Impact of Hardware/Software Partitioning and MicroBlaze FPGA Configurations on the Embedded Systems Performances

Imène Mhadhbi, Nabil Litayem, Slim Ben Othman and Slim Ben Saoud

Abstract Due to their flexible architecture, lower-cost and faster processing, Field Programmable Gate Array (FPGA) presents one of the stimulating choices for implementing modern embedded systems. This is due to their intrinsic parallelism, fast processing speed, rising integration scale and lower-cost solution. This kind of platforms can be considered as a futuristic implementation platform. The growing configurable logic capacity of FPGA has enabled designers to incorporate one or more processors in FPGA platform. In contrast to the traditional hard cores, the soft cores processors present an interesting solutions for implementing embedded applications. They give designers the ability to adapt many configurations to their specific application; including memory subsystems, interrupt handling, ISA features, etc. Faced to the various problems related to the selection of an efficient soft-core FPGA embedded processor with appropriate configuration, co-design methodology presents a good deal for embedded designers. The most crucial step in the design of embedded systems is the hardware/software partitioning. This step consists of deciding which component is suitable for hardware implementation and which one is more appropriate for software implementation. This research field is especially active (always on the move) and several approaches are proposed. In this chapter, we will present our contribution on the hardware/software partitioning co-design approach, and discuss their involvement on design acceleration and architecture performances. The first part of this chapter describes the effect of the MicroBlaze Xilinx configuration on the embedded system performance. The second part

I. Mhadhbi (✉) · S.B. Othman · S.B. Saoud
LSA Laboratory, INSAT-EPT, University of Carthage, Tunis, Tunisia
e-mail: imene.mhadhbi@gmail.com

S.B. Othman
e-mail: boslim@yahoo.fr

S.B. Saoud
e-mail: slim.bensaoud@gmail.com

N. Litayem
Department of Computer Science, College of Arts & Science, Salman Bin Abdelaziz University KSA, Al-Kharj, Saudi Arabia
e-mail: nabil.litayem@gmail.com

Q. Zhu and A.T. Azar (eds.), *Complex System Modelling and Control Through Intelligent Soft Computations*, Studies in Fuzziness and Soft Computing 319,
DOI 10.1007/978-3-319-12883-2_25

introduces our new hardware/software partitioning approach on a complex secure lightweight cryptographic algorithm. This work can contribute to enforce the security of SCADA (Supervision Control and Data Acquisition) systems and the DSS (Digital Signal Standard) without compromising the cost and the performance of the final system.

1 Introduction

Embedded systems are now present in practically all domestic and industrial systems (appliances and applications) such as cellular telephones, personal digital assistants (PDAs), digital cameras, Global Positioning System (GPS) receivers, defense systems and security applications. The increased complexities of embedded systems and their real-time operation's constraints allow semiconductor markets to build other solutions for processing. Traditionally, embedded systems were designed and implemented using Microprocessors (MP), Microcontrollers (MCUs), Digital Signal Processors (DSPs), Application-Specific Integrated Circuits (ASICs) and FPGAs. Due to their advantages, FPGAs have substituted DSPs in different applications such as motor controllers (Arulmozhiyal 2012; Xiaoyin and Dong 2007) which are widely used in industrial applications, image processing (Kikuchi and Morioka 2012), wireless (Jing-Jie and Rui 2011; Nasreddine et al. 2010), automotive and aerospace systems. Continuing increases in FPGA performance, capability and architectural features are enabling more embedded systems designs to be implemented using FPGAs. Additionally, FPGAs costs are decreasing, for less than $12, allowing designers to incorporate FPGAs circuits with one million equivalent gates. This made the implementation of Programmable System-On-Chip (SoPCs) possible what also allowed this implementation their pipeline ability, intrinsic parallelism and flexible architecture (Jianzhuang et al. 2008). