oriented memory organization. The population is a struct stored in global shared memory, composed of an array of structs dedicated to individuals. Each individual struct points further to a number of structs holding individual's complex data, such as the placement struct, the timing graph struct, the placement metrics struct etc. These structs are initialized at run-time through customized malloc system calls. Given that the allocation and deallocation patterns are not exhibiting a producer/consumer relation, we customized the heap architecture according to the *pure private heaps* schema [35]. Specifically, such a customization assumes that each thread has its own private heap used for every memory operation. In terms of performance and scalability, *pure private heaps* are a very efficient, as each thread has its own heap space and induces the minimum lock contention.

The efficiency of GENESIS framework is based on the parallel execution of its tasks. However, the parallelization takes place within (and not between) each of the main GA operators, i.e selection, mating, crossover and mutation. Each operator works on several subsets of individuals found in the GA population. Thus, in a coarse description, parallelization is performed as follows: The processing work of each subset is assigned to a different parallel thread, which forms the tasks to be

executed in parallel within each GA's era. By exploring many solutions (population pool) simultaneously, it is more likely to find a high quality solution, as compared to placers that initiate from a random state and incremental improve these solutions, e.g. the simulated annealing placer of VPR tool.

In general, the overhead of managing parallelism is a problem of the shared memory programming model. However it has been shown that programs parallelized for shared memory architectures can achieve satisfactory scaling up to a few hundreds of processors [36], [37]. This is possible with reasonable scaling of the problem's size in order to increase the granularity of threads and reduce the frequency of synchronization. In order to leverage the right combination of task and data parallelism while avoiding data hazards, we performed detailed profiling of the proposed algorithm during sequential execution. This enables the identification of data transactions at inter- and intra- computational kernels of the proposed algorithm. This step prevents the run-time gains retrieved by the parallel execution to be overshadowed by the communication and synchronization costs. Based on this analysis, we found that the maximum locality of references to main memory is performed inside the code of every GA operator (namely selection, mating, crossover, mutation). This is expected since anyone of these operators accumulates the same kind of data over the population pool, i.e. the mutation access placements after crossover and constructs the dependability vector iterative. Given also that every operator works on up to two solutions (individual) per iteration, we chose a coarse-grain parallelism scheme, where the main loop of every operator spawns to multiple threads, according to the number of available processing cores and the user options.

In order to better clarify the aforementioned investigation, Figure 4.8 plots the average (over the 20 biggest MCNC benchmarks) percentages for executing sequentially the functions that compose GENESIS algorithm. These results are retrieved with the usage of Valgrind suite (Callgrind tool). For this analysis, the algorithm's kernels are clustered into two groups depending if they could be executed in parallel with the OpenMP API, or not. Even though these results depend on the netlist size, the FPGA architecture, the number of physical cores and the usage of hyperthreading, as well as numerous parameters of GENESIS algorithm (e.g., population size, age, elitism percentage, mating percentage and mutation probability), the deviations in execution run-time required for different kernels are small enough. Based on Figure 4.8, we conclude that, on average, 81% of the total execution time corresponds to source code that is parallelized with OpenMP API. This result sets an upper bound constraint to the theoretical speedup obtained using a multicore shared memory CPU, as it is further discussed in Section 4.5.

4.4. Application-Aware Tuning of GENESIS Evolutionary Placement

The GENESIS evolutionary placement algorithm exposes a set of parameters that control the optimization procedure. In a common case scenario, these parameters are pre-configured by the tool developers, thus each application to be placed

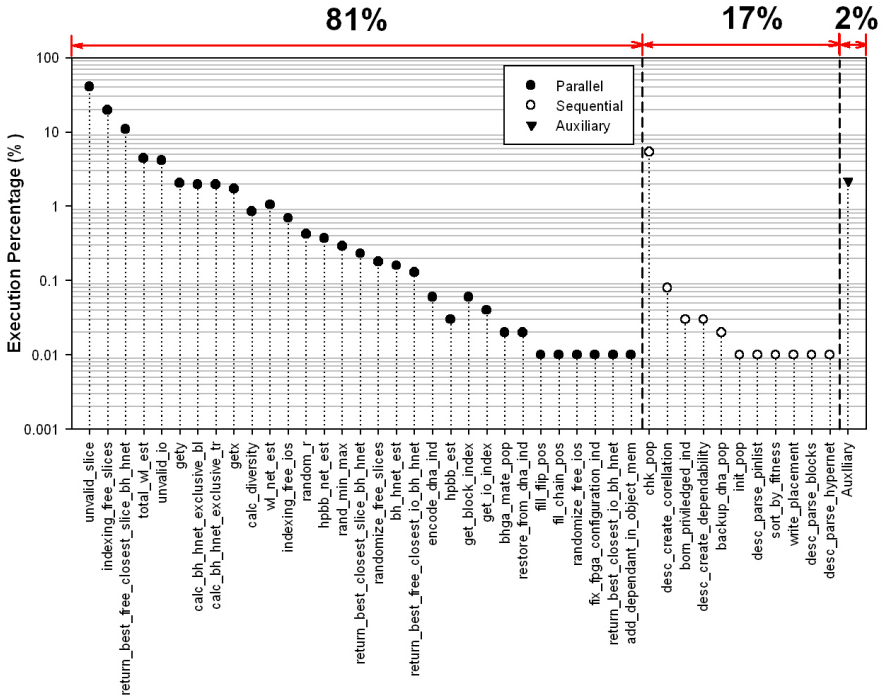


Figure 4.8 Execution run-time breakdown analysis for functionalities of GENESIS placer.

on the FPGA will be optimized according to a fixed configuration vector that represent a typical instance for the algorithm. However, there is a large diversity regarding the circuit structures and their organization found in different applications, e.g. compute-bound applications versus I/O-bound applications etc., which make extremely difficult to find a single "golden" configuration that optimizes the placement algorithm in all the cases. In order to address this diversity, we extended GENESIS placement framework with an exploration methodology that enables the application-aware tuning of the placer's parameters.

The proposed tuning methodology consists of two phases. Given a set of representative applications, phase 1 performs an offline clustering operation that groups together applications sharing the same features. The applications bound to each cluster are explored in a combined manner to characterize the per cluster solution space. We consider the general case of multiple optimization objectives, thus for each cluster we derive a set of approximated Pareto configurations, so as the tool developer to select the one that fits his criteria, e.g. Power×Delay, Power×Wirelength, etc.

In phase 2, the selected configuration per cluster (computed offline by phase 1) will be used during the actual GENESIS run-time to configure the evolutionary optimization algorithm according to the characteristics of the application under

placement. The assignment of each new application to a specific cluster – thus also to a specific configuration vector for the GENESIS engine – is based on its distance to each cluster centroid. The closer the application is to a specific centroid the more similar characteristics is shared with the other applications that formed the cluster. In case of a tie, i.e. the distance between two or more centroids is the same, the application is assigned to the cluster with the higher cardinality.

4.4.1. Application Level Clustering

The goal of application level clustering is that given a set of representative applications to generate a set of categories that group together the application that share similar features regarding their circuit structure. Without loss of generality, in this research we considered the 20 largest applications found in the MCNC benchmark suite (Table 4.3). The goal of application-level clustering is to group together applications exposing similar features. For each of the formed clusters a promising configuration will be derived in order to be used during the run-time of the GENESIS tool. Thus, when a new application is entered for placement, it is assigned to one of these clusters so that the optimization engine will be configured appropriately. This instructs that the clustering should be based on features available after logic synthesis and packing (to enable a cluster indexing prior the GENESIS engine invocation) rather than actual placement of the application.

During logic synthesis and packing the circuit description of the application is quantitatively analyzed. The type of supported analysis varies according to the logic synthesis engine and packing engine. In order to be independent from tool specific reports and analysis, in this research we consider a minimal set of available analysis data given by a metrics vector, i.e. $\#CLBs, \#IOs, \#Nets$. The first four columns of Table 4.3 show the logic synthesis and packing analysis results for the targeted set of representative applications.

Although the metrics vector evaluates the application in terms of number of allocated resources, it fails to characterize the actual circuit's internal structure that drives placement. We alleviate this characterization inefficiency of the original metrics vector by defining a new set of features that captures in a better manner the circuit's structure regarding placement. We define two new features and reuse the number of connections ($\#Nets$) feature found in the original characterization metrics vector. Specifically:

$$f_1 = \#Nets \tag{4.6}$$

$$f_2 = \frac{\#LUTs}{\frac{\#IOs+\#DFFs}{2}} \tag{4.7}$$

$$f_3 = \begin{cases} \sqrt{\#CLBs}, & \text{if } \sqrt{\#CLBs} > \frac{\#IOs}{4}. \\ \frac{\#IOs}{4}, & \text{if } \sqrt{\#CLBs} \leq \frac{\#IOs}{4}. \end{cases} \tag{4.8}$$

Feature f_1 reports the intensity of each benchmark in terms of wire connections. Feature f_2 estimates the timing path size, i.e. the number of $\#LUTs$ found in each

timing path consisting between two DFF or two $\#I/O$ nodes, considering a balanced distribution of nodes to timing paths. Finally, feature f_3 estimates the minimum square FPGA that fits the application considering either the case that the logic elements dominate the application, e.g. compute-intensive or the I/O elements are the dominant ones, e.g. I/O-intensive applications..

After defining the new feature space, F, through Equations 4.6 - 4.8, we characterized each application with the feature vector and perform clustering to derive with groups of application that share the same feature behavior. We used k-means [38] clustering for the aforementioned procedure, with a configurable number of clusters, k. The clustering procedure partitions the feature space F into k regions of interest, F_i $i \in \{0 \dots k-1\}$, e.g. a region of many connections, short timing paths and large minimum squared sized FPGAs that corresponds to IO intensive applications with limited computation intensity. The number of regions, k is highly correlated with the effectiveness of the clustering procedure. Coarse-grained clustering, i.e. small values of k, increases the diversity of the features characterizing each application group, while fine-grained clustering, i.e. one cluster per application, reduces the diversity features but approximates aggressive application specific solutions. In this exploration, we considered k=8 for the given application set (Fig. 4.9), as a balanced solution between coarse- and fine-grained clustering.

Table 4.3 Feature characterization of the employed benchmark suite.

Benchmark	Original Metrics Space				Min. sized FPGA
	#CLBs	#IOs	#Nets	#DFFs	
alu4	1,519	22	1,523	0	39×39
apex2	1,878	42	1,899	0	44×44
apex4	1,262	27	1,264	0	36×36
bigkey	1,707	460	2,046	224	107×107
clma	8,071	33	8,135	33	90×90
des	1,591	501	1,837	0	126×126
diffeq	1,497	103	1,539	377	39×39
dsip	1,370	426	1,599	224	107×107
elliptic	3,604	245	3,731	1122	62×62
ex1010	4,598	20	4,555	0	68×68
ex5p	1,064	71	1,046	0	33×33
frisc	3,556	136	3,552	886	60×60
misex3	1,397	28	1,390	0	38×38
pdc	4,575	56	4,522	0	68×68
s298	1,931	10	1,935	8	44×44
s38417	6,235	135	6,237	1463	79×79
s38584	6,447	343	6,331	1260	81×81
seq	1,750	76	1,774	0	42×42
spla	3,690	62	3,647	0	61×61
tseng	1,047	174	1,096	385	44×44
Average:	2,679.42	154.58	2982.9	299.1	64×64

Figure 4.9 Clustering result on feature space F.

4.4.2. Cluster Level Exploration for Optimal Configuration Extraction

The goal of cluster level exploration is to find the configuration of GENESIS's parameters that provide optimized placement for the application set included in each of the clusters F_i. Currently, the GENESIS placement framework exposes three parameters that control the evolutionary optimization procedure, namely (i) *population age (pa)*, (ii) *population size (ps)* and (iii) *aggressiveness of timing over wirelength optimization* (parameter α of the fitness function). Specifically, regarding the latter parameter, its sensitivity in Equation 4.1 highly affects the QoR of the solution space. Even though there is correlation between wirelength and delay, as the Elmore model takes into account the length of wires, however, based on our analysis, depicted in Table 4.4, we show that both of these parameters have to be taken into account during placement, as this leads to the maximum gains in operating frequency. In order to determine the value of α we propose a heuristic exploration analysis as follows.

Configuration space Ω is defined as the set of all the possible configuration vectors, $\cup < ps_k, pa_k, a_k > \in \Omega$. For each of the formed clusters, F_i, a configuration vector that optimizes one or more design objectives, e.g. delay, power, wirelength,

Table 4.4 Evaluation of different (wirelength-driven and timing-driven) configurations for the GA placer against the introduced GENESIS framework.

	GENESIS	**Wirelength-driven (a=0)**		**Timing-driven (a=1)**	
Benchmark	Delay (Solution 1)	Delay	Penalty (vs. GENESIS)	Delay	Penalty (vs. GENESIS)
alu4	7.38E-008	1.07E-007	1.45×	1.09E-007	1.47×
apex2	8.81E-008	1.39E-007	1.58×	1.14E-007	1.29×
apex4	7.72E-008	1.16E-007	1.50×	9.16E-008	1.19×
bigkey	6.63E-008	2.05E-007	3.09×	9.79E-008	1.48×
clma	1.89E-007	2.57E-007	1.36×	2.11E-007	1.12×
des	1.05E-007	2.86E-007	2.73×	2.60E-007	2.48×
diffeq	6.09E-008	9.73E-008	1.60×	7.42E-008	1.22×
dsip	6.38E-008	1.92E-007	3.00×	1.02E-007	1.60×
elliptic	1.10E-007	2.68E-007	2.43×	1.36E-007	1.24×
ex1010	1.16E-007	1.97E-007	1.69×	1.66E-007	1.43×
ex5p	6.64E-008	1.20E-007	1.80×	9.57E-008	1.44×
frisc	1.27E-007	2.35E-007	1.85×	1.78E-007	1.40×
misex3	7.11E-008	1.10E-007	1.54×	8.58E-008	1.21×
pdc	1.08E-007	2.74E-007	2.54×	1.74E-007	1.61×
s298	1.32E-007	2.32E-007	1.75×	1.41E-007	1.07×
s38417	1.01E-007	1.74E-007	1.72×	1.14E-007	1.13×
s38584	8.7E-008	1.33E-007	1.53×	1.15E-007	1.32×
seq	7.79E-008	1.25E-007	1.60×	1.07E-007	1.37×
spla	8.46E-008	1.90E-007	2.25×	1.51E-007	1.79×
tseng	5.29E-008	8.25E-008	1.56×	6.17E-008	1.17×
Average:	**9.29E-008**	**1.77E-007**	**1.93**×	**1.29E-007**	**1.40**×

tool's run-time etc., of the GENESIS engine is derived through off-line exploration. This optimized configuration is be used after the deployment of the GENESIS tool for tuning the placement engine according to the features of the input application.

We target the problem of finding/approximating an optimized per cluster configuration in its generalized form, i.e. as a multi-objective optimization problem. Single-objective optimization variants is a subset of the targeted problem form, thus the same techniques can be applied in a straightforward manner. Let $< LO = \{o_1, \dots, o_N >$ defined as the vector of optimization objectives, e.g. $< o_1, o_2, o_3 >=< Delay, Power, Runtime >$. Given that each cluster, F_i, includes a number of applications, i.e. $app_{i,j} \in F_i$, multiple instances of optimization objective vectors, $LO^{i,j}$ exist within each cluster F_i, one for each $app_{i,j} \in F_i$. Thus, the optimization should be performed across the all the applications, $app_{i,j}$ found within each cluster, in order to optimize the central tendency of the overall cluster F_i. The problem of finding the per cluster optimal configuration, can be defined as follows for cluster F_i:

$$\min_{x \in \Omega} \begin{bmatrix} gm(o_1^{i,0}, \dots, o_1^{i,j}, o_1^{i,|F_i|-1})(x) \\ gm(o_2^{i,0}, \dots, o_2^{i,j}, o_2^{i,|F_i|-1})(x) \\ \dots \\ gm(o_N^{i,0}, \dots, o_N^{i,j}, o_N^{i,|F_i|-1})(x) \end{bmatrix} \tag{4.9}$$

where $gm(x_1, \dots, x_i, \dots, x_N) = \sqrt[N]{x_1 \times \dots \times x_i \times \dots \times x_N}$ is the geometric mean of the objective associate with $app_{i,j}$.

Since the problem is multi-objective, we are interested in finding the Pareto optimal set (Pareto front) of configurations, $X_i^{Popt} \in \Omega$. Then, the tool developer can select/decide for each cluster F_i the configuration vector $x \in X_i^{Popt}$ that satisfies its requirements. However, finding the exact Pareto front, for the optimization problem 4.9, is computationally impractical since it requires the full evaluation (placement and routing) over all the configurations found in Ω of all the applications found in each F_i.

We developed an exploration strategy, shown in Algorithm 5, to extract an approximate Pareto front, $\widehat{X}_i^{Popt}$ of each cluster F_i. The proposed strategy iterates over all the defined clusters. For each cluster, it samples the configuration space Ω following a uniform probability distribution, $SampleSpace(F_i) \subseteq \Omega$. Each application, $app_{i,j} \in F_i$ is explored (= evaluated) according to the common sample configuration space, $SampleSpace(F_i)$. The generated vector space, D_j^i, is a map linking each configuration vector found in $SampleSpace(F_i)$ with its metric defined in the vector of optimization objectives, LO. The geometric mean operator is applied to every D_j^i generating the D^i space that represents the problem defined in eq. 4.9.

D^i provides a localized view of the targeted solution space, since it includes characterized solutions (regarding the optimization objectives) only on the "known points", i.e. the configurations, $SampleSpace(F_i) \subseteq \Omega$, that has been actually evaluated through GENESIS. The rest of the configuration vectors found in Ω, which have not been evaluated, form the "unknown points" of the solution space. We approximate the "unknown points" of Ω, by using interpolation[6] with the "known points" in D^i forming the training database. The $\widehat{D}_{total}^i$ is the predicted approximation of the solution space over all the configuration vectors of $|Omega$, generated after interpolation. Specifically, the *inverse distance weighting* (IDW) [39] interpolation technique has been adopted. The value of an interpolated unknown point, $x \in F_i$ is computed by using N known observations $y_k \in D^i$ as follows:

$$r(x) = \frac{\sum_{k=1}^{N} w_k(x) y_k}{\sum_{k=1}^{N} w_k(x)}, w_k(x) = \frac{1}{dist(x, x_k)^p} \tag{4.10}$$

where $w_k(x)$ is the weighting function, $dist$ is the distance between the known point x_k and the unknown point x and p is a real number called the power parameter of the model. The larger the value of p the greater the influence of the points

[6] Differently from regression techniques, interpolation does not produce any prediction error on the known data.

closest to the interpolated point. For N-dimensional configuration space, there is a theoretical lower bound of the power parameters, $p \geq N$, in order to restrict the interpolated values to be dominated by distant points. Thus, for the target three-dimensional configuration space, $< ps_k, pa_k, a_k > \in \Omega$, we consider a range of $p \in [3, 6]$ with step 0.5. Through iterative evaluation, the selected p value is determined as one that maximizes the IDW's model accuracy. Finally, the approximated solution space, $\widehat{D}^i_{total}$, is filtered according to the targeted list of objectives, LO to generate the corresponding Pareto front.

Fig. 4.10 shows the Pareto front curves for each F_i generated by the proposed exploration approach. In the specific setup, we consider tuning GENESIS according to two optimization objectives, i.e. critical delay of the placed and routed circuits and actual tool run-time of the GENESIS framework to perform placement and routing. The tool developer is responsible to select/decide a specific Pareto configuration per cluster to be used during actual tool run-time, when an application is recognized to match the characteristics of a specific cluster.

ALGORITHM 5: Exploration for Per Cluster Approximate Optimal Configuration

Input: Parameter space of GENESIS optimizer: Ω
Input: Cluster set: F_i, $i \in \{0, .., k-1\}$
Input: Max. number of per cluster evaluations: M_E
Input: List of optimization objectives: $LO = \{o_1, ..., o_N\}$
Output: Optimal per cluster parameter configuration vector: $\widehat{X}^{Popt}_{F_i} \in \Omega$
foreach F_i, $i \in \{0, \dots, k-1\}$ **do**
  $SampleSpace(F_i) \leftarrow rand(\Omega, M_E)$
  foreach $app_{i,j} \in F_i$ **do**
    $D^i_j \leftarrow explore(app_{i,j}, SampleSpace(F_i))$
  end
  $D^i \leftarrow gmean(D^i_0, \dots, D^i_j, \dots, D^i_{|F_i|-1})$
  $\widehat{D}^i_{total} \leftarrow interpolate(D^i, \Omega)$
  $\widehat{X}^{Popt}_{F_i} \leftarrow ParetoFrontGeneration(\widehat{D}^i_{total}, LO)$
end

4.5. Experimental Results

This section provides a number of quantitative results that prove the effectiveness of the introduced solution, as compared to the state-of-the-art placement algorithm found in VPR 4.30 tool [14]. For this purpose, we employ the 20 biggest MCNC benchmarks, whereas the target FPGA is an island-style architecture [40]. Note that such an architecture is commonly found in commercial reconfigurable platforms, such as the Xilinx Virtex and Altera Stratix FPGAs. The cluster level exploration strategy has been implemented in the Multicube explorer [41] framework. Regarding the application placement, it was performed on an 8-core Intel XEON Processor running at 2.33GHz with 4GB of RAM. Table 4.3 presented previously summarizes the properties of the employed benchmark suite, as well as the size of the minimum FPGA array, where each of the benchmarks are mapped to. For the rest of the experimental results we provide results for two configurations of our GA placer. More specifically, the first configuration (mentioned as "Solution 1")

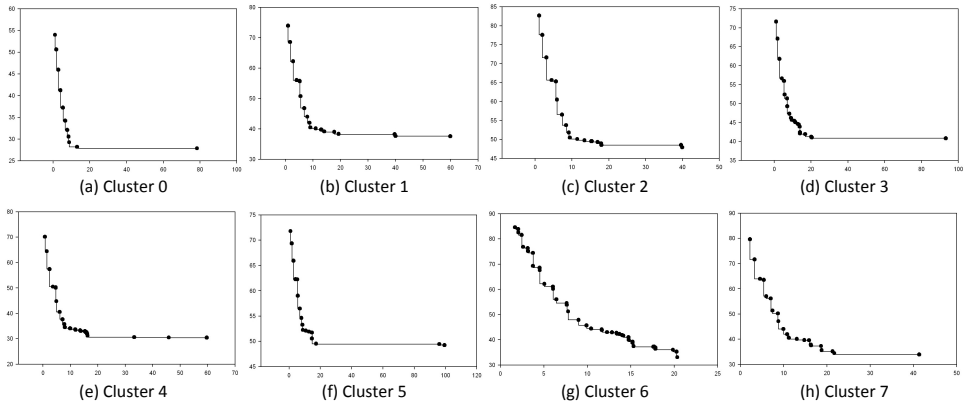


Figure 4.10 The Delay vs. Rutime Pareto front per cluster F_i. Data normalized to the per cluster least efficient evaluated solution.

corresponds to faster execution of GA (i.e. minimum execution run-time), whereas the second instance ("Solution 2") leads to the minimum application's delay (i.e. maximum operating frequency).

Since the quality of solutions derived from GA is tightly firmed to the calibration of GENESIS parameters, various instantiations of this algorithm were tested during the development phase. For instance, Figure 4.11 plots the impact of *population size* and *population age* parameters at the application's delay and the execution run-time for placement of *bigkey* benchmark. For demonstration purposes, at this figure we also plot the corresponding solution that retrieved with the usage of VPR tool [1]. Based on this figure we can conclude that almost the half of the available solutions can operate under higher frequencies (i.e. with smaller critical path delay) than the corresponding placements retrieved with VPR, whereas the maximum performance improvement achieved by our introduced GA placer is up 40%.

One of the most important features provided by our GA, as compared to rest placers, affects the inherent flexibility to be executed in a parallel fashion. As we mentioned in Section 3, the new algorithm was developed with the usage of OpenMP API in order to provide the maximum possible speedup. Table 4.5 quantifies the execution run-time, the maximum operating frequency, as well as the power consumption for performing application placement with the original VPR tool and the two instances of introduced GA algorithm.

As shown in Table 4.5, the proposed approach achieves on average to place a benchmark 15× and 67× faster, as compared to VPR, whereas for specific benchmarks (e.g. *tseng*) this gain is up to 137×. Additionally, the proposed GA algorithm outperforms our previous implementation presented in [17], since in that work the average speedup in execution run-time compared to VPR was 6.6×. We have to mention that the execution speedup is not constant among the benchmarks because it is tightly firmed to inherent properties of benchmarks (e.g. number of I/Os

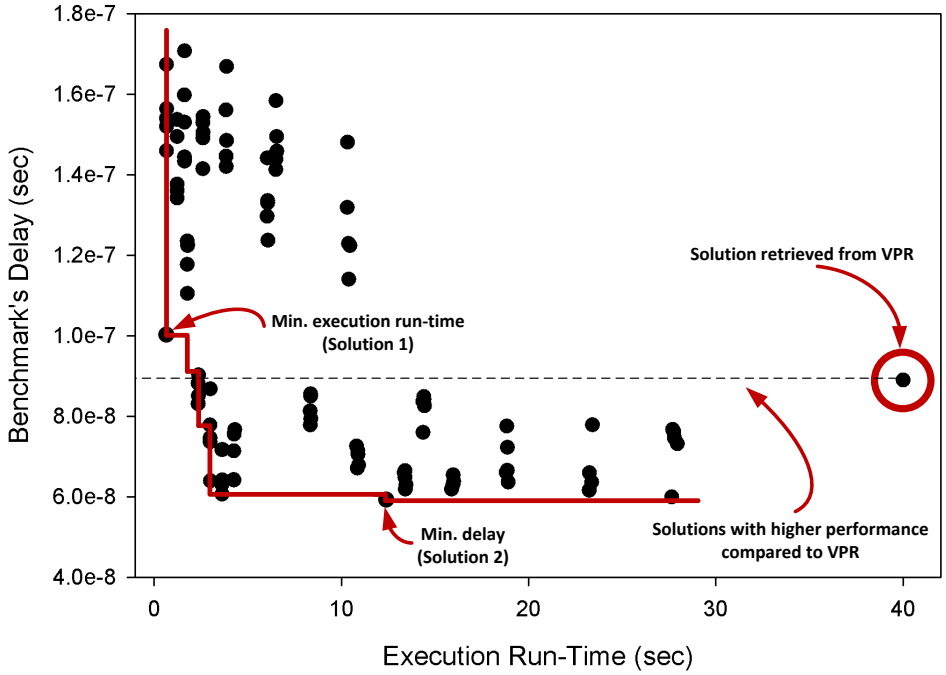


Figure 4.11 Evaluation for different combinations of *population size* and *population age* regarding the *bigkey* benchmark.

and logic blocks, length of critical path, fanout). GENESIS framework provides to the designer also the freedom to configure the tool parameters in a hand-written manner in case that he desires to fine-tune the optimization engine, for maximizing as much as possible the gains per application. However we underline that Table 4.5 presents results when the parameters of GENESIS are tuned by the exploration methodology of Section 4.4.

A second important observation from Table 4.5 is that the aforementioned run-time enhancement achieved with the usage of the proposed framework is coming together with gains in maximum operating frequency and power consumption. More specifically, the algorithmic instantiation that corresponds to the "Solution 1" achieves to improve the maximum operating frequency (i.e. reduce critical path) by a factor of 1.16×, whereas the "Solution 2" exhibits an average maximum operating frequency lower by 0.53× as compared to the application implementation derived with VPR tool. For the case of "Solution 1", we have to take into consideration that these gains are complementary to the speedups discussed previously regarding the GENESIS's run-time execution, while for "Solution 2" we can observe the trade-off between ultra-fast execution of the placer (Table 4.5, col. 4) versus the quality of the circuit solution (Table 4.5, col. 7).

Table 4.5 Comparison in term of execution run-time, maximum operating frequency and power consumption.

Benchmark	Execution Run-Time (second)			Delay (second)			Power Consumption (Watt)		
	VPR	GENESIS		VPR	GENESIS		VPR	GENESIS	
		Solution 1	Solution 2		Solution 1	Solution 2		Solution 1	Solution 2
alu4	25.00	1.587	0.386	7.79E-08	7.38E-08	1.30E-07	0.1841	0.1889	0.1704
apex2	34.00	2.288	0.562	8.63E-08	8.81E-08	1.81E-07	0.2175	0.2170	0.1927
apex4	21.00	1.276	0.317	7.33E-08	7.72E-08	1.41E-07	0.1478	0.1455	0.1289
bigkey	56.00	2.162	0.714	8.96E-08	6.63E-08	1.54E-07	1.1096	1.1879	1.1004
clma	372.00	23.121	4.938	2.06E-07	1.89E-07	4.65E-07	0.7684	0.7800	0.7055
des	50.00	2.446	0.910	1.47E-07	1.05E-07	3.09E-07	1.4125	1.4293	1.3928
diffeq	25.00	1.711	0.427	6.48E-08	6.09E-08	1.79E-07	0.1695	0.1738	0.1460
dsip	45.00	1.703	0.639	9.52E-08	6.38E-08	1.51E-07	1.0896	1.1670	1.0790
elliptic	98.00	6.527	1.483	1.13E-07	1.10E-07	2.61E-07	0.3891	0.3891	0.3520
ex1010	127.00	7.724	1.918	1.98E-07	1.16E-07	2.70E-07	0.4214	0.4351	0.3856
ex5p	18.00	1.071	0.265	6.60E-08	6.64E-08	1.30E-07	0.1381	0.1371	0.1180
frisc	94.00	6.240	1.422	1.44E-07	1.27E-07	3.24E-07	0.3314	0.3391	0.3004
misex3	22.00	1.441	0.356	7.71E-08	7.11E-08	1.14E-07	0.1728	0.1777	0.1594
pdc	134.00	8.844	2.020	1.42E-07	1.08E-07	2.27E-07	0.4374	0.4460	0.3910
s298	33.00	2.078	0.495	1.31E-07	1.32E-07	2.78E-07	0.1989	0.1992	0.1798
s38417	189.00	16.385	3.212	1.02E-07	1.01E-07	2.00E-07	0.6902	0.4873	0.6600
s38584	204.00	16.697	3.223	8.95E-08	8.70E-08	1.34E-07	0.7070	0.5169	0.7241
seq	32.00	2.078	0.507	6.98E-08	7.79E-08	1.08E-07	0.2135	0.2064	0.1897
spla	94.00	5.861	1.219	1.24E-07	8.46E-08	1.33E-07	0.3529	0.3633	0.3238
tseng	17.00	0.983	0.124	5.53E-08	5.29E-08	1.60E-07	0.2141	0.2182	0.1862
Average:	84.50	5.611	1.257	1.08E-07	9.28E-08	2.03E-07	0.4683	0.4602	0.4443
Ratio:	1.00	15.06×	67.24×	1.00	1.16×	0.53×	1.00	1.02×	1.05×

Similar to previous metrics, Table 4.5 shows that our introduced framework does not impose any significant degradation in power consumption, as compared to the application implementation with VPR. More specifically, the proposed framework leads to an increase in power consumption up to 1.05× on average. Note that these values are affected by the maximum operating frequency discussed previously. In case that our placer is tuned to retrieve a solution with identical delay to VPR, then the corresponding average power savings for this alternative solution ("Solution 2") is about 6%.

The previous discussed results highlight that the proposed approach exhibits faster execution compared to existing placers (e.g. VPR), whereas it also leads to superior performance at the derived placements. This mainly occurs because it explores more effectively the search space, whereas due to its increased diversity, the probability to be trapped in a local minima is significant lower compared to solvers based on simulated annealing. In order to depict this differentiation, Fig. 4.12 quantifies the number of different intermediate solutions (in order to find the final placement) that are explored with the usage of VPR and our introduced GA. We have to notice that vertical axis in this figure is plotted in logarithmic scale. The results summarized in this figure depict that "Solution 1" and "Solution 2" explore faster about 157× and 35× more solutions, as compared to VPR tool.

Next, we quantify the efficiency of the proposed GA to reduce execution runtime by incorporating the advantages provided by existing multi-core platforms. Fig. 4.13 plots the speedup whenever the introduced GA is executed on a 4-core and 8-core processor, for various number of threads. For demonstration purposes, all the results are plotted in normalized manner over the corresponding single-core and single-thread execution. Additionally, for sake of completeness we also provide the corresponding theoretical curve, as it is retrieved from the Amdahl's model [42]. This curve highlights the theoretical limit, given that the GENESIS tool has currently a parallelized code portion of 81%, based on profiling results.

Based on this diagram, we can conclude that the experimental results for both curves (4-core and 8-core) are very close to the theoretical thresholds. This means that our implementation of GA takes advantage almost of the maximum possible performance enhancement due to the processor parallelism. Also, it is confirmed that the maximum speedup enhancement of GA is achieved when the number of threads are equal to the number of processing cores, since after this turn point, there is saturation. Finally, we observe that the deviation between the actual and the theoretical scaling behavior for the 8-core machine is higher than the one for the 4-core. The main reason of this observation is that the 8-core workstation is composed of two 4-core individual chipsets. While the parallel runtime (OpenMP) is aware of 8 cores with the same characteristics, the true hardware offers two single chip 4-core CPUs. Thus, the communication among threads running in different CPUs is established out-of-chip and not in the same die as happens in the case of the 4-core CPU based workstation, which impacts the scaling behavior.

More specifically, the scope of Fig. 4.13 is to depict the scalability of the proposed algorithm in terms of self-speedup according to the number of threads and the available number of physical cores. In order to retrieve the corresponding

Figure 4.12 Candidate solutions that are evaluated during placement.

measurements we run GENESIS in two working stations, the first equipped with two Intel Xeon E5345 2.33GHz quad-core processors (total 8 cores), 8MB cache (total cache 16MB) and 4GB RAM running Red Hat Enterprise Linux Server 6.3 and the second equipped with an Intel Core 2 Quad Q9400 2.66GHz (total 4 cores), 6MB cache and 4GB RAM running GNU/Linux kernel 3.4.

The theoretical upper bound of the self-speedup gain is also depicted in the figure according to the Amdahl's law. According to this speedup model, since we measured that the 81% of the code is parallelized (after profiling of all 20 biggest benchmarks), we can theoretically estimate the maximum speedup when executing the code to a CPU with M-processors and N-threads of execution. Of course this is only a theoretical analysis which neglects any bottlenecks coming from the underlying CPU/system architecture, such as the memory hierarchy, the memory bandwidth, the cache size/organization and the architecture of the shared memory among individual cores of the CPU. Not only these hardware aspects but also software related factors such as the OS scheduler, the memory manager and the parallel runtime (in our case OpenMP), including forking mechanism, dispatcher and resource binder, lead to deviation of the true speedup curve over the theoretical. For the same reason we also expect some deviation when executing our algorithm in different hardware setups. This is the case of the measurements related to Fig 4.13, where the deviation theoretical speedup is bigger at the 8-core

Figure 4.13 Scaling of execution speedup for the GENESIS placer.

system than the deviation at the 4-core system. We believe that the main reason for this imbalance is that our 8-core CPU based workstation is composed of two 4-core individual CPUs. While the parallel runtime (OpenMP) is aware of 8 cores with the same characteristics, the true underlying hardware offers two single chip 4-core CPUs. Thus the communication among threads running in different CPUs is established out-of-chip and not in the same die as happens in the case of the 4-core CPU based workstation. This means that intra-thread communication requests are exported to front-side-bus (FSB) in our 8-core based workstation, when the same transactions are established only on the back-side-bus (BSB) in our 4-core based workstation. Given the high clock/bandwidth difference between FSB-BSB and the unawareness of the parallel runtime (OpenMP) of this special architecture (in order to decrease as possible the off-chip intra-thread communication), an increase in the aforementioned deviation metric is expected.

For the scope of this research work, in order to support the aforementioned argument, we profiled GENESIS tool on the two workstations using 1-32 OpenMP threads. Note that since we employ a coarse grain parallelization model, where the granularity level is set at individuals structs of the population, there is no gain on increasing the number of threads more than the population size, since extra threads will remain idle. The setup for the analysis below is $pop_size = 30$, $pop_age = 90$. For the rest of measurements we employed Intel Vtune Aplifier XE 2013 (Non-

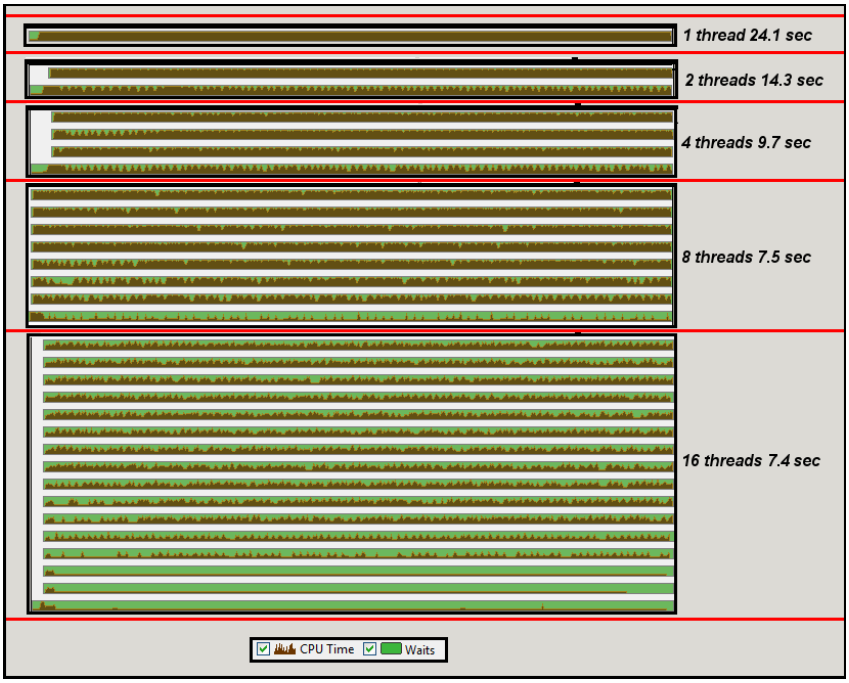


Figure 4.14 GENESIS thread time breakdown analysis: CPU and Wait Time.

Commercial license). Firstly we show that increasing only the number of threads does not guarantee increased speedup, since there is saturation point due the overhead of threads synchronization. This happens because increasing the number of threads performing some fixed load of work in parallel, each thread gets less work and the overhead, as a relative measure, will get larger. Figure 4.14 depicts the profiling of *ex1010* benchmark on our 8-core based workstation.

To better clarify Figure 4.14, we employ the definitions of *Wait Time* and *CPU Time*. The *Wait Time* is the amount of time when a thread is stalled, waiting for some event to occur, such as synchronization or I/O event. The *CPU Time* is the amount of time when a thread is executed on a logical processor. It is clear that as long as the number of threads is increased, the *Wait Time* of every thread is increased. When a total of 16 threads is used, there is the highest balance between *CPU Time* and *Wait Time*, leading to the minimal runtime. However the speedup gain from the 8-thread configuration is negligible as also expected by the theoretical analysis. Since we follow a coarse grain parallelization model, the spawned threads have increased portion of executed code compared to a fine-grain parallelization model, i.e. the case of simple parallelization in for/while-loops. This implies lesser synchronization points for the GENESIS approach. Consequently while the bottleneck is not in the synchronization of the threads, increasing only

the number of threads leads to a simple dispatch of more atomic coarse-grain tasks to the same number of processors. Consequently, it is not expected high speedup gain after the saturation point where the number of threads is equal to the number of physical processors.

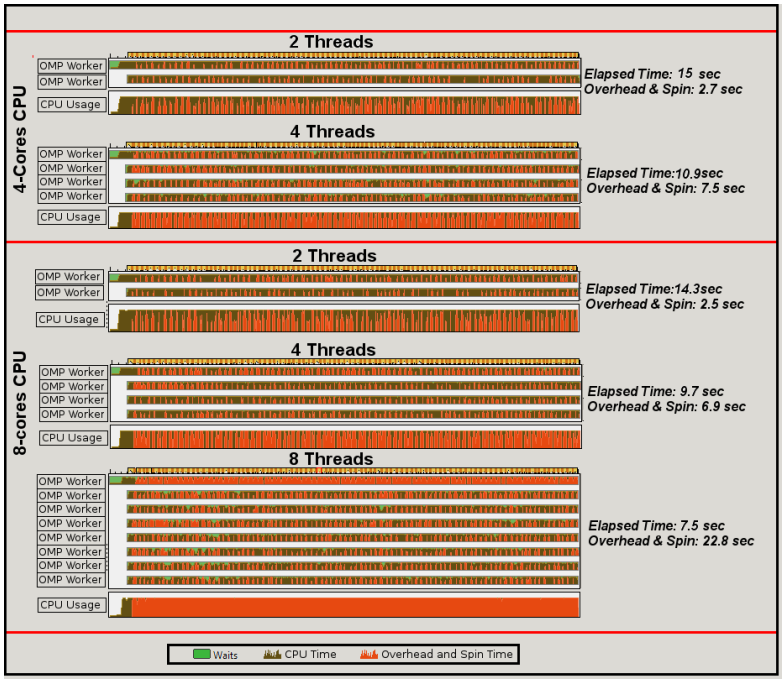


Figure 4.15 GENESIS thread time breakdown analysis: Overhead and Spin Time.

In order to identify the bottlenecks of our coarse-grained based OpenMP thread structure, we further performed profiling on OpenMP runtime. For this scope we recompiled the OpenMP runtime library in order to include debug symbols (-ggdb GCC option) and we linked GENESIS using this library. This way, Vtune suite is able to report hotspot functions in OpenMP libraries, CPU usage, and thread concurrency. We use the same benchmark (ex1010) with the same GA configuration with previous analysis. We conclude results in Figure 4.15. We employ the definitions of *Overhead Time* and *Spin Time*. *Overhead Time* is the CPU time spent on the overhead of known synchronization and threading libraries, i.e. in our case OpenMP. *Spin Time* is the *Wait Time* during which the CPU is busy. This often occurs when a synchronization API causes the CPU to poll while the software thread is waiting. Some *Spin Time* may be preferable to the alternative of increased thread context switches. Too much *Spin Time*, however, can reflect lost opportunity for productive work. Note that the *Overhead and Spin Time* in the following figure is a sum over all threads. While the brown and green colors refer to CPU and Wait time accordingly,

as in the previous figure, the orange color refers to the sum of the Overhead and Spin time.

We expect that as we increase the number of threads leads to decreased runtime and increased *Overhead and Spin Time*. However the *Wait Time* is increased among threads while the study meets the saturation point, i.e the number of threads is equal to the number of physical processors. Increasing more threads from that point, more *Wait Time* is added to the execution timeline of every thread, due to the increased *Overhead and Spin Time*. This leads to negligible speedup gains from this point on, which also verifies the theoretical background.

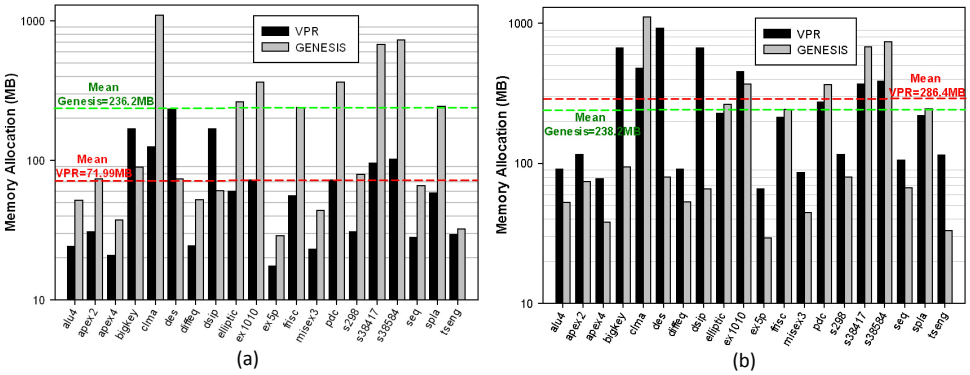


Figure 4.16 Memory requirements for (a) minimum FPGA size and (b) double size for the FPGA array.

Finally, we evaluate the memory footprint efficiency of the GENESIS placer in comparison to the VPR one, by increasing the number of slices of the underlying FPGA. While the VPR tool allocates memory resources firmed to the size of the underlying FPGA device, the proposed GENESIS algorithm depends only to the number of nodes found in the application's netlist, thus being independent from the actual FPGA size. Two different scenarios are studied, to show the impact of the size of underlying FPGA device. More specifically, scenario 1 (depicted in Fig. 4.16(a)) corresponds to application mapping onto the minimum FPGA device (depicted in last column of Table 4.3), whereas the array size for second scenario (Fig. 4.16(b)) is the double compared to scenario 1. The selection of scenario 2 was performed in order to show the scalable behavior of GENESIS framework in respect to the FPGA size (i.e. number of FPGA slices). In case of small FPGA devices (scenario 1 in Fig. 4.16(a)), the average memory footprint of the GENESIS framework is around 236MB, which is consistently higher than the memory required from the VPR engine (average around 72MB). However, in the case of larger FPGAs (scenario 2 in Fig. 4.16(b)), the memory footprint of the GENESIS placer remains almost constant, i.e. average around 238 MB, while for VPR the average memory consumption explodes around to 286MB. Thus moving towards larger FPGA devices, our solution imposes an average overhead of 8%, while the corresponding memory requirements for VPR tool leads to a penalty of 398%. Figure 4.17 shows the average memory allocation trend for the employed benchmarks, regarding the FPGA array size, in the case

of previously studied scenarios, i.e. minimum FPGA size (Fig. 4.17(a)) and double FPGA size (Fig. 4.17(b)). It is evident that the GENESIS placer express a relaxed scaling behavior, compared to VPR tool, especially when the FPGA free slices are doubled.

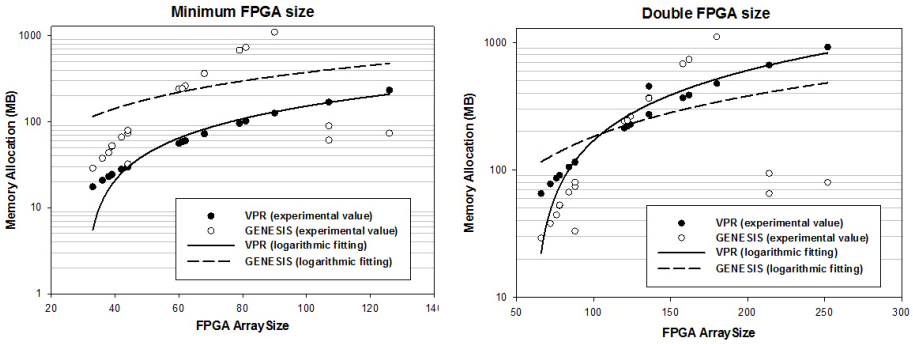


Figure 4.17 Memory footprint trend towards FPGA slices for (a) minimum FPGA size and (b) double size for the FPGA array.

4.6. Conclusion

This chapter presented GENESIS, a fast and effective parallel placement engine for island-based FPGA architectures. Through GENESIS, we proposed the usage of genetic algorithms extended by the concepts of mixability, to improve the quality of the derived placement solutions. A thorough discussion on the internal structures, definitions and operators developed within GENESIS for tailoring the genetic algorithm to the problem of application placement, has been provided. In addition, we showed that through a divide-and-conquer approach, effective coarse-grained parallelization can be applied to the proposed placement engine. The parallelization can greatly benefit the speedup of the placement engine on modern multi-core processors, thus enabling designers to study trade-offs between improved quality versus tool's runtime. For improved quality, an exploration methodology has been proposed for customizing the GENESIS's parameters in an application specific manner. The quality of derived design solutions as well as the efficiency of the proposed framework have been evaluated through extensive experimentation. Experimental results show that in comparison with the state-of-art placer of the VPR framework, GENESIS is able to perform application placement either up to 67× faster, or to deliver circuit solutions of 1.16× higher operation frequency, while sustaining a scalable behavior regarding both memory and computational resources requirements.

References

[1] ITRS, *International technology roadmap for semiconductos,* (2012).

[2] C. C. Wang and G. G. Lemieux, *Scalable and deterministic timing-driven parallel placement for fpgas,* in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '11 (ACM, New York, NY, USA, 2011) pp. 153–162.

[3] H. Sidiropoulos, K. Siozios, P. Figuli, D. Soudris, and M. Hubner, *On supporting efficient partial reconfiguration with just-in-time compilation,* in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International* (2012) pp. 328–335.

[4] R. Tessier, *Fast placement approaches for fpgas,* ACM Trans. Des. Autom. Electron. Syst. **7**, 284 (2002).

[5] J. M. Emmert and D. Bhatia, *Tabu search: Ultra-fast placement for fpgas,* in *Proceedings of the 9th International Workshop on Field-Programmable Logic and Applications*, FPL '99 (Springer-Verlag, London, UK, UK, 1999) pp. 81–90.

[6] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli, *A parallel simulated annealing algorithm for the placement of macro-cells,* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **6**, 838 (1987).

[7] A. Choong, R. Beidas, and J. Zhu, *Parallelizing simulated annealing-based placement using gpgpu,* in *Field Programmable Logic and Applications (FPL), 2010 International Conference on* (2010) pp. 31–34.

[8] J. Rose, W. Snelgrove, and Z. Vranesic, *Parallel standard cell placement algorithms with quality equivalent to simulated annealing,* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **7**, 387 (1988).

[9] A. Ludwin, V. Betz, and K. Padalia, *High-quality, deterministic parallel placement for fpgas on commodity hardware,* in *Proceedings of the 16th International ACM/SIGDA Symposium on Field Programmable Gate Arrays*, FPGA '08 (ACM, New York, NY, USA, 2008) pp. 14–23.

[10] S. Kravitz and R. Rutenbar, *Placement by simulated annealing on a multiprocessor,* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **6**, 534 (1987).

[11] M. G. Wrighton and A. M. DeHon, *Hardware-assisted simulated annealing with application for fast fpga placement,* in *Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays* (ACM, 2003) pp. 33–42.

[12] E. Witte, R. Chamberlain, and M. Franklin, *Parallel simulated annealing using speculative computation,* Parallel and Distributed Systems, IEEE Transactions on **2**, 483 (1991).

[13] G. Smecher, S. Wilton, and G. Lemieux, *Self-hosted placement for massively parallel processor arrays,* in *Field-Programmable Technology, 2009. FPT 2009. International Conference on* (2009) pp. 159–166.

[14] V. Betz and J. Rose, *Vpr: A new packing, placement and routing tool for fpga research,* in *Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications*, FPL '97 (Springer-Verlag, London, UK, UK, 1997) pp. 213–222.

[15] V. Betz, J. Rose, and A. Marquardt, eds., *Architecture and CAD for Deep-Submicron FPGAs* (Kluwer Academic Publishers, Norwell, MA, USA, 1999).

[16] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli, *Convergence and finite-time behavior of simulated annealing,* in *Decision and Control, 1985 24th IEEE Conference on* (1985) pp. 761–767.

[17] D. Diamantopoulos, K. Siozios, S. Xydis, and D. Soudris, *A framework for supporting parallel application placement onto reconfigurable platforms,* in *Proceedings of the PARMA Workshop, HiPEAC Conference, Berlin, Germany* (2013).

[18] W.-J. Sun and C. Sechen, *A loosely coupled parallel algorithm for standard cell placement,* in *Computer-Aided Design, 1994., IEEE/ACM International Conference on* (1994) pp. 137–144.

[19] A. Ludwin and V. Betz, *Efficient and deterministic parallel placement for fpgas,* ACM Trans. Des. Autom. Electron. Syst. **16**, 22:1 (2011).

[20] N. Selvakkumaran and G. Karypis, *Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization,* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **25**, 504 (2006).

[21] C. Ababei, *Speeding up fpga placement via partitioning and multithreading,* Int. J. Reconfig. Comput. **2009**, 6:1 (2009).

[22] Y. Sankar and J. Rose, *Trading quality for compile time: Ultra-fast placement for fpgas,* in *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, FPGA '99 (ACM, New York, NY, USA, 1999) pp. 157–166.

[23] P. Jamieson, *Revisiting genetic algorithms for the fpga placement problem.* in *GEM* (Citeseer, 2010) pp. 16–22.

[24] H. Sidiropoulos, K. Siozios, and D. Soudris, *A methodology and tool framework for supporting rapid exploration of memory hierarchies in fpgas,* in *Field Programmable Logic and Applications (FPL), 2011 International Conference on* (2011) pp. 238–243.

[25] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, *SIS: A System for Sequential Circuit Synthesis*, Tech. Rep. UCB/ERL M92/41 (EECS Department, University of California, Berkeley, 1992).

[26] J. Lamoureux and S. Wilton, *On the interaction between power-aware fpga cad algorithms,* in *Computer Aided Design, 2003. ICCAD-2003. International Conference on* (2003) pp. 701–708.

[27] M. Srinivas and L. Patnaik, *Adaptive probabilities of crossover and mutation in genetic algorithms,* Systems, Man and Cybernetics, IEEE Transactions on **24**, 656 (1994).

[28] A. Livnat, C. Papadimitriou, J. Dushoff, and M. W. Feldman, *A mixability theory for the role of sex in evolution,* Proceedings of the National Academy of Sciences **105**, 19803 (2008), http://www.pnas.org/content/105/50/19803.full.pdf+html .

[29] P. Darwen and X. Yao, *A dilemma for fitness sharing with a scaling function,* in *Evolutionary Computation, 1995., IEEE International Conference on*, Vol. 1 (1995) pp. 166–.

[30] R. E. Smith and C. Bonacina, *Mating restriction and niching pressure: results from agents and implications for general evolutionary computation,* in *In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003* (Springer, 2003) pp. 1382–1393.

[31] C.-K. Ting, S.-T. Li, and C. Lee, *On the harmonious mating strategy through tabu search,* Information Sciences **156**, 189 (2003), evolutionary Computation.

[32] E. M. A. Ronald, *When selection meets seduction,* in *Proceedings of the 6th International Conference on Genetic Algorithms* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995) pp. 167–173.

[33] T.-P. Hong and H.-S. Wang, *A dynamic mutation genetic algorithm,* in *Systems, Man, and Cybernetics, 1996., IEEE International Conference on*, Vol. 3 (1996) pp. 2000–2005 vol.3.

[34] S. Hartmann, *A self-adapting genetic algorithm for project scheduling under resource constraints,* Naval Research Logistics (NRL) **49**, 433 (2002).

[35] S. Xydis, A. Bartzas, I. Anagnostopoulos, D. Soudris, and K. Pekmestzi, *Custom multi-threaded dynamic memory management for multiprocessor system-on-chip platforms,* in *Embedded Computer Systems (SAMOS), 2010 International Conference on* (2010) pp. 102–109.

[36] J. Singh, J. Hennessy, and C. Holt, *Application and architectural bottlenecks in large scale distributed shared memory machines,* in *Computer Architecture, 1996 23rd Annual International Symposium on* (1996) pp. 134–134.

[37] D. Jiang and J. Singh, *Scaling application performance on a cache-coherent multiprocessors,* in *Computer Architecture, 1999. Proceedings of the 26th International Symposium on* (1999) pp. 305–316.

[38] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, *An efficient k-means clustering algorithm: analysis and implementation,* Pattern Analysis and Machine Intelligence, IEEE Transactions on **24**, 881 (2002).

[39] D. Shepard, *A two-dimensional interpolation function for irregularly-spaced data,* in *Proceedings of the 1968 23rd ACM National Conference*, ACM '68 (ACM, New York, NY, USA, 1968) pp. 517–524.

[40] V. Kalenteridis, H. Pournara, K. Siozos, K. Tatas, N. Vassiliadis, I. Pappas, G. Koutroumpezis, S. Nikolaidis, S. Siskos, D. Soudris, and A. Thanailakis, *A complete platform and toolset for system implementation on fine-grain reconfigurable hardware,* Microprocessors and Microsystems **29**, 247 (2005).

[41] V. Zaccaria, G. Palermo, F. Castro, C. Silvano, and G. Mariani, *Multicube explorer: An open source framework for design space exploration of chip multiprocessors,* in *Architecture of Computing Systems (ARCS), 2010 23rd International Conference on* (2010) pp. 1–7.

[42] G. M. Amdahl, *Validity of the single processor approach to achieving large scale computing capabilities,* in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring) (ACM, New York, NY, USA, 1967) pp. 483–485.

5

Architectural Synthesis of Reconfigurable Many-Accelerator Systems

Recent IT application requirements have established the formation of new technological terms, in order to describe their characteristics. Among them, the terms "Big Data" and "Internet-of-Things" have prevailed and they describe the ever increasing demands of applications regarding their processing and communication data size. These requirements increase the processing complexity and thus the energy needs. At the same time, technological progress in the semiconductor industry offers greater density of processing resources by silicon surface, per technology node integration, marking a continuation of the Law of Moore [1–3] and less energy management capacity per silicon surface, marking the end of Dennard Law [4, 5]. The result from the opposition of two laws designated by the research and industrial community as the era of "Dark Silicon" [5–8]. Through this chapter, the thesis proposes new architectural standards for scalable computing solutions. These architectures targets the increase of the processing strength per power consumption unit. Along with the architectural templates, also there are introduced the respective programming environments. Chapter 5 is associated with the general methodology of the thesis presented in Section 1.2, on the contribution to the "Acceleration Datapath Synthesis" and "Synthesis Flow Optimization".

5.1. Architectural template and programming interface for M.A. systems

5.1.1. Introduction - Research motivation for M.A. systems

Breaking the exascale barrier has been recently identified as the next big challenge in computing systems. Several studies [9], [10] showed that reaching this

goal requires a design paradigm shift towards more aggressive hardware/software co-design solutions at the architecture and technology level. Recently, many-accelerator heterogeneous architectures have been proposed to overcome the utilization/power wall [11–13]. For instance, Microsoft Corp. showed that such many-accelerator systems on reconfigurable fabrics can accelerate portions of large-scale software [14], delivering 95% improvements in servers's throughput.

From an electronic design automation (EDA) perspective, high-level synthesis (HLS) tools are expected to play a central role [15] in enabling effective design of many-accelerator computing platforms. Raising the design abstraction layer, designers can now quickly evaluate the performance, power, area and cost requirements of a differing accelerator configurations, thus providing controllable system specifications with reduced effort over traditional HDL-based development flow.

From a hardware perspective, heterogeneous FPGAs are proven to form an interesting platform solution for many-accelerator architectures. Their inherent flexibility/programmability and their continuous scaling of hardware density enables the accommodation of several types of hardware accelerators in comparison with conventional ASICs. In [16], it has been shown that FPGAs can potentially provide orders-of-magnitude speedups over conventional processors for compute-intensive algorithms, with greatly reduced power consumption. In a similar to multi-core trend, many parallel datapaths can be built and programmed into the FPGA to increase performance.

However, in such diverse and large pool of accelerators the memory organization forms a significant performance bottleneck, thus a carefully designed memory subsystem is required in order to keep accelerator datapaths busy [17], [18]. In [17], the authors propose a many-accelerator memory organization that statically shares the address space between active accelerators. Similarly in [18], a many-accelerator architectural template is proposed that enables the reuse of accelerator memory resources adopting a non-uniform cache architecture (NUCA) scheme. Both [17] and [18] are targeting ASIC-like many-accelerator systems and they mainly focusing on the performance implications of the memory subsystem.

In this research work we investigate the impact of the memory-intensive nature of many-accelerator systems onto the scalability potential of MA architectures. A recent survey of eleven publicly available accelerators reveals that "an average of 69% of accelerator area is consumed by memory" [17]. Rapid starvation of the available on-chip memory leads in severe resource under-utilization of the FPGA, similar to the "Dark Silicon" concept of future many-core chips. In fact, modern FPGA CAD tools (both at the RTL or HLS-level) allow only static memory allocation, which dictates the reservation of the maximum memory that an accelerator needs, for the entire execution window. While static allocation works fine for a limited number of accelerators, it does not scale to a many-accelerator design paradigm. Figure 5.14 shows an exemplary study of the memory-induced "Dark Silicon", considering the resource demands for the K_{means} clustering algorithm[1] on the Virtex

[1]The same behavior is observed for the overall set of the evaluated applications exhibiting diverse resource utilization features.

Ultrascale XVCU190 FPGA device[2] when scaling the number of parallel accelerators. Figure 5.14 is annotated with two threshold values regarding to maximum resource count and maximum power budget (device TDP=125^{o}C), respectively. Considering an ambient temperature of 50^{o}C, the power-induced "Dark Silicon" manifests itself with an allocation scenario of 105 accelerators consuming around 20 Watts. As shown, memory induced "Dark Silicon" poses a stricter constraint in accelerators' count, i.e. up to 2.5× less accelerators, meaning that memory resources starve faster than power. The BRAM memory is the resource that saturates faster than the rest FPGA resources types (FFs, LUTs, DSPs), thus forming the main limiting factor of higher accelerator densities as well as generating large fractions of under-utilized resources.

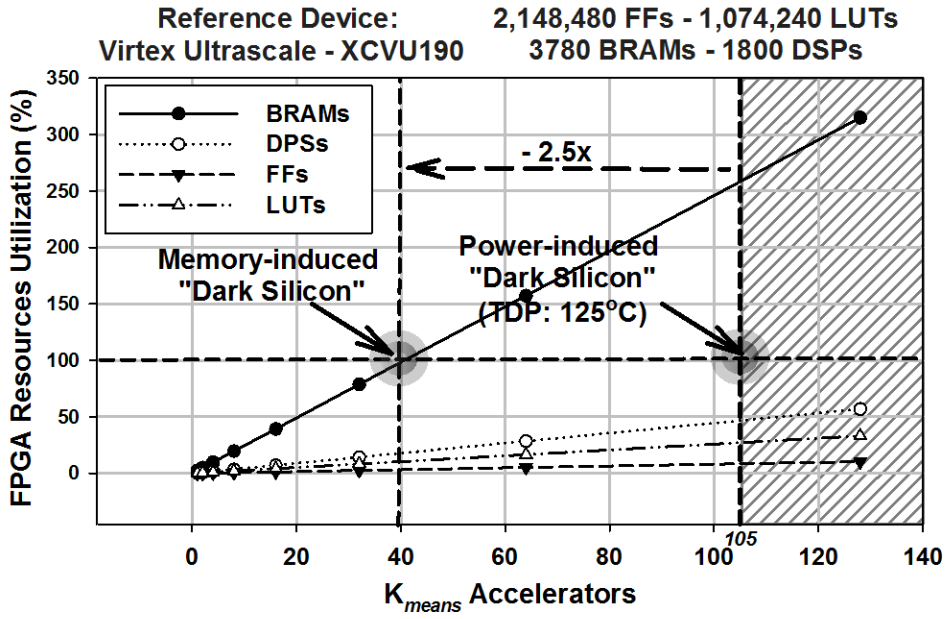


Figure 5.1 Accelerators scalability analysis of K_{means} clustering algorithm: A_i-Accelerators= $[1:128]$, N_p-Points=2×10^4, P_k-Clusters=3

In this research work, we target to alleviate the aforementioned memory-induced "Dark Silicon" problem by proposing the elimination of the pessimistic memory allocation forced by static approaches. The main contribution of our proposal is the introduction of a novel HLS-based design framework for many-accelerator platforms that adopts a dynamically allocated run-time memory model. The focus of the work is to show how DMM can be used for mitigating the memory-induced resource under-utilization problem in MA systems, rather than a detailed hardware description of the implemented DMM mechanisms. In typical HLS with static

[2]The FPGA with the highest on-chip block RAM, total size: 132.9Mb

memory allocation, if the accelerators' memory requirements exceeds the available on-chip memory resources, then the design becomes un-synthesizable, i.e. the designer should degrade system characteristics to match available resources. The proposed solution allows high accelerator densities by alleviating the resource under-utilization inefficiencies mainly induced by the static memory allocation strategies used in modern HLS tools. We show that applications with both dynamically and statically allocated data can be benefited from the proposed techniques, with the prerequisite of utilizing the proposed malloc/free interface for performing data allocation. For static applications this can be performed through a minimal and straightforward source code modification.

Prior art investigated specialized hardware, i.e. architecture templates for many-accelerator systems, as a response to dark silicon era. In [17], the authors propose a many-accelerator memory organization that only statically shares the address space between active accelerators. Similarly in [18], a many-accelerator architectural template is proposed that enables the reuse of accelerator memory resources adopting a non-uniform cache architecture (NUCA) scheme. Both [17] and [18] are targeting ASIC-like many-accelerator systems and they mainly focusing on the performance implications of the memory subsystem. Related to HLS prior art, to the best of our knowledge, only [19], [20] and [21] studied the dynamic memory allocation for high-level synthesis. However, they do not target many-accelerator systems, thus providing no support for mitigating the memory-induced under-utilization problem.

We propose the adoption of Dynamic Memory Management (DMM) techniques for relaxing the stringent constraints imposed by static memory allocation. Under static memory allocation, all decisions are taken at design-time. If the accelerators' memory requirements exceeds the available on-chip memory resources, then the design becomes un-synthesizable, i.e. the designer should degrade system characteristics to match available resources. However, this is not the case for dynamic allocation, which allocates memory according to accelerators' runtime requests and stalls the execution of accelerators whenever the memory requests exceeds the available resources. Figure 5.2 depicts such an exemplary scenario, where four accelerators with aggregated static memory requirements higher than the FPGA's available memory, can be finally synthesized and scheduled with the support of dynamic memory management. The key element is the incorporation of a management layer within system-level synthesis, e.g. HLS, that dynamically allocates memory according to accelerators' runtime requests and stalls the execution of accelerators whenever the memory requests exceeds the available resources.

Following the above discussion, we introduce the DMM-HLS framework that (i) extends typical HLS with DMM mechanisms and (ii) provides an HLS malloc/free API that enables statically allocated memory to be transformed to a dynamic one. We extensively evaluated the effectiveness of the proposed DMM-HLS framework over several many-accelerator architectures for representative applications of emerging computing domains. We show that DMM-HLS delivers more scalable MA platform configurations with an average 3.8× increment on the accelerator count in comparison to MA systems designed using state-of-art HLS. Better scalability leads also to

significant throughput gains under both private and shared memory model configurations, 24.1× and 3.8× in average, respectively.

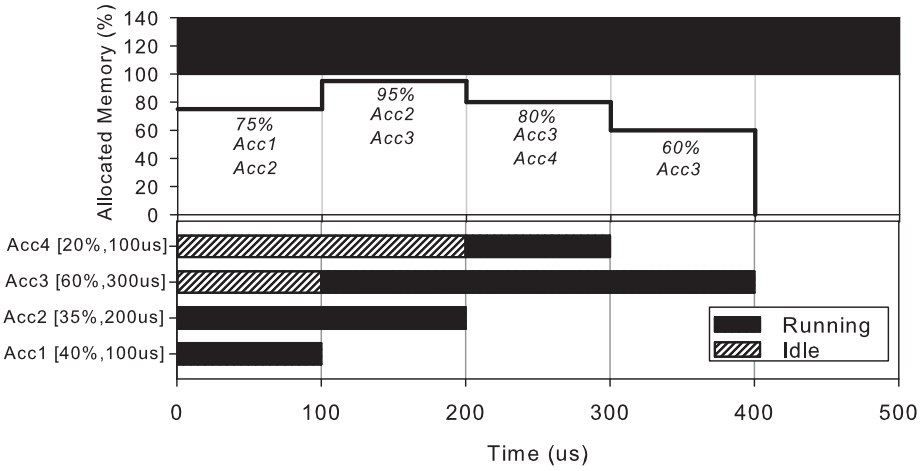


Figure 5.2 Example scenario with four accelerators *Acc-i[Static Memory Utilization%, Latency]*. All accelerators should start at time 0. The design is un-synthesizable with static memory allocation. Lower part shows the final scheduling with DMM. Upper part shows the respective memory footprint. Due to time-scale, we neglect showing instantaneous memory footprint transitions occurred whenever a new DMM allocation arrives.

5.1.2. DMM-HLS for Many-Accelerator FPGAs

Figure 5.15 shows a typical FPGA-based MA system. It includes i) the processor subsystem executing the application control flow and ii) the accelerators subsystem holding the computationally intensive kernels of the application. The accelerators are designed and synthesized through Vivado-HLS. The on-chip memory resources (BRAMs) are managed through the DMM-HLS framework. DMM-HLS supports the description of accelerators with both static and dynamic allocated data stored in BRAMs. It exposes a DMM API composed of two main function calls, similar to *glibc* malloc/free API, for memory allocation and deallocation.

- `void* HlsMalloc(size_t size, uint heap_id)`
- `void HlsFree(void *ptr, uint heap_id)`

, where **size** is the requested allocation size in bytes, **heap_id** is the identification number of the heap on which allocation shall occur and ***ptr** is the pointer which shall be freed up. A partitioning of accelerator's data as dynamic or static memory objects can performed by the designer after analysing the application's memory access traces. Without loss of generality, in this research work we adopt a data partitioning scheme in which global scope data structures, i.e. data structures

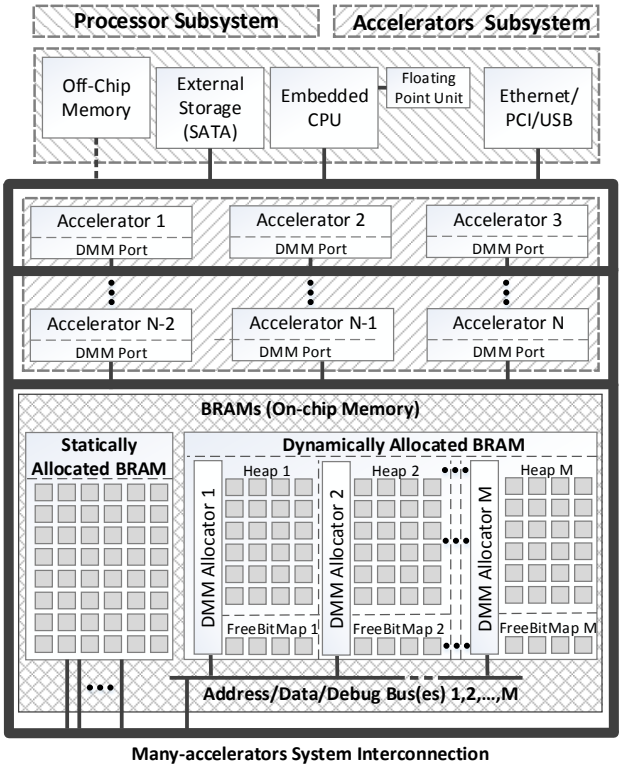


Figure 5.3 Abstract proposed architectural template for memory efficient many-accelerator FPGA-based systems.

that accelerators operate on upon invocation, are allocated as dynamic data, while accelerator's internal data structures, e.g. local register files etc. are allocated as static objects.

The DMM-HLS framework implements techniques from multi-threaded dynamic memory management [22], [23]. It supports parallel memory access paths, by grouping BRAM modules into memory banks, named heaps (Fig. 5.15). Each heap implements its own allocator consisting of two major hardware components, i) the free-list memory structure holding the freed and allocated memory blocks and ii) the fit allocation algorithm that searches over the free-list and allocates memory in a first fit manner. The maximum number of heaps controls the supported memory level parallelism of the dynamically allocated data. More than one accelerators can be bound to a specific memory heap for allocating data. The increased accelerator parallelism in combination with the overlapped execution offered by the multiple heaps DMM configurations delivers significant throughput gains when tasks of variable workload are co-scheduled on the FPGA. Figure 5.5 shows such a scenario,

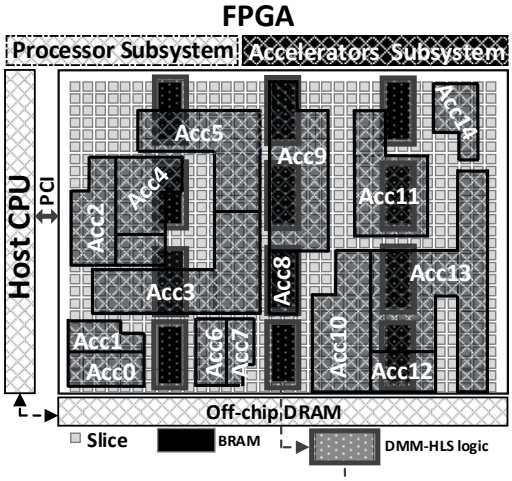
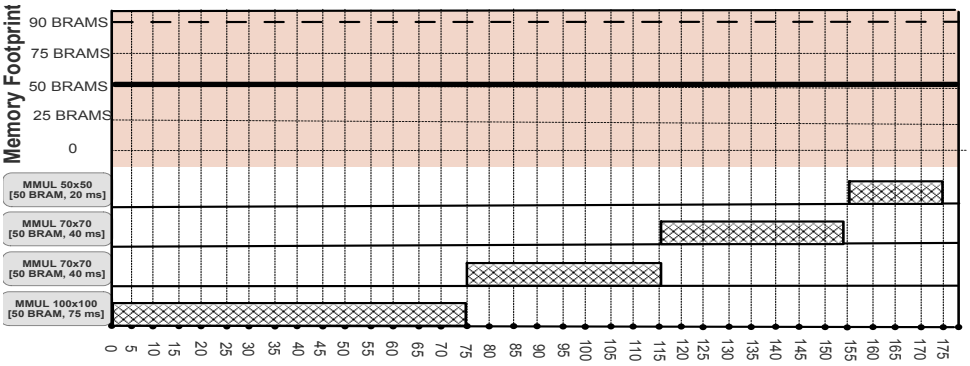


Figure 5.4 Proposed architectural template for memory efficient many-accelerator FPGA-based systems .

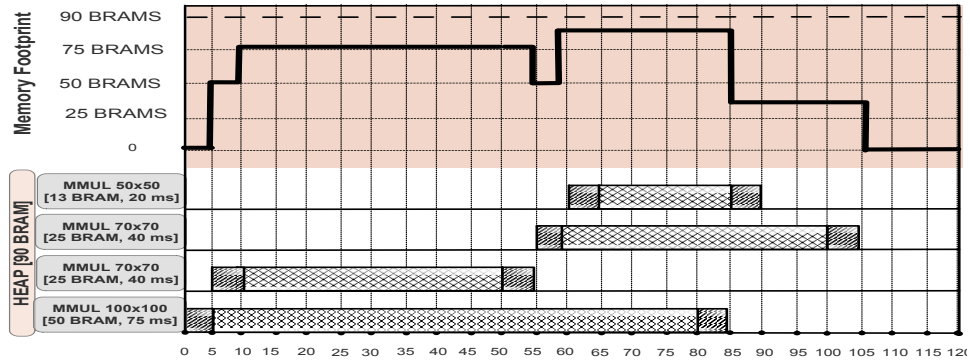
i.e. accelerator parallelism and overlapping, for the execution of 4 MMUL applications with differing workload characteristics onto a FPGA with maximum 90 BRAMs assuming a 2-heap DMM. It can be easily verified that the proposed solution delivers throughput gains of 42.8% over the static HLS solution that serializes tasks' execution.

Since a shared hardware interface of $HlsMalloc$ and $HlsFree$ limits parallelism when multiple accelerators request to allocate/free memory simultaneously, function inlining of the $HlsMalloc$ and $HlsFree$ is utilized, trading-off unconstrained parallel access on heaps with an increased resource occupation. Data access sequencing is performed over the BRAMs ports to eliminate data access conflicts during concurrent memory accesses.

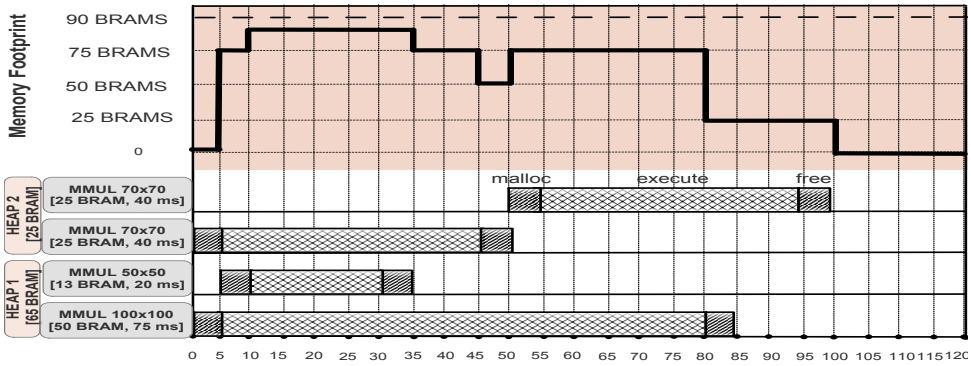
In the proposed DM organization, every heap implements its own allocator. It consists of two major hardware components, i) the free-list memory structure holding the freed and allocated memory blocks and ii) the fit allocation algorithm that searches over the free-list and allocates memory in a first fit manner. A FreeBitMap structure, i.e. bit-map free-list holding information about the free space inside this heap, tracks the occupied memory space. FreeBitMap, is an array of registers, every bit of which maps to a singe byte of heap. The management of the FreeBitMap structure is carefully performed through bit-wise operations to enable fast and efficient hardware synthesis. The term "maps" refers to the allocation status (allocated or free) of this byte. In general, every heap-i is highly parameterizable on a number of design options, most important of which are (i) the heap depth D_i^H, i.e. the total number of unique addresses, (ii) the heap word length L_i^H, i.e. the number of bytes of every single word of heap, (iii) the FreeBitMap depth D_i^F, i.e. the total number of



(a)



(b)



(c)

Figure 5.5 Performance gains due to parallel and overlapped accelerators' execution enabled by DMM-HLS. Accelerator scheduling and memory footprint a) Conventional HLS with static allocation, b) DMM-HLS with 1 single heaps, c) DMM-HLS with 2-heaps. FPGA platform: 90 BRAMs. Total MMULs' memory request: 113 BRAMs.

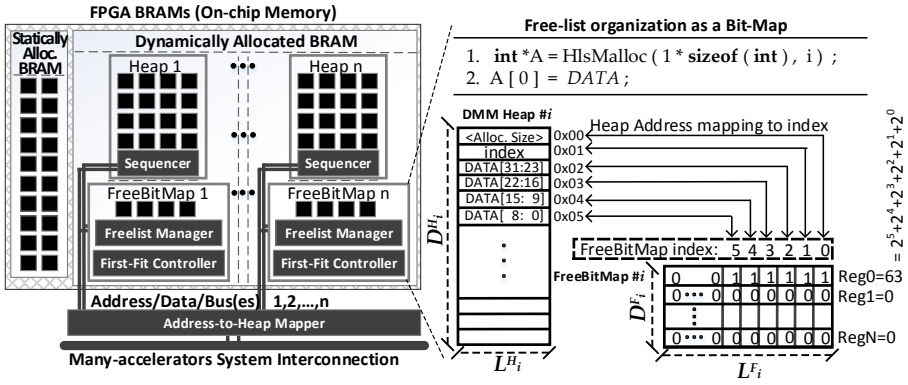


Figure 5.6 Architectural template of DMM-HLS memory controllers supporting DMM onto FPGAs.

free-list registers, (iv) the FreeBitMap word length L_i^F, i.e. the number of bytes of every single word of FreeBitMap, (v) the allocation alignment A_i, i.e. the minimum number of bytes per allocation so that every new allocation starts from a unique address and (vi) the meta-data header size H_i, i.e. the number of bytes reserved on first address(es) of every allocation to store meta-data related to the allocation, e.g. allocation length. This leads to a flexible design space per heap, that can be customized according to the memory allocation requests that each heap is going to serve. In the rest of this research work, we use the following configuration vector per heap: $[D_i^H, L_i^H, D_i^F, L_i^F, A_i, H_i] = [8192, 4, 1024, 4, 4, 2]$. The aforementioned architectural scheme of the proposed memory controllers of DMM-HLS framework is depicted in Figure 5.6.

There are three major runtime issues related to the DMM in many accelerator systems, i.e. *memory fragmentation*, *memory coherency* and *memory access conflicts*. In many accelerator systems, we recognize two fragmentation types, alignment and request fragmentation. *Alignment fragmentation* accounts for the extra bytes reserved for keeping every allocation padded to the heap word length L_i^H, including allocation size and header meta-data information. As long as the size of DMM requests (malloc/free) is multiple of L_i^H, the alignment fragmentation is zero. *Request fragmentation* refers to the situation that a memory request skips freed memory blocks to find a continuous memory space equivalent to the size of the request. In the worst case scenario the request cannot be served even if there are available memory blocks in the heap if they are merged. Request fragmentation is strongly dependent on the memory allocation patterns of each accelerator. In case of an homogeneous many-accelerator system, i.e. each accelerator allocates the same memory size, request fragmentation is zero. Regarding to *memory coherency* problems, they are inherently eliminated in DMM-HLS, since every accelerator has

its own memory space, thus no other accelerator may access it. However, *memory access conflicts* may become a performance bottleneck in case that a large set of accelerators share the same heap. As previously mentioned, DMM-HLS supports multiple-heap configurations that relax the pressure on the dynamic allocated memory space.

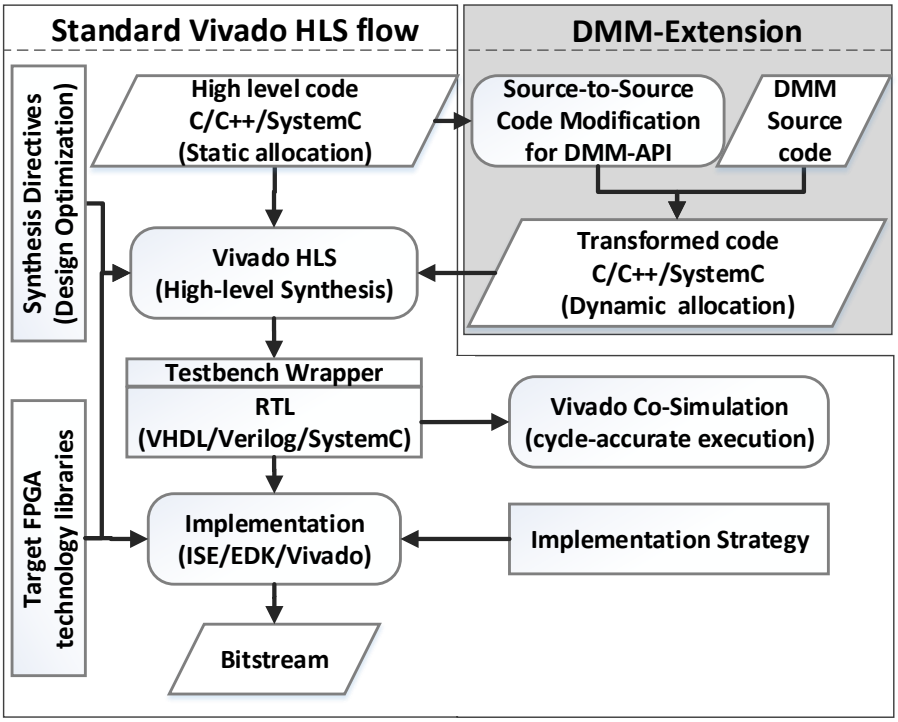


Figure 5.7 Extension on Vivado HLS flow to support dynamic memory management for many-accelerators FPGA-based systems.

The employed flow for the evaluation of DMM-HLS, which is depicted in Figure 5.17, is based on Xilinx Vivado-HLS, a state-of-art and industrial strength HLS tool, targetting the Virtex Ultrascale XVCU190 device. We evaluated the efficiency of the proposed DMM-HLS framework considering many-accelerator architectures targeting to emerging application domains, e.g. artificial intelligence, scientific computing, enterprise computing etc. Specifically, we used six applications (Table 5.2) found in Phoenix MapReduce framework for shared-memory systems [24]. We considered a set of 1000 tasks to be mapped onto each examined MA system. Task's memory size requirements derived by a normal distribution N($\frac{MaxSize}{2}, MaxSize$), where $MaxSize$ is defined in the last column of Table 5.2. The tasks are inserted to the MA system in the same time and scheduled whenever enough memory is available. In case of a tie, tasks with larger memory size requests are prioritized

For static applications, the original code is source-to-source transformed to a dynamically allocated one using specific function calls from the proposed DMM-HLS API. The transformed code is augmented by the DMM-HLS function calls and it is synthesized into RTL implementation through the back-end of Vivado HLS tool. An exemplary scenario of the code patterns triggering code transformations is given in Table 5.1.

Table 5.1 Exemplary scenario of the code patterns triggering code transformations, using DMM-HLS API.

Original Code	**Transformed Code for DMM-HLS**
	`const unsigned int T=500; /* check period */`
	`int *IN,*OUT;`
`int IN [10];`	`while((IN =HlsMalloc(10,0))==-1){HlsSleep(T)}`
`int OUT[10];`	`while((OUT=HlsMalloc(10,0))==-1){HlsSleep(T)}`
`foo(IN,&OUT);`	`foo(IN,&OUT);`
	`HlsFree(IN,0);`
	`HlsFree(OUT,0);`

Regarding to the execution stalling of an accelerator, when there is no available free memory in the heap, we utilize a while-loop wrapper around HlsMalloc call. In normal operation, when there is available free memory on a specific memory chunk of the heap, the allocator returns the first address of this memory chunk. Whenever there is no free memory available in the heap, the allocator returns -1. Using the while-loop wrapper, we force the stalling of the accelerator at the HlsMalloc call. At the same time other accelerators may be executed in parallel and eventually free some memory. Until then, the stalled accelerator repetitively checks for available free space through the while-loop wrapper. In order to avoid extra allocator's activity due to repetitive unsuccessful memory requests from stalled accelerators, we employ an appropriate synthesizable $HlsSleep$ function of T-cycles delay. According to the activity frequency of allocator and size/frequency of accelerator's requests for DMM, the T-variable may be either increased to reduce allocator's activity or decreased in order to make accelerator more responsive to stalled accelerators, in the case of a newly freed memory chunk.

5.1.3. Evaluation

Table 5.2 Applications Characterization

Application Domain	**Kernel**	**Description**	**Parameters**
Image Processing	Histogram	Determine frequency of RGB channels in image.	M_{size} = 640×480 pixels
Scientific Computing	Matrix Multiplication	Dense integer matrix multiplication.	M_{size} = 100×100
Enterprise Computing	String Match	Search file with keys for an encrypted word	$N_{file-keys}$ = 307,200, M_{wor}
Artificial Intelligence	Linear Regression	Compute the best fit line for a set of points.	N_{points} = 100,000
Artificial Intelligence	PCA	Principal components analysis on a matrix.	M_{size} = 250×250
Artificial Intelligence	K_{means}	Iterative clustering algorithm to classify n-D data points into groups.	N_{points} = 20,000, $P_{clusters}$ = 10, $n_{dimensions}$

In order to evaluate the per-accelerator latency overhead due to DMM mechanisms, Figure 5.8 depicts the normalized latency of the employed kernels for the case of 16 accelerators using 1,2,4,8 and 16 heaps. As shown, the average over-

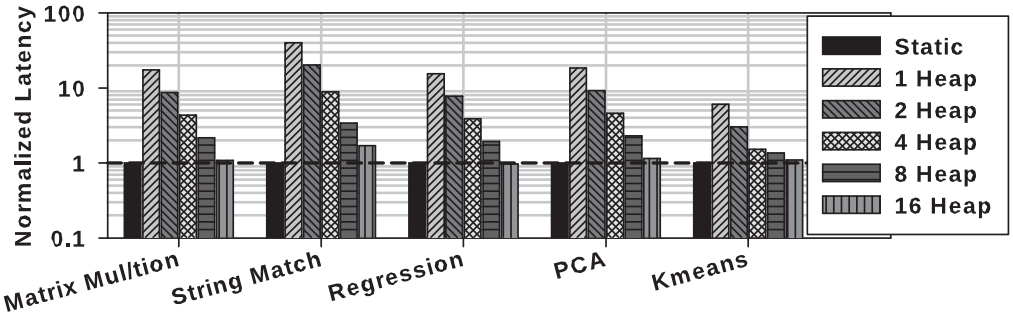


Figure 5.8 Per-accelerator latency overhead for different number of heaps.

head for all accelerators is 19.9× when only one heap is employed, while it drops to 10×, 4.7×, 2.3× and 1.2× as long as the heaps are doubled. The increase of heaps allows higher memory level parallelism to be achieved, since less accelerators are sharing the same heap. Yet, even in the case that every accelerator has its own unique heap, i.e. 16-accelerators - 16-heaps configuration, the latency overhead is still persist (1.2×), due DMM internal operation (freelist check, first-fit operation, etc.). However the DMM-HLS framework enables effective system configurations, which trade-off accelerator density and heap sharing.

We evaluate the practical performance improvements delivered by the proposed DMM-HLS framework in a twofold manner. First, we evaluate accelerators' density, i.e. the number of accelerators that can be programmed onto the FPGA simultaneously. Figure 5.9 depicts the accelerators gain/loss of DMM-HLS compared to static allocation. The bars express the min-max number of accelerators that can be loaded using 1:16 heaps. As shown, the proposed DMM-HLS framework is able to deliver many-accelerator architectures with 3.8× more accelerators in average (up to 9.7× for Histogram), compared to static allocation. The highest gains on accelerators' density come from the usage of a single heap configuration that delivers the least possible overhead regarding the resources consumed by the DM manager. As long as configurations with more heaps are adopted, the extra resources needed to implement the corresponding allocators decrease the maximum number of accelerators, e.g. the instantiation of 16 heaps delivers an average gain of 1.7× in accelerators' density.

We note that simply increasing the number of accelerators does not imply performance gains in a straightforward manner. Figure 5.10 shows the number of accelerators where the system exhibits maximal performance, in terms of throughput[3] Figure 5.10 shows normalized throughput over static for all employed applications.

[3]Throughput is calculated as the workload size (in Mbytes) over the latency of a kernel to process it (in us).

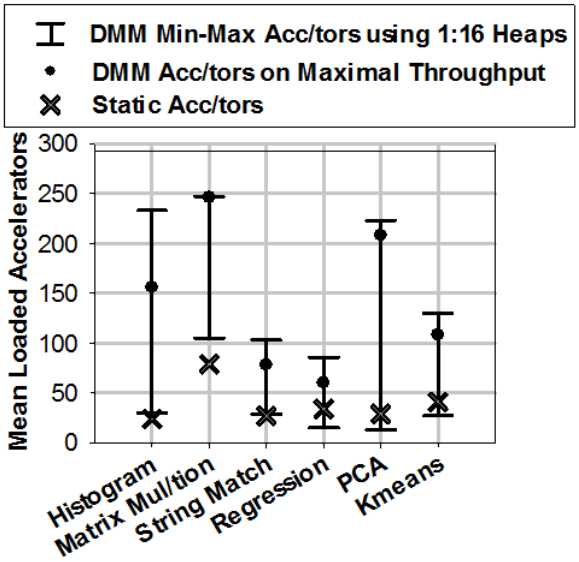


Figure 5.9 Comparison on accelerators density between Static and DMM-HLS setups.

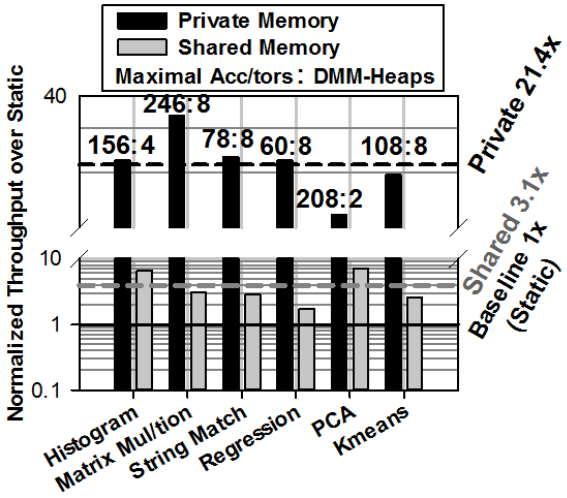


Figure 5.10 Comparison on system's throughput between Static and DMM-HLS setups.

The observed variations in throughput and accelerator density originates from the differing resource requirements and workload characteristics. We consider two use-case scenarios: the loaded accelerators are initialized with data from i) individual memory space i.e. private memory and ii) shared memory, i.e. the case that ac-

celerators are working on the same data, e.g. finding different strings on the same document with String Match algorithm. The results report the configuration (denoted as *accelerators number : heaps number* on top of every column) that delivers maximal throughput for each application. We measured an average throughput increase of 21.4× with private memory initialization and 3.1× with shared memory initialization over the static allocation of the conventional Vivado-HLS.

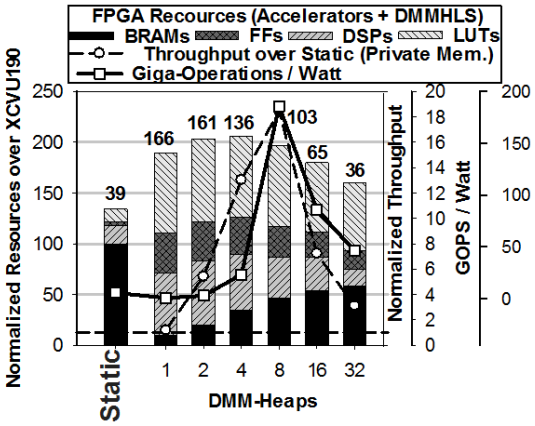


Figure 5.11 Comparison on resources breakdown versus throughput and energy trade-off, between Static and DMM-HLS setups.

Figure 5.11 shows aggregated averaged results in terms of number of accelerator, resource usage and throughput. We examine both static allocation and DM allocation with several heaps ranging from 1 up to 32, respectively. The left y-axis refers to the normalized resources[4]. The average accelerators' density (i.e. maximum number of deployed accelerators) for each examined configuration is reported on top of the stacked bar. The right y-axis refers to the normalized throughput over static allocation, which is highlighted as a dashed horizontal line. Following the same trend as throughput, energy efficiency is also reported in terms of Giga-Operations Per Second/Watt (GOPS/W). While the single heap configurations delivers the highest number of accelerators (3.8×), it exhibits low throughput since memory accesses are executed sequentially. By adding more heaps, the overall system exhibits higher throughput up to the point that the extra resources of multi-heap allocators cause the decrease in number of accelerators. Area overheads is due to the heap allocator modules and the extended interface of the accelerators. For single heap implementations an average area overhead of 0.3% FFs and 1.2% LUTs is reported in respect to the static implementations, which scales up to +10.7% FFs and +55.2% LUTs for the case of 32 heaps. However, considering both the DMM infrastructure and accelerators, the on-chip memory utilization is around 60%, i.e. other resources and not memory exceed the maximum limit,

[4]Every stacked vertical bar contains the cumulative percentage of the four studied resources (BRAMs, DSPs, FFs and LUTs), thus the theoretical maximum value of left y-axis, on the diagrams of Figure 5.11, is 400%.

while the rest FPGA resources are increased in average by a factor of 6.3×, 17, 6× and 29.7× for DSPs, FFs and LUTs respectively. Figure 5.12 depicts in detail the cumulative results for all the employed applications.

5.1.4. Conclusions

This research work targeted the scalability issues of modern many-accelerator FPGA systems. We showed that the on-chip memory resource impose severe bottlenecks on the maximum number of deployed accelerators, leading to large resource under-utilization in modern FPGA devices. We proposed the incorporation of dynamic memory management during HLS to alleviate the resource under-utilization inefficiencies mainly induced by the static memory allocation strategies used in state-of-art HLS tools. The proposed approach has been extensively evaluated over real-life many-accelerator architectures targeting to emerging applications, showing that its adoption delivers significant gains regarding to accelerators density and throughput improvements.

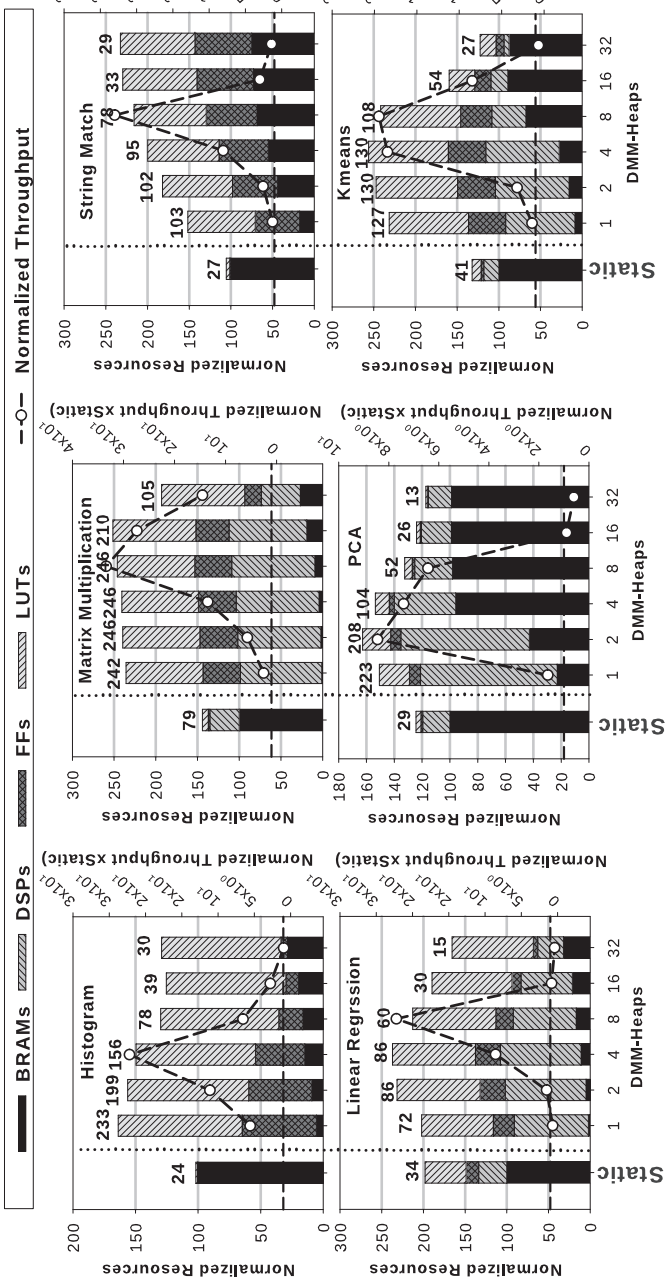


Figure 5.12 Comparison on resources breakdown versus throughput and energy trade-offs between Static and DMM-HLS setups, for all employed applications.

5.2. Scaling many-accelerator systems to workstations

5.2.1. Introduction

The previous section presented a scalable architecture of processing power, using many-accelerator architectures in FPGAs. The proposed framework was applied to computational intensive applications and proved its superiority compared to conventional methodologies and design tools. In this research work we analyze a similar approach for workstations. The purpose of this research work is to investigate the efficiency of coarse-grain accelerators systems in real-life workstation scenarios.

One of the main promising programming frameworks for processing large data sets in the workstations and other clusters of computers is the MapReduce framework [25]. MapReduce was firstly inspired by Google for application development on data-centers with thousands of servers. It allows programmers to write functional-style code that is automatically parallelized and scheduled in a distributed system.

MapReduce can be used to easily utilize the resources of large distributed systems for processing large data sets. However, such large data sets, that the data centers have to process under constrained time and energy budgets, have increased significantly, due to emerging applications like big data and cloud computing. The increase of the traffic in the data centers has also resulted to higher power consumption. The server processors need to provide higher throughput without consuming excessive power. Currently, one of the main challenges in the data center operators is the power consumption of the servers that account for over 45% of the overall power consumption in the data centers.

Therefore, novel architectures are required that can increase the performance of the data centers and also be more energy efficient. FPGAs can be utilized to increase the performance of the systems and also can be used to reduce the total power consumption due to the specialized accelerators for specific tasks. However, the main drawback of utilizing FPGAs in the data centers is the high programming complexity. In this research work, we present a novel framework that allow the seamless development of FPGA-based hardware accelerator for data centers by extending the current HLS toolflow to include the MapReduce framework. We develop several hardware accelerators for typical MapReduce application using the proposed framework and we compare the performance and the energy efficiency with high-end multi-core processors. The main contributions of this research work are the followings:

- a novel HLS-based MapReduce dataflow architecture,
- development of several hardware accelerators for typical MapReduce application based on the HLS-enabled MapReduce dataflow architecture
- performance evaluation on typical MapReduce applications (wordcount, histogram, etc.) on a Virtex7 FPGA that shows up to 4.3x throughput gains and
- up to two orders of magnitude energy consumption savings

The main advantage of the proposed scheme is that we utilize the available resources of the FPGA to achieve higher parallelism. In the terms of raw performance the proposed scheme is comparable to the General Purpose Processor (GPP), but by exploiting the parallelism and much lower clock frequency compared to GPP, we can achieve much lower power consumption. The rest of the research work is organized as follows. Subsection 5.2.2 presents our FPGA-based implementation. Subsection 5.2.3 presents the evaluation results and subsection 5.2.4 concludes the work.

We survey related work on accelerator-based implementations of MapReduce framework. In [26], a reconfigurable MapReduce framework is presented but the proposed scheme is implemented as a custom RTL-design that is used to implement only the RankBoost application entirely on an FPGA. Although the basic architecture of this work is very close to the one presented in this research work, we would like to note that the proposed approach is implemented in C/C++ level and it is seamlessly synthesized to RTL level using HLS tools, thus its employment is highly transparent to new applications exploiting MapReduce framework. On the contrary, in [26] both of the Map and Reduce tasks for a specific application are mapped to custom RTL logic and thus for any new application a new design has to be implemented. In [27] a MapReduce Framework on FPGA accelerated commodity hardware is presented where a cluster of worker nodes is designed for the MapReduce framework, and each worker node consists of commodity hardware and special hardware. Although this approach offers high flexibility and run-time optimization of the framework, it still increases the programming difficulty of both custom-RTL and CPU for every node, while new applications have to be custom tailored to such a diverse hybrid-node implementation layer. Regarding general purpose GPU platforms, MapReduce framework was also explored [28]. "However, GPU prefers coalesced memory access pattern, which makes it fumble while dealing with complex data structure and the SIMT architecture restricts its computation performance to handle irregular applications" [26]. Authors in [29] adopted a hybrid architecture approach combining both GPU and FPGA to implement a MapReduce framework, which leaves scheduling work to the host CPU and employs GPU and FPGA for co-processing, while in [30] a MapReduce framework is implemeted targeting to an embedded many-core Network-on-Chip platform. However, regardless the implementation medium, a recent MapReduce survey verifies that MapReduce technique shall complement database management system with scalable and flexible parallel processing for various data analysis such as scientific data processing [31]. In [32] Microsoft has used FPGAs to increase the performance of the page ranking applications. More specifically a medium-scale deployment on a bed of 1,632 servers, measuring its efficacy in accelerating the Bing web search engine, reported improvements on the ranking throughput of each server by a factor of 95%. However, until now the utilization of FPGAs in data centres is limited mainly due to the high programming complexity of FPGAs. However this work is a standalone research and currently there is no wide adoption of FPGAs in the data centers.

In [33], we have previously developed a hardware acceleration unit for the MapReduce framework that can be combined efficiently with ARM cores in fully

programmable platforms . To develop and evaluate the proposed scheme, authors selected the Xilinx Zynq-7000 All Programmable SoC, which comes hardwired with a dual-core Cortex-A9 processor on-board.

In essence, this accelerator was used to alleviate the processors from executing the Reduce tasks, and thus executing only the Map tasks and emitting the intermediate key/value pairs to the hardware acceleration unit that performs the Reduce operation. The performance evaluation shows that the proposed accelerator can achieve up to 1.8x system speedup of the MapReduce applications. Motivated by these results, we identify the following performance-boost limitations in this work:

- Low parallelism exploitation: The partitioner, mapper and scheduler are controlled by the same on-board CPU, i.e. ARM Cortex-A9 on Zynq, which may limit the inherent parallelization opportunity of these tasks.
- High memory conflicts: Although the *reduce* tasks may exploit DMA engines to work directly to memory, still the main framework, including both the MapReduce tasks and the working kernels, have their data-path on the same AXI bus, thus the memory is over-populated by read/write (R/W) calls from the multiple processing threads, which eventually are serialized.
- Low acceleration opportunity regarding overall system execution time breakdown, due to the speed-up of only Reduce step. According to [24], the Reduce step accounts for less than 5% of the total execution time in the original implementation of a MapRecude framework, i.e. Phoenix, using a chip multi-processor (CMP) with 2, 4 and 8 cores[5]. The Figure 5.13 presents the execution time breakdown between Map, Reduce, and Merge tasks for the CMP system. It is evident that the execution time of Map task, where the main algorithm's processing occurs, is significantly reduced by the Phoenix framework, as long as more cores are employed, which motivate us to investigate acceleration scenarios for map tasks on the coarse-grain parallelization potential of FPGA devices.

Identifying these issues, we propose the complete decoupling of MapReduce's tasks data-paths to distinct buses, accessed from individual processing engines, eliminating the necessity of the supervisor on-board CPU, i.e. the processor-centric SoC. Such an approach implies a holistic C/C++ to RTL-level domain-level MapReduce transition. In this work, we employ HLS tools as a state-of-art system-level implementation toolflow, in order to examine the performance exploitation options, yet constrained by the HLS limitations of such a complex framework.

5.2.2. HLSMapReduceFlow Architecture

5.2.2.1 Phoenix MapRecude Framework

We adopt the open-source Phoenix MapRecude framework [24] as the initial code base of our work. In this framework, users specify a Map function that processes

[5]The referred CMP system is based on the UltraSparc T1 multi-core chip with 8 multithreaded cores sharing the L2 cache.

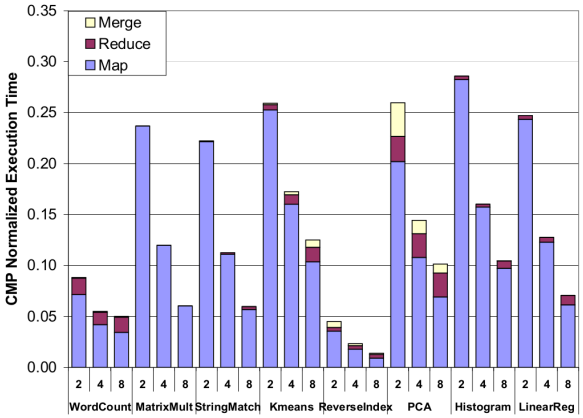


Figure 5.13 Execution time breakdown for a CMP system on Phoenix MapRecude framework[24].

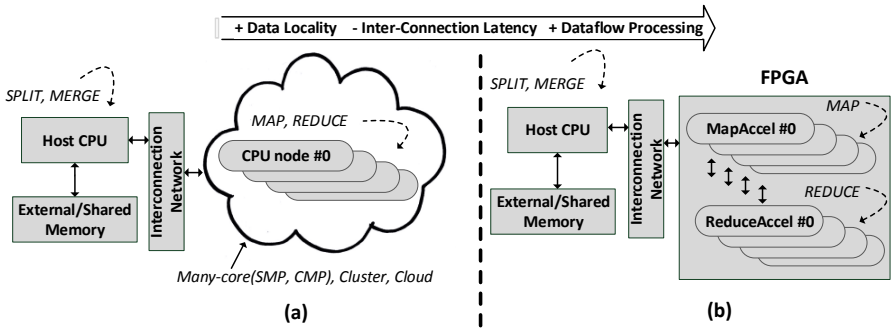


Figure 5.14 Architecture topology of a) original MapRecude framework and b) proposed HLSMapReduce-Flow.

a key/value pair to generate a set of intermediate key/value pairs, and a Reduce function that merges all intermediate values associated with the same intermediate key. Finally, the last stage merge together all the key/value pairs. Programs written in this functional style are automatically parallelized and executed on a large cluster of computation nodes. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of computation nodes, providing fault-tolerance, and managing the required inter-machine communication. This allows programmers without any experience of parallel and distributed systems, to easily utilize the resources of a large distributed system.

The Figure 5.15 shows the basic data flow for the runtime system. The runtime is controlled by the scheduler, which is initiated by user code. The scheduler creates and manages the threads that run all Map and Reduce tasks. It also manages the buffers used for task communication. After initialization, the scheduler determines the number of cores to use for this computation. For each core, it spawns a worker

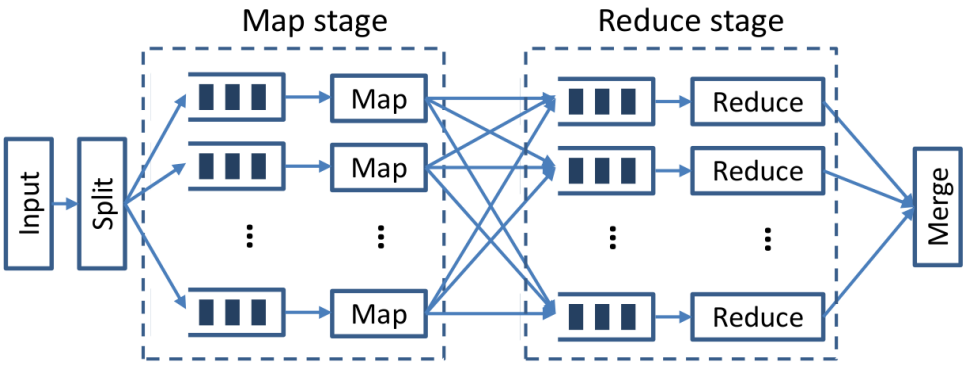


Figure 5.15 The MapReduce programming framework.

thread that is dynamically assigned some number of Map and Reduce tasks.

To start the Map stage, the scheduler uses the Splitter to divide input pairs into equally sized units to be processed by the Map tasks. The Splitter is called once per Map task and returns a pointer to the data the Map task will process. The Map tasks are allocated dynamically to workers and each one emits intermediate $< key, value >$ pairs. The Partition function splits the intermediate pairs into units for the Reduce tasks. The function ensures all values of the same key go to the same unit. Within each buffer, values are ordered by key to assist with the final sorting. At this point, the Map stage is over. The scheduler must wait for all Map tasks to complete before initiating the Reduce stage. Reduce tasks are also assigned to workers dynamically, similar to Map tasks. The one difference is that, while with Map tasks we have complete freedom in distributing pairs across tasks, with Reduce we must process all values for the same key in one task. As the last step, the final output from all tasks is merged into a single buffer, sorted by keys.

5.2.2.2 Dataflow FPGA-based Acceleration

The basic proposed architecture scheme is inspired by our group's prior novel implementation, presented in [33]. Authors developed a MapReduce configurable accelerator which is used to alleviate the processors from executing the Reduce tasks, and thus executing only the Map tasks and emitting the intermediate key/-value pairs to the hardware acceleration unit that performs the Reduce operation. On top of this architecture we further built the acceleration infrastructure for the Map tasks. Figure 5.14 shows a high-level differentiator of the proposed architecture, compared to the current state of art.

Originally, the Map and Reduce tasks are running as software threads on the CPU cores of the available MapRecude deploying infrastructure (Figure 5.14(a)). The success of such an architecture relies on the availability of a large shared-memory that facilitate communication without excessive data copying. Moreover, at runtime, an efficient scheduler is highly required to schedule tasks dynamically across the available processors in order to achieve load balance and maximize task

throughput. In the previous approach we highlight two major drawbacks:

- A shared-memory is usually available in monolithic datacenter architectures. Such architectures are composed by single-board/single-die many-core systems (CMP, SMP). However, there is limited shared-memory organization support for clusters and cloud datacenter topologies. Thus in case of inefficient data splitting, the shared data have to travel along different datacenter nodes, reserving resources and spending energy. Such an inefficiency in data splitting procedure may be the case in which the same subset of data is required by two Map steps which have been scheduled in long-distance computation nodes.
- So far, the state-of-art MapReduce implementations do not provide an efficient scheduler that checks the inter-application control and data flow graphs, prior to scheduling. Thus, it turns out that there is not real application partitioning and scheduling among computation nodes, but rather a quick-and-dirty application's runtime slicing, followed by a first-come-first-serve distribution on computation nodes.

We propose to create customized Map accelerators that exploit high data locality and thus eliminate the need of large shared-memory architectures or distributed systems. Instead of brute-force arbitrary splitting the input data to multiple subsets for further scheduling to CPUs, we select to split the input according to the application's data processing flow, in a way that optimized chunks of data are processed independently by distinct accelerators. Using this approach we manage to increase the system's throughput by a) increasing data locality, b) decreasing inter-connection latency among computation nodes and c) increasing computation parallelism by exploiting dataflow processing.

Specifically, we investigate the optimal point of dataflow processing for every application, i.e. splitting and scheduling is based on control-flow-graph (CFG), data-flow-graph (DFG) and variable liveness analysis (LA). Based on such information, we built the corresponding optimal Map accelerator engines. Figure 5.16(a) shows the basic HLSMapReduceFlow architecture. While, this looks similar to the original Phoenix architecture, we highlight in Figure 5.16(b) the novel architecture modifications of our approach. Firstly, the on-board available block RAM (BRAM) of the FPGA is organized in distinct memory banks. Every bank has its own unique address and data bus, while it is accessed by only one computation node. This scheme allows for full parallel simultaneous operation of the computation nodes in FPGA.

The critical step of this procedure relies on the efficient mapping of application's parallel-ready computation paths. For this step we employ the Vivado HLS tool. Apart from typical high level synthesis steps, i.e. resource binding, scheduling etc., Vivado HLS also provides a high number of architecture exploration options through the source code annotation with special pre-processor directives. In this work we force the exploration with the *DATAFLOW, INLINE* and *ARRAY PARTITION* directives.

Firstly we employ the partition, map and reshape directives in order to reconfigure arrays on the interface they are accessed. Arrays are partitioned into

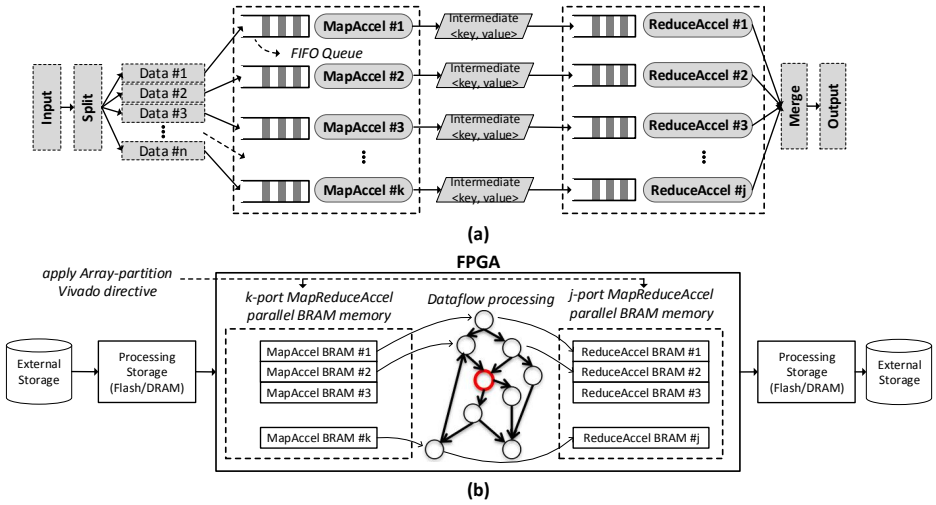


Figure 5.16 HLSMapReduceFlow dataflow architecture: Every dataflow computation node is working in its unique memory. The system memory is partitioned to *k-port* and *j-port* banks for the *k*-map and *j*-reduce tasks respectively.

multiple smaller arrays, each implemented with its own interface. This includes the ability to partition the array into fine grain elements. On the function interface, this results in a unique port for every element in the array. This provides maximum parallel access, but creates many more ports and may introduce routing issues in the hierarchy above. By partitioning the arrays, on which input data of every map task are stored, we reduce the possibility of simultaneous access of the same data, given the inherent locality of the application, which may exploit parallelism. Locality is managed by adjusting the granularity and assignment of parallel tasks.

After having partitioned the input memory, we force the micro-architecture exploration within Vivado HLS, following a dataflow computation model. From the definition back in 80's [34], we consider dataflow machines to be all programmable computers of which the hardware is optimized for fine-grain data-driven. Fine grain means that the processes that run in parallel are approximately of the size of a conventional machine code instruction. We deploy a fully spatial architecture for every map task by applying recursive inline option of Vivado HLS, i.e. *#pragma AP inline recursive*. Although this approach leads to increased resources utilization, it allows for parallel instances of shared sub-functions and removed hierarchy of sub-functions, which leads to logic optimization across function boundaries and improved latency/interval by the reduction of function call overhead.

After the above optimizations, we have already forced the creation of fine-grain fully-parallel map tasks which does not share neither data nor computation elements among them. The last optimization of the proposed scheme deals with the controlling of the the way the input data are fed to these tasks. We force a dataflow approach. Figure 5.17 shows the basic idea behind this approach. The input code

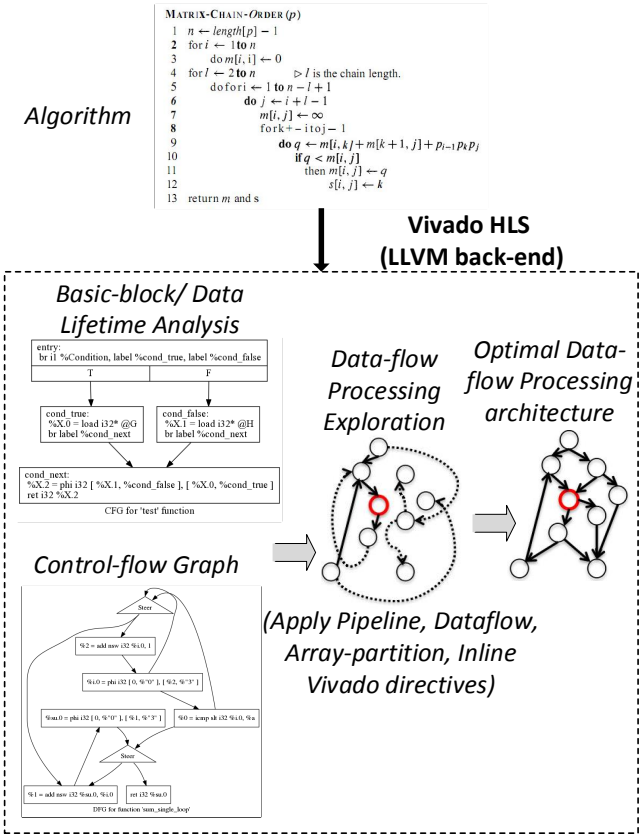


Figure 5.17 Forcing dataflow exploration from control-flow algorithm description with Vivado HLS

is decomposed by Vivado's back-end LLVM compiler to basic blocks, i.e. single-entry single-exit section of code, connected through a control-flow network, i.e. control-flow-graph (CFG). Having already applied above optimizations, we further force the dataflow optimization, i.e. *#pragma AP dataflow* which takes a series of sequential tasks (functions and or loops) (Figure 5.18(a)) and creates a parallel process architecture from it (Figure 5.18(b)). Dataflow optimization in Vivado HLS is a very powerful method for improving design throughput. The channels shown in Figure 5.18(a) ensure a task is not required to wait until the previous task has completed all operations before it can begin. Figure 5.18(b) shows how DATAFLOW optimization allows the execution of tasks to overlap, increasing the overall throughput of the design and reducing latency.

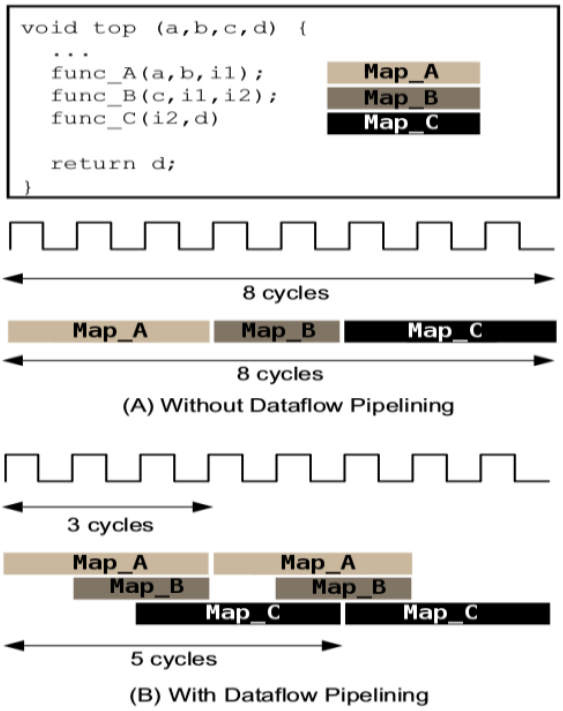


Figure 5.18 (a) Sequential Functional Description (b) Parallel Process Architecture

5.2.2.3 HLSMapReduceFlow Methodology for Vivado-HLS

Figure 5.19 shows an overview of the proposed HLSMapReduceFlow design and verification flow. The flow is based on Xilinx Vivado-HLS, a state-of-art and industrial strength HLS tool. The HLSMapReduceFlow extension is applied explicitly to the high-level source code of the application, thus it keeps minimum implementation overhead to the designers. A source-to-source code modification stage is the step where the original code is transformed to synthesizable one. These transformations cover limitations regarding the lack of dynamic memory management support, pointer arithmetic, complete ANCI C functions etc, in Vivado HLS. Moreover this step includes the process of architecture optimization directives insertion. Currently, this step is performed manually. An automated flow is considered a highly useful utility for wide and transparent adoption in data centers deployment. The transformed code is augmented by the HLSMapReduceFlow function calls, i.e. *Emit_Intermediate_accelerator(key,value)* and it is synthesized into RTL implementation through the back-end of Vivado HLS tool.

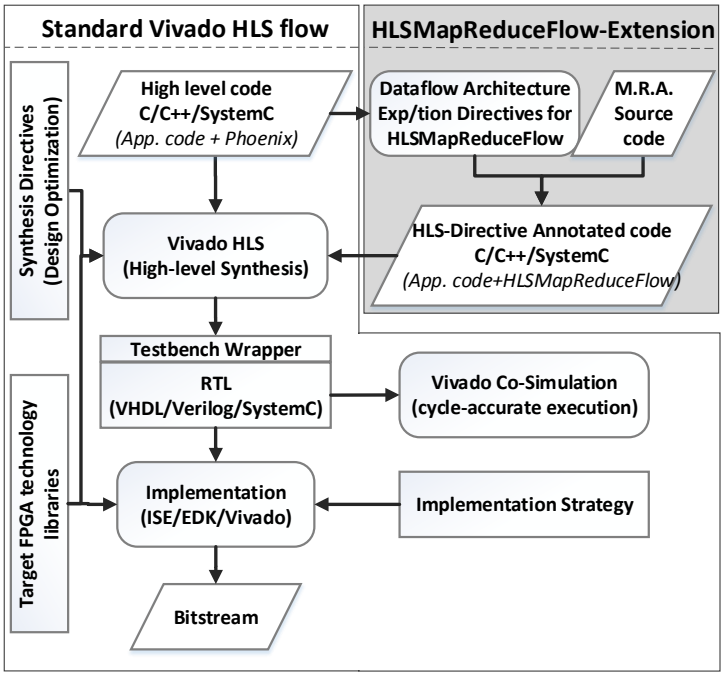


Figure 5.19 Proposed extension on Vivado HLS flow to support MapReduce framework for FPGA-based systems.

5.2.2.4 Vivado-HLS Limitations for MapRecude

During the development of HLSMapReduceFlow we faced several limitations regarding the implementation of the complex Phoenix's API in Vivado HLS. These limitations are reported as follows:

- **Dynamic Memory Management**: The Phoenix framework highly uses malloc/free calls for effective memory operations and reduced run-time footprint during map and reduce tasks. All DMM functions are not supported by Vivado HLS tools. We replace DMM calls with static code allocation on the heap of every application's code segment. This replacement affects the BRAM resource utilization, while it also forces the predefined at compile time variable definition. When the application uses dynamic size for specific variables, e.g. the length of word that is searched in Word Count application, then the designer has to set a static maximum variable's size, thus decreasing runtime flexibility.
- **Pointer Manipulation**: The Phoenix framework uses direct memory addressing, using pointer-based memory access. Also it uses arithmetic operations, arithmetic re-interpretation, and pointer casting. However, none of these features is available in Vivado HLS. We had to refactor the code by

Table 5.3 Applications Characterization

Kernel	Description	Parameters	Bytes/Iteration
Histogram	Determine frequency of image RGB channels.	M_{size} = 640×480	307,200
Matrix Mul.	Dense integer matrix multiplication.	M_{size} = 100×100	40,000
String Match	Search file with keys for 4 encrypted words.	N_{keys} = 307,200	307,200
Word Count	Counts occurrence frequency of words in file.	N_{words} = 50,000	90,094
Linear Regr.	Compute the best fit line for a set of points.	N_{points} = 100,000	400,000
PCA	Principal components analysis on a matrix.	M_{size} = 250×250	250,000
K_{means}	Clustering 3-D data points into 10 groups	N_{points} = 20,000	240,000

eliminating such coding forms.

- **Data structures**: The Phoenix framework uses a lot of complex data structures, i.e. structs with array and pointer elements. While scalar pointers that point to statically reserved data are normally deployed in Vivado HLS, the same does not happen with double and beyond pointers, i.e. pointer-to-pointer. We had to refactor such complex data types to simple scalar or simple pointer based structures.
- **ANCI C synthesizable subset**: The Phoenix framework uses many functions of ANCI C that are not synthesizable by Vivado HLS., e.g. limitation of memory copy operations such as memmove, memcpy, etc., string functions, e.g. strcmp, strlen, strcpy, toupper, etc. and math functions, e.g. rand, sort, etc.. For all of these functions we developed synthesizable versions, working on byte/cycle rate. Depending on application characteristics, we customized these functions to be more efficient using pipelining and loop unrolling techniques.

5.2.3. Experimental Results

This section describes the experimental setup we used to evaluate HLSMapReduceFlow, as well as the respective obtained results. We evaluated the efficiency of the proposed HLSMapReduceFlow framework considering a MapRecude accelerator for a FPGA-based architecture, targeting to emerging application domains, e.g. artificial intelligence, scientific computing, enterprise computing etc. In this research work, we considered six applications evaluation test-bed of Phoenix MapReduce framework for shared-memory systems [24]. The performance evaluation covers a representative set of application that typically use the MapReduce framework. The characterization setup of the employed applications is summarized in Table 5.3.

To evaluate our framework in performance and scalability, we build up a testbed for the HLSMapReduceFlow. Since the main scope of this work is the acceleration of Map tasks (95% of total execution time in Phoenix), we explore different architecture exploiting Map accelerators in the FPGA, while for the following measurements we have used only one Reduce task. Also we do not measure communication overhead for transferring input data streams to the FPGA. Instead we use the on-board FPGA memory to store input streams. This scheme may not be a complete architecture for datacenters where new requests are coming constantly. However in this work we study the performance micro-architecture exploitation by instantiating

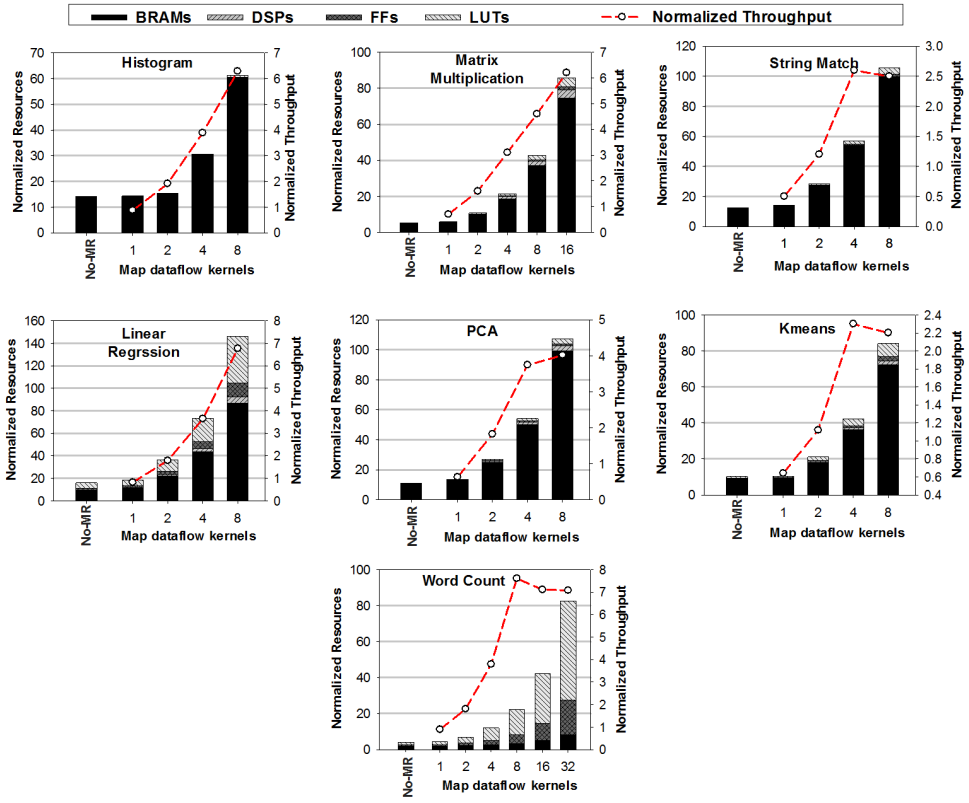


Figure 5.20 Self performance-scalability tradeoff of HLSMapReduceFlow framework.

dataflow-based Map accelerators in the FPGA, regardless the input source and the way input data are reaching the Map tasks.

Figure 5.20 shows the self overall performance-scalability tradeoff results for every employed application, when we use, or not, the HLSMapReduceFlow framework. Every horizontal axis scales the Map accelerators from single-instance to the maximal number of accelerators. This number is limited by the reserved FPGA resources of applications. As shown by the comparison of the architecture without the MapRecude framework (No-MR), and the 1-Map accelerator instance, the implementation of HLSMapReduceFlow framework introduces a cost in both resources and execution time by a factor of 18% and 38% respectively. However, as long as more Map accelerators are instantiated, the performance in terms of throughput is almost linear boosted.

Table 5.4 Real-word representative comparison between HLSMapReduceFlow-accelerated FPGA and commodity workstation.

Framework	**GNU/Linux 3.18.6 x86-64 / GCC-glibc**			**HLSMapReduceFlow**						
Platform	**AMD 8-core FX-8350 4GHz**			**Virtex7-XC7VX485T 150MHz**				**Ratio**		
Metrics	**Time(ms)**	**Power(W)**	**Energy(J)**	T_p**(ms)**	T_c**(ms)**	**Power(W)**	**Energy(J)**	**T**	**P**	**E**
Histogram	344	41.1	14.1	72.2	4.8	1.84	0.13	0.21	0.04	0.009
Matrix Mul/tion	177	41.3	7.3	208	0.6	1.02	0.21	1.17	0.03	0.029
String Match	206	41.6	8.5	95	4.9	2.33	0.22	0.46	0.06	0.026
Word Count	172	40.8	7.0	84	1.4	1.87	0.16	0.48	0.05	0.023
Linear Reg/sion.	158	41.6	6.6	73	6.4	2.08	0.15	0.46	0.05	0.023
PCA	392	41.9	16.4	964	4.1	1.17	1.13	2.45	0.03	0.070
K_{means}	435	40.3	17.5	503	3.8	1.03	0.52	1.16	0.03	0.029
Average	269	41.2	11	285	3.7	1.62	0.36	**1.06**×	**0.04**×	**0.03**×

However some applications reach a saturation point where the instantiation of more accelerators does not lead to expected speedup. This is the case for String Match, PCA, Kmeans and Word Count. We found that both the dynamic behavior of these applications and data dependency among calculations prevents Vivado HLS for applying effective dataflow processing optimizations. For instance, the PCA kernel is a streaming application with no dynamism. However its computations have high data dependencies without equivalent data locality. Thus, the fine-grain splitting of input data to data chunks that include elements needed by more than one map accelerators, causes performance drop due to stalled Map processing tasks.

In order to provide a more real-word representative comparison, we evaluated HLSMapReduceFlow against a high-end workstation. The workstation is powered by the 8-core AMD FX-8350 processor clocked at 4GHz. This processor has a TDP value of 125 Watts. We compiled the employed applications using GCC compiler (v4.9.2) and run the applications with glibc runtime linking, in a GNU/Linux (Kernel 3.18.6) 64-bit OS, enabling many compiler omptimizations (-O2), including vector processing ones (SSE, AVX etc.). The derived measurements for execution time, power and energy are shown in the first three columns of Table 5.4. The next four columns show the respective metrics for a system composed of a Virtex-7 FPGA (XC7VX485T) clocked at 150MHz utilizing the HLSMapReduceFlow framework. The PC-FPGA communication is established with a PCI Express 3.0 link, offering maximum bandwidth of 8Gbps. The overall measured time for the FPGA deployment is represented by the time for processing on the FPGA, T_p and the time for PC-FPGA communication, T_c (downloading input data from PC to FPGA and uploading results from FPGA to PC).

In the current design, data are stored in the block RAMs (BRAMs) embedded in the FPGAs. In order to hide the communication overhead we could pipeline the I/O transfer with the computation tasks. For example we could use additional BRAMs to store the next stream of data that is going to be processed by the FPGA while computing the current stream of data. The power values have been derived through the usage of PowerTop[6] utility for the CPU and Xilinx Xpower[7] utility for the FPGA. As shown, the proposed framework delivers extremely performance-per-watt efficient solutions, reporting two orders of magnitude less energy for the same execution timing window. Consequently we show that the proposed scheme is an elegant candidate implementation infrastructure for data centers that promises high energy efficiency for specific types of applications.

5.2.4. Conclusions

This research work introduces HLSMapReduceFlow, a MapReduce framework on FPGA devices, which provides implementation abstraction, hardware architecture and an exemplary system-level framework for system designers. The HLSMapReduceFlow framework can be widely used as a programming framework both for FPGA-coupled data centers and for cloud-computing applications. The proposed hardware accelerator can be used to reduce the total execution time for a workstation coupled with an FPGA device such as the Xilinx Virtex-7 and the cloud-computing applications based on the MapReduce framework by accelerating the Map/Reduce tasks of these applications. Due to micro-architecture exploitation offered by the employed state-of-art Vivado HLS tool, the framework succeed in high-parallelism by applying fundamental concepts from spatial and dataflow computing. We evaluated HLSMapReduceFlow with well-established server workloads against a high-end workstation and we obtained throughput gains of 4.3× while the FPGA device saved up to two orders of magnitude energy consumption. As a future step we plan to integrate to HLSMapReduceFlow our novel synthesizable dynamic memory management allocator [35], presented in section 5.1 of this Chapter, exploiting hardware memory paging capabilities and thus offering support for complex dynamic applications.

[6] https://01.org/powertop
[7] http://www.xilinx.com/products/design_tools/logic_design/verification/xpower.htm

References

[1] G. Moore, *Cramming more components onto integrated circuits,* Proceedings of the IEEE **86**, 82 (1998).

[2] CNET.com, *End of moore's law: It's not just about physics,* .

[3] Recode.net, *Moore's law hits 50, but it may not see 60,* .

[4] R. Dennard, F. Gaensslen, H.-N. YU, V. Rideout, E. BASSOUS, and A. R. LEBLANC, *Design of ion-implanted mosfet's with very small physical dimensions,* Proceedings of the IEEE **87**, 668 (1999).

[5] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, *Power challenges may end the multicore era,* Commun. ACM **56**, 93 (2013).

[6] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, *The eda challenges in the dark silicon era: Temperature, reliability, and variability perspectives,* in *Proceedings of the 51st Annual Design Automation Conference*, DAC '14 (ACM, New York, NY, USA, 2014) pp. 185:1–185:6.

[7] C. Mack, *Fifty years of moore's law,* Semiconductor Manufacturing, IEEE Transactions on **24**, 202 (2011).

[8] M. Taylor, *Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse,* in *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE* (2012) pp. 1131–1136.

[9] J. Shalf, D. Quinlan, and C. Janssen, *Rethinking hardware-software codesign for exascale systems,* Computer **44**, 22 (2011).

[10] Q. Zhu, B. Akin, H. Sumbul, F. Sadi, J. Hoe, L. Pileggi, and F. Franchetti, *A 3d-stacked logic-in-memory accelerator for application-specific data intensive computing,* in *3D Systems Integration Conference (3DIC), 2013 IEEE International* (2013) pp. 1–7.

[11] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, *Conservation cores: Reducing the energy of mature computations,* SIGARCH Comput. Archit. News **38**, 205 (2010).

[12] Y.-T. Chen, J. Cong, M. Ghodrat, M. Huang, C. Liu, B. Xiao, and Y. Zou, *Accelerator-rich cmps: From concept to real hardware,* in *Computer Design (ICCD), 2013 IEEE 31st International Conference on* (2013) pp. 169–176.

[13] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, *Architecture support for domain-specific accelerator-rich cmps,* ACM Trans. Embed. Comput. Syst. **13**, 131:1 (2014).

[14] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith,

J. Thong, P. Y. Xiao, and D. Burger, *A reconfigurable fabric for accelerating large-scale datacenter services,* in *41st Annual International Symposium on Computer Architecture (ISCA)* (2014).

[15] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, *High-level synthesis for fpgas: From prototyping to deployment,* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on **30**, 473 (2011).

[16] M. Araya-Polo, J. Cabezas, M. Hanzich, M. Pericas, F. Rubio, I. Gelado, M. Shafiq, E. Morancho, N. Navarro, E. Ayguade, J. Cela, and M. Valero, *Assessing accelerator-based hpc reverse time migration,* Parallel and Distributed Systems, IEEE Transactions on **22**, 147 (2011).

[17] M. J. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks, *The accelerator store: A shared memory framework for accelerator-based systems,* ACM Trans. Archit. Code Optim. **8**, 48:1 (2012).

[18] E. Cota, P. Mantovani, M. Petracca, M. Casu, and L. Carloni, *Accelerator memory reuse in the dark silicon era,* IEEE Computer Architecture Letters **99**, 1 (2012).

[19] L. Semeria and G. De Micheli, *Spc: synthesis of pointers in c application of pointer analysis to the behavioral synthesis from c,* in *Computer-Aided Design, 1998. ICCAD 98. Digest of Technical Papers. 1998 IEEE/ACM International Conference on* (1998) pp. 340–346.

[20] L. Séméria, K. Sato, and G. De Micheli, *Resolution of dynamic memory allocation and pointers for the behavioral synthesis form c,* in *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '00 (ACM, New York, NY, USA, 2000) pp. 312–319.

[21] M. Shalan and V. J. Mooney, *A dynamic memory management unit for embedded real-time system-on-a-chip,* in *Proceedings of the 2000 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, CASES '00 (ACM, New York, NY, USA, 2000) pp. 180–186.

[22] S. Xydis, A. Bartzas, I. Anagnostopoulos, D. Soudris, and K. Z. Pekmestzi, *Custom multi-threaded dynamic memory management for multiprocessor system-on-chip platforms,* in *ICSAMOS* (2010) pp. 102–109.

[23] Y. Sade, M. Sagiv, and R. Shaham, *Optimizing c multithreaded memory management using thread-local storage,* in *Proceedings of the 14th International Conference on Compiler Construction*, CC'05 (2005) pp. 137–155.

[24] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, *Evaluating mapreduce for multi-core and multiprocessor systems,* in *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on* (2007) pp. 13–24.

[25] J. Dean and S. Ghemawat, *Mapreduce: Simplified data processing on large clusters,* Commun. ACM **51** (2008), 10.1145/1327452.1327492.

[26] Y. Shan, B. Wang, J. Yan, Y. Wang, N. Xu, and H. Yang, *Fpmr: Mapreduce framework on fpga,* in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '10 (ACM, New York, NY, USA, 2010).

[27] D. Yin, G. Li, and K.-d. Huang, *Scalable mapreduce framework on fpga accelerated commodity hardware,* in *Internet of Things, Smart Spaces, and Next Generation Networking*, Lecture Notes in Computer Science, Vol. 7469, edited by S. Andreev, S. Balandin, and Y. Koucheryavy (Springer Berlin Heidelberg, 2012).

[28] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, *Mars: A mapreduce framework on graphics processors,* in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08 (ACM, New York, NY, USA, 2008) pp. 260–269.

[29] J. H. C. Yeung, C. C. Tsang, K. H. Tsoi, B. S. H. Kwan, C. C. C. Cheung, A. P. C. Chan, and P. H. W. Leong, *Map-reduce as a programming model for custom computing machines,* in *Proceedings of the 2008 16th International Symposium on Field-Programmable Custom Computing Machines*, FCCM '08 (IEEE Computer Society, Washington, DC, USA, 2008) pp. 149–159.

[30] K. Gyftakis, I. Anagnostopoulos, D. Soudris, and D. Reisis, *A mapreduce framework implementation for network-on-chip platforms,* in *Electronics, Circuits and Systems (ICECS), 2014 21st IEEE International Conference on* (2014) pp. 120–123.

[31] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, *Parallel data processing with mapreduce: A survey,* SIGMOD Rec. **40**, 11 (2012).

[32] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Xiao, and D. Burger, *A reconfigurable fabric for accelerating large-scale datacenter services,* in *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on* (2014) pp. 13–24.

[33] C. Kachris, G. Sirakoulis, and D. Soudris, *A reconfigurable mapreduce accelerator for multi-core all-programmable socs,* in *System-on-Chip (SoC), 2014 International Symposium on* (2014) pp. 1–6.

[34] A. H. Veen, *Dataflow machine architecture,* ACM Comput. Surv. **18**, 365 (1986).

[35] D. Diamantopoulos, S. Xydis, K. Siozios, and D. Soudris, *Dynamic memory management in vivado-hls for scalable many-accelerator architectures,*

in *Proceedings of the 20015 11th International Symposium on Applied Reconfigurable Computing*, ARC 2015 (2015 - accepted for presentation in April, 2015).

6

Conclusions and Future Directions

This chapter presents the main conclusions of this doctoral thesis. Specifically, the following section highlights the main innovation directions introduced by the proposed methodological framework and drawing tools. Finally, the last section reports the future research insights, based on the results of this thesis.

6.1. Thesis Overview

The doctoral thesis "Cross-Layer Rapid Prototyping and Synthesis of Application-Specific and Reconfigurable Many-accelerator Platforms" introduced new architectural templates and innovative design approaches in order to provide solutions to modern design problems of integrated systems-on-chip devices. The target of this thesis is twofold, technological and commercial. The thesis aims to contribute to competitive solutions, which both promise to obey to market trends and to be sustainable, regarding the scaling in technology, semiconductor material, design tools and development time. This approach is accomplished by developing a holistic methodological framework which categorizes design levels and suggests distinct and non-overlapping optimization procedures. Alongside, the thesis proposes architectural templates that are consistent with commercial directions, offering elevated systemic complexity, low design and verification time and a set of meta-characteristics, such as reliability, reprogramming and adaptability, which marks a milestone in transition to the new generation of integrated semiconductor systems. Basic direction of this generation is to bridge the gap of design complexity and technological productivity, which is achieved by diagonal design coursing, between the two perpendicular axes of development, the technological and the commercial one. The contributions per chapter are enumerated as follows:

1. We developed a new methodological framework for rapid virtual prototyping of embedded systems and systems that combine general purpose processors

with hardware accelerators (Chapter 2). This framework proposes the adoption of virtual models of hardware components during the design phase to ensure a functional level of the platform for software development. This new co-design approach makes it possible to start software development, testing and debugging much earlier than is was possible in the past with conventional sequential co-design methodologies. The proposed methodology is verified by evaluation scenarios ranging from the design of specialized processing cores, to the HW/SW co-design of heterogeneous embedded systems [1, 2].

2. We introduced a framework for effective hardware/software co-design and partitioning flow for heterogeneous systems (Section 3.3). The template of such systems includes general purpose processors and reconfigurable logic. The partitioning task is based on an application characterization process. The innovation introduced in this context deals with the multi-objective and systemic analysis of the applications, in order to select the right decision for placement of each processing task to the optimum processing resource, e.g. Software-CPU/Hardware-FPGA. The proposed systematic methodology examines the computational complexity, the memory requirements (storage size and variables data life) as well as the communication costs, to find the optimal coarse-grained accelerators architecture as a means of accelerating. This approach was used in the design of a heterogeneous system, which supports autonomous navigation in space robotic vehicles through the use of computer vision algorithms. The implemented system manages to achieve the standards set by the ESA organization through the design of specialized hardware accelerators, according to the co-design and partitioning suggestions of the proposed framework [3–5].

3. We proposed a methodology and the corresponding design tools for evaluating the three-dimensional semiconductor integration technology (3-D), using conventional commercial CAD design tools (Section 3.4). The proposed approach introduces new design tools that manage to insert the constraints posed by the third dimension in the conventional 2-D tools. Most of the developed tools uses open formats in order to provide a competitive advantage over the limited industrial propriety 3-D design tools. The framework was used in designing 3-D SoC systems for signal processing, based on the embedded processors *Leon 3* and *OpenRISC*, on technology nodes of TSMC 45nm and 130nm. The adoption of 3-D design approach reported energy savings of 20% compared to the conventional two-dimensional implementation.and optimization of the maximum operating frequency up to 1,26× [1, 6–8].

4. We developed an effective and rapid solution to the placement problem for reconfigurable architectures (Chapter 4). The proposed placement algorithm uses genetic algorithms to efficiently explore the solutions space. Alongside the proposed placement tool, by exploiting the inherent parallelism of genetic algorithm, achieves optimal scaling on modern multicore processors, using a coarse-grain programming parallelism. To improve the quality of placement solutions, the proposed methodological framework uses an investigative stage

of the parameters of the genetic algorithm, adapted to the respective application. The experimental results verified the superiority of the proposed tool compared to existing tools, reporting solutions which either being explored faster by a factor of up to 67×, or provide qualitative regarding the operating frequency of 1.16×. The introduced algorithm also maintains steady scaling regarding to the memory footprint and computational resource requirements.

5. We introduced a systematic framework which proposes automated exploration of the micro-architecture of a system-on-chip, in order to minimize the thermal profile and therefore increase the reliability of the integrated circuit (Section 3.1). The framework recognizes the problem of power density, i.e. the increased temperature at the places of an IC, where there is a lot of resources activity and small area to dissipate the cumulative energy. This problem leads to continuous strain of a particular silicon surface, due to high temperature values. The framework propose the selective replication of hardware modules that face the aforementioned problem, and through the design of a dedicated real-time controller, the implemented system is capable of maintaining a constant thermal profile per application. At run-time the controller schedules the processes at the replica copies, based on their temperature values. The experimental results from the application of the proposed approach in the design of a software defined radio system (SDR), showed that the framework can achieve an average reduction of 17 degrees Kelvin. Such reduction implies an increase of aging phenomena of about 14% with a silicon overhead of only 15% [9, 10].

6. We developed a new architectural template for coarse-grained many-accelerator systems in reconfigurable platforms (Section 5.1). Through measurements it was established that the main problem for the development of such accelerators is both their efficient programming and limited the available on-chip memory of FPGA devices. Specifically, through this work, it is stressed that although the research community indicate the adequate energy as the biggest problem in designing mass accelerator systems (Dark Silicon) [11, 12]. However, the problem of limited on-chip memory, even in modern FPGAs, can lead to systems with a 2.5-fold fewer accelerators, relative to the maximum tolerable threshold posed by thermal and energy profiles. The major cause of this problem is the static memory allocation of FPGAs drawing tools. To this end, the proposed framework proposes the adoption of high-level synthesis tools for the abstract programming and flexible time-management of multiple processes for many-accelerator systems. Also we developed a dynamic memory library to overcome the obstacle of the static binding memory. The proposed framework was applied in modern applications of high-performance computing (HPC) on the mature industrial HLS tool for FPGAs, the Xilinx Vivado HLS and managed to increase the accelerators density at 3.8× and thus the system throughput by 3.1× and 21.4× for shared and private memory among the accelerators, respectively [13, 14].

7. We introduced an architectural model and the corresponding programming

environment for scalable design solutions in data centers and workstations (Section 5.2). The framework is based on MapReduce framework and targets the design of processing nodes in distributed systems that process large amounts of data. In particular, we developed a design flow for many-accelerator systems that utilize the MapReduce framework on FPGA devices. The approach is evaluated with a high-end FPGA device and it is compared with a typical workstation. The test-case proved the effectiveness of the proposed solution, by reporting twofold less energy consumption for the same processing load and timing window [15].

6.2. Future Directions

In this dissertation, viable scaling architecture solutions were presented. The proposed design flows and tools can sustain efficiently the requirements posed by the contemporary age, denoted as "Internet-of-Things", "Cyber-Physical Systems", "Systems-on-Systems" era etc., i.e. high performance per watt, reconfigurability, application adaptability, heterogeneity and systemic synthesis. Meeting such diverse and contradictory specifications has been the major motivation and, at the same time, the major hindrance in this dissertation. Given the continuous advances in market needs and the time frame at our disposal, a number of open issues in this dissertation work remain. We highlight the major research insights, steamed from this dissertation, in accordance to research activities of Microprocessors and Digital Systems Laboratory, NTUA, as follows:

- Extending virtual prototyping at operating system level, enabling early verification of software applications that depend on the operating system libraries. Such an extension could enable the debugging and verification in specific late operating system scenarios, i.e. booting-up the system, running application for some seconds/minutes and freezing-up the OS in order to explicitly test the under-development hardware module against specific OS-related events, e.g. interrupts from external devices.

- Combined optimization of temperature, energy consumption and reliability in the genetic algorithm for the FPGA placement problem, in order to investigate the search space at every execution step.

- Supporting partial-reconfiguration in the proposed placement tool to support self-adaptability of FPGA devices to dynamic operating conditions. The proposed placement tool, i.e. GENESIS, has built-in codebase to support the partial-reconfiguration feature by keeping fragmented areas for the target FPGA. The tool can be extended with a run-time partial-reconfiguration controller that adapts pre-configure bitstreams according to dynamic environmental changes, i.e. temperature, power supply, radiation profile etc.

- Supporting multiple supply voltage islands in three-dimensional integration technology architectures, to support low energy and thermal efficiency, so as to balance low heat dissipation problem of 3-D circuits.

- Combined exploration of parameters of dynamic memory management at hardware level ,to support optimal adaptation of memory subsystem in data requirements of dynamic applications on many-accelerators systems.
- Developing an automated exploration methodology of high level synthesis parameters for many-accelerator architectures, in order to associate the solution space with the HLS optimization parameters. A prior case study has shown that a carefull selection of HLS parameters can eliminate the search space by 34× [6, 16]. However the framework lacks a heuristic approach for guiding the pruning of search space. An early idea on this direction involves the training of neural networks with HLS micro-architecture optimization patters.
- Integrating dynamic memory manager to the MapReduce environment for workstation applications and data centers, in order to support flexible data transfer techniques with custom memory footprint on the real needs of the dynamic application and therefore increasing performance per watt. The key idea of this insight is the transparent adaptation of on-chip FPGA memory, i.e. BRAMs, to the MapReduce API calls, through High-Level-Synthesis design flow, i.e. apply the proposed DMM-HLS framework to support the complex data-types (e.g. linked lists) of MapReduce codebase.

References

[1] D. Diamantopoulos, E. Sotiriou-Xanthopoulos, K. Siozios, G. Economakos, and D. Soudris, *Plug&chip: A framework for supporting rapid prototyping of 3d hybrid virtual socs,* ACM Trans. Embed. Comput. Syst. **13**, 168:1 (2014).

[2] D. Diamantopoulos, K. Siozios, E. Sotiriou-Xanthopoulos, G. Economakos, and D. Soudris, *Hvsocs: A framework for rapid prototyping of 3-d hybrid virtual system-on-chips,* in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International* (2013) pp. 2194–2199.

[3] I. Kostavelis, L. Nalpantidis, E. Boukas, M. A. Rodrigalvarez, I. Stamoulias, G. Lentaris, D. Diamantopoulos, K. Siozios, D. Soudris, and A. Gasteratos, *Spartan: Developing a vision system for future autonomous space exploration robots,* Journal of Field Robotics **31**, 107 (2014).

[4] D. Diamantopoulos, K. Siozios, G. Lentaris, D. Soudris, and M. Rodrigalvarez, *Spartan project: On profiling computer vision algorithms for rover navigation,* in *Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on* (2012) pp. 174–181.

[5] G. Lentaris, D. Diamantopoulos, K. Siozios, D. Soudris, and M. Rodrigalvarez, *Hardware implementation of stereo correspondence algorithm for the exomars mission,* in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on* (2012) pp. 667–670.

[6] D. Diamantopoulos, I. Galanis, K. Siozios, G. Economakos, and D. Soudris, *A framework for rapid system-level synthesis targeting to reconfigurable platforms,* in *Workshop on Reconfigurable Computing (WRC), 2015, Netherlands*.

[7] D. Diamantopoulos, K. Siozios, and D. Soudris, *Framework for performing rapid evaluation of 3d socs,* Electronics Letters **48**, 679 (2012).

[8] D. Diamantopoulos, K. Siozios, D. Bekiaris, and D. Soudris, *A novel methodology for architecture-level exploration of 3d socs,* in *Design Technology of Integrated Systems in Nanoscale Era (DTIS), 2011 6th International Conference on* (2011) pp. 1–6.

[9] D. Diamantopoulos, K. Siozios, S. Xydis, and D. Soudris, *A systematic methodology for reliability improvements on soc-based software defined radio systems,* VLSI Des. **2012**, 13:13 (2012).

[10] D. Diamantopoulos, K. Siozios, S. Xydis, and D. Soudris, *Thermal optimization for micro-architectures through selective block replication,* in *Embedded Computer Systems (SAMOS), 2011 International Conference on* (2011) pp. 59–66.

[11] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, *Dark silicon and the end of multicore scaling,* in *Proceedings of the 38th Annual*

International Symposium on Computer Architecture, ISCA '11 (ACM, New York, NY, USA, 2011) pp. 365–376.

[12] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, *Toward dark silicon in servers,* Micro, IEEE **31**, 6 (2011).

[13] D. Diamantopoulos, S. Xydis, K. Siozios, and D. Soudris, *Mitigating memory-induced dark silicon in many-accelerator architectures,* Computer Architecture Letters **PP**, 1 (2015).

[14] D. Diamantopoulos, S. Xydis, K. Siozios, and D. Soudris, *Dynamic memory management in vivado-hls for scalable many-accelerator architectures,* in *Applied Reconfigurable Computing*, Lecture Notes in Computer Science, Vol. 9040, edited by K. Sano, D. Soudris, M. Hübner, and P. C. Diniz (Springer International Publishing, 2015) pp. 117–128.

[15] D. Diamantopoulos and C. Kachris, *High-level synthesizable dataflow mapreduce accelerator for fpga-coupled data centers,* in *Embedded Computer Systems (SAMOS), 2015 International Conference on* (2015).

[16] I. Koutras, K. Maragos, D. Diamantopoulos, K. Siozios, and D. Soudris, *A framework for rapid system-level synthesis targeting to reconfigurable platforms,* Integration, the VLSI Journal, under review process .

List of Publications

Book Chapters

1. D. Diamantopoulos, K. Siozios, G. Economakos, and D. Soudris, "Chapter 9 On Designing 3-D Platforms", in "Designing 2D and 3D Network-on-Chip Architectures", pp.209-236, doi:10.1007/978-1-4614-4274-5_9, Springer, 2014.

Journals

7. D. Diamantopoulos, S. Xydis, K. Siozios, D. Soudris, "Mitigating Memory-induced Dark Silicon in Many-Accelerator Architectures," IEEE Computer Architecture Letters , vol.PP, no.99, pp.1,1 doi: 10.1109/LCA.2015.2410791, March 2015.

6. D. Diamantopoulos, K. Siozios, S. Xydis, D. Soudris. "GENESIS: Parallel Application Placement onto Reconfigurable Architectures (Invited for the Special Issue on Runtime Management)." ACM Transactions on Embedded Computing Systems (TECS) vol. 14, no. 1: 18, doi: 10.1145/2629651, January 2015.

5. I. Kostavelis, L. Nalpantidis, E. Boukas, M. Aviles Rodrigalvarez, I. Stamoulias, G. Lentaris, D. Diamantopoulos, K. Siozios, D. Soudris, A. Gasteratos. "SPARTAN: Developing a vision system for future autonomous space exploration robots." Journal of Field Robotics vol. 31, no. 1, pp.107-140. doi:10.1002/rob.21484, 2014.

4. E. Sotiriou-Xanthopoulos, D. Diamantopoulos, K. Siozios, G. Economakos, D. Soudris. "A framework for rapid evaluation of heterogeneous 3-D NoC architectures." Elsevier Microprocessors and Microsystems vol. 38, no. 4, pp. 292-303, doi:10.1016/j.micpro.2013.09.003, June 2014.

3. D. Diamantopoulos, E. Sotiriou-Xanthopoulos, K. Siozios, G. Economakos, D. Soudris. "Plug&Chip: A Framework for Supporting Rapid Prototyping of 3D Hybrid Virtual SoCs". ACM Transactions on Embedded Computing Systems (TECS), vol. 13, no. 5s, Article 168, pp. 1-25, 25 pages, doi:10.1145/2661634, December 2014.

2. D. Diamantopoulos, K. Siozios, D. Soudris, "Framework for performing rapid evaluation of 3D SoCs," IET Electronics Letters, vol.48, no.12, pp.679,681, doi: 10.1049/el.2012.1321, June 2012.

1. D. Diamantopoulos, K. Siozios, S. Xydis, and D. Soudris. "A systematic methodology for reliability improvements on SoC-based software defined radio systems", Hindawi VLSI Design, Article ID 784945, doi:10.1155/2012/784945, 15 pages, January 2012.

International Conferences

16. D. Diamantopoulos, C. Kachris. "High-level Synthesizable Dataflow MapReduce Accelerator for FPGA-coupled Data Centers", Embedded Computer Systems (SAMOS), 2015 International Conference on, Accepted for publication. To be presented in July 2015.

15. D. Diamantopoulos, S. Xydis, K. Siozios, D. Soudris. "Dynamic memory management in Vivado-hls for scalable many-accelerator architectures." In Applied Reconfigurable Computing (ARC), pp. 117-128. Springer International Publishing, March 2015.

14. E. Sotiriou-Xanthopoulos, D. Diamantopoulos, G. Economakos. "Evaluation of High-Level Synthesis Techniques for Memory and Datapath Tradeoffs in FPGA Based SoC Architectures." In Applied Reconfigurable Computing (ARC), pp. 321-330. Springer International Publishing, March 2015.

13. K. Siozios, P. Figuli, H. Sidiropoulos, C. Tradowsky, D. Diamantopoulos, K. Maragos, S. Percy Delicia, D. Soudris, J. Becker. "TEAChER: TEach AdvanCEd Reconfigurable Architectures and Tools." In Applied Reconfigu-rable Computing (ARC), pp. 103-114. Springer International Publishing, March 2015.

12. G. Lentaris, I. Stamoulias, D. Diamantopoulos, K. Maragos, K. Siozios, D. Soudris, M. Aviles Rodrigalvarez, M. Lourakis, X. Zabulis, I. Kostavelis, L. Nalpantidis, E. Boukas, A. Gasteratos, "SPARTAN/SEXTANT/COMPASS: Advancing Space Rover Vision via Reconfigurable Platforms." In Applied Reconfigurable Computing (ARC), pp. 475-486. Springer International Publi-shing, March 2015.

11. D. Diamantopoulos, G. Economakos, D. Reisis, "Using high-level synthesis to build memory and datapath optimized DSP accelerators," Electronics, Circuits and Systems (ICECS), 2014 21st IEEE International Conference on, pp.714,717, doi: 10.1109/ICECS.2014.7050085, December 2014.

10. D. Diamantopoulos, C. Economakos, D. Soudris, G. Economakos, "A new design paradigm for floating point DSP applications based on ESL/HLS and FPGAs," Signal Processing and Information Technology (ISSPIT), 2013 IEEE International Symposium on, pp.000404,000409, 12-15,
doi: 10.1109/ISSPIT.2013.6781915, December 2013.

9. D. Diamantopoulos, K. Siozios, E. Sotiriou-Xanthopoulos, G. Economakos, D. Soudris, "HVSoCs: A Framework for Rapid Prototyping of 3-D Hybrid Virtual

System-on-Chips," Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International, pp.2194,2199, 20-24, doi: 10.1109/IPDPSW.2013.202, May 2013.

8. G. Lentaris, D. Diamantopoulos, G. Stamoulias, K. Siozios, D. Soudris, M.A. Rodrigalvarez, "FPGA-based path-planning of high mobility rover for future planetary missions," Electronics, Circuits and Systems (ICECS), 2012 19th IEEE International Conference on, pp.85,88, doi: 10.1109/ICECS.2012.6463793, December 2012.

7. D. Diamantopoulos, K. Siozios, G. Lentaris, D. Soudris, M.A. Rodrigalvarez, "SPARTAN project: On profiling computer vision algorithms for rover navigation," Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on, pp.174,181, doi: 10.1109/AHS.2012.6268647, June 2012.

6. G. Lentaris, D. Diamantopoulos, K. Siozios, D. Soudris, M.A. Rodrigalvarez, "Hardware implementation of stereo correspondence algorithm for the ExoMars mission," Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on , pp.667,670, doi: 10.1109/FPL.2012.6339173, August 2012.

5. D. Diamantopoulos, P. Galiatsatos, A. Karachalios, G. Lentaris, D. Reisis, D. Soudris, "Configurable baseband digital transceiver for Gbps wireless 60 GHz communications," Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on, pp.192,195, doi: 10.1109/ICECS.2011.6122246, December 2011.

4. E. Sotiriou-Xanthopoulos, D. Diamantopoulos, G. Economakos, D. Soudris, "Design and experimentation with low-power morphable multipliers," Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on , vol., no., pp.752,755, doi: 10.1109/ICECS.2011.6122383, December 2011.

3. K. Siozios, D. Diamantopoulos, I. Kostavelis, E. Boukas, L. Nalpantidis, D. Soudris, A. Gasteratos, M. Aviles, I. Anagnostopoulos, "SPARTAN project: Efficient implementation of computer vision algorithms onto reconfigurable platform targeting to space applications," in Proceedings of the 6th Internatio-nal Workshop on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), doi: 10.1109/ReCoSoC.2011.5981524, pp.1,9, June 2011.

2. D. Diamantopoulos, K. Siozios, S. Xydis, D. Soudris, "Thermal optimization for micro-architectures through selective block replication," Embedded Computer Systems (SAMOS), 2011 International Conference on, pp.59,66, doi: 10.1109/SAMOS.2011.6045445, July 2011.

1. D. Diamantopoulos, K. Siozios, D. Bekiaris, D. Soudris, "A novel methodology for architecture-level exploration of 3D SoCs," Design & Technology of Inte-

grated Systems in Nanoscale Era (DTIS), 2011 6th International Conference on, pp.1,6, doi: 10.1109/DTIS.2011.5941425, April 2011

International Workshops

13. D. Diamantopoulos, I. Galanis, K. Siozios, G. Economakos, and D. Soudris, "A Framework for Rapid System-Level Synthesis Targeting to Reconfigurable Platforms", Workshop on Reconfigurable Computing (WRC), 2015.

12. D. Diamantopoulos, K. Siozios, E. Sotiriou-Xanthopoulos, G. Economakos and D. Soudris, "HVSoCs: A Framework for Rapid Prototyping of 3-D Hybrid Virtual System-on-Chips", Workshop on Virtual Prototyping of Parallel and Embedded Systems (VIPES), May 20-24th, 2013.

11. D. Diamantopoulos, P. Galiatsatos, A. Karachalios, G. Lentaris, D. Reisis and D. Soudris, A Reconfigurable Baseband Architecture for Gbps Wireless 60 GHz Communications, Fifth Friday Workshop on Designing for Embedded Parallel Computing Platforms (DEPCP), March 22, 2013, Grenoble, France.

10. G. Lentaris, D. Diamantopoulos, K. Siozios, I. Stamoulias, I. Kostavelis, E. Boukas, L. Nalpantidis, D. Soudris, A. Gasteratos, and M. Aviles, "SPARTAN: Efficient Implementation of Computer Vision Algorithms for Autonomous Rover Navigation", Workshop on Reconfigurable Computing (WRC), Jan. 2013, Germany.

9. G. Lentaris, I. Stamoulias, D. Diamantopoulos, K. Siozios, and D. Soudris, "An FPGA implementation of the SURF algorithm for the ExoMars program-me", Workshop on Reconfigurable Computing (WRC), Jan. 2013, Germany.

8. D. Diamantopoulos, K. Siozios, and D. Soudris, "A Framework for Performing Fault-Tolerant Placement Based on Genetic Algorithm", Workshop on Reconfigurable Computing (WRC), Jan. 2013, Germany.

7. D. Diamantopoulos, K. Siozios, S. Xydis and D. Soudris, A genetic algorithm-based FPGA placer for multi-core processors, Fifth Friday Workshop on Designing for Embedded Parallel Computing Platforms (DEPCP), March 22, 2013, Grenoble, France

6. D. Diamantopoulos, K. Siozios, I. Stamoulias, G. Lentaris, D. Soudris and M. Aviles, Towards Computer Vision FPGA Acceleration, DATE Friday Workshop on Reconfigurable Computing (Configcomp), March 22, 2013, Grenoble, France.

5. D. Diamantopoulos, K. Siozios, S. Xydis and D. Soudris, "A Framework for Supporting Parallel Application Placement onto Reconfigurable Platforms", Workshop on Parallel Programming and Run-time Management Techniques for Many-core Architectures (PARMA), Jan. 2013.

4. K. Siozios, H. Sidiropoulos, D. Diamantopoulos, P. Figuli, D. Soudris, M. Hubner and J. Becker, "On Designing Self-Aware Reconfigurable Platforms", Workshop on Self-Awareness in Reconfigurable Computing Systems (SRCS), pp. 14-17, Norway, 2012.

3. D. Diamantopoulos, G. Lentaris, K. Siozios, D. Soudris and M. Aviles, "Towards Accelarating Computer Vision Algorithms Targeting to Space Applications with a Heterogeneous Platform", Friday Workshop on Designing for Embed-ded Parallel Computing Platforms: Architectures, Design Tools, and Applications at DATE 2012, Germany, 2012.

2. K. Siozios, D. Diamantopoulos, H. Sidiropoulos, A. Papanikolaou, and D. Soudris, Rapid Evaluation of 3-D Interconnection Schemes , DATE 2011 3D Integration Workshop, Grenoble, 2011, France.

1. M. Aviles, K. Siozios, D. Diamantopoulos, L. Nalpantidis, I. Kostavelis, E. Boukas, D. Soudris and A. Gasteratos, A Co-design Methodology for Imple-menting Computer Vision Algorithms for Rover Navigation onto Reconfigu-rable Hardware,Workshop on Computer Vision on Low-Power Reconfigurable Architectures, International Conference on Field Programmable Logic and Applications, Sept. 2011, Chania, Greece.

Awards

6. Cadence Design Systems: 2^{nd} award at contest "Cadence Thesis Contest for Automotive Embedded Systems", 2013.

5. Best Paper Award at 4th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures (PARMA) January 23, 2013, Berlin, Germany.

4. University Booth on Conference IEEE International Conference on Design Automation and Test in Europe (DATE 2013).

3. Journal Invitation "ACM Transactions on Embedded Computing Systems (TECS)" λόγω υψηλής βαθμολογίας κρίσης εργασίας στο "Workshop on Virtual Prototyping of Parallel and Embedded Systems (VIPES), 2013"

2. University Booth at conference IEEE International Conference on Design Automation and Test in Europe (DATE 2012).

1. HiPEAC: Grant for "International Summer School (7 days) on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems", Fiuggi, Italy, July 2011.

Curriculum Vitae

Dionysios Diamantopoulos born on February 12 of 1985 in Athens. After having national exams at the Lyceum Pylos Messinia, he was accepted in the Department of Computer Engineering and Informatics, Faculty of Engineering, University of Patras. In 2009 he received from that institution his diploma in Computer Engineering and Informatics.

In 2009 he was admitted to the PhD program of the School of Electrical and Computer Engineering, National Technical University. At the same time he joined the Microprocessors and Digital Systems Laboratory and he pursued his doctoral thesis in the research field of reconfigurable & application-specific embedded systems, under the supervision of Prof. Dimitrios Soudris.

From 2009 he works as a researcher at the Institute of Communication Systems & Computer Systems (ICCS) in European research projects. Also he has been working as a principal investigator in research projects of the European Space Agency (ESA) as well as in research projects of the National Strategic Reference Framework (NSRF). The aforementioned activities have provided financing of his doctoral thesis from European and national funds.

Up to date, Dionysios has published seven (7) articles in international scientific journals of IEEE, ACM, Elsevier and Hindawi and twenty-nine (29) papers in international conferences and workshops. From his publications he has received fourteen (14) citations. He has also co-supervised seven (7) undergraduates diploma theses. Additionally. Moreover he offers referee services to the research communities IEEE and ACM in international conferences and journals.

His research interests include reconfigurable architectures, the multi-objective optimization algorithms for EDA tools, the architectures for many-accelerators platforms and the technical high level synthesis for reconfigurable heterogeneous platforms.

Dionysios is a member of the Technical Chamber of Greece (TEE), European HiPEAC organization and the scientific community IEEE.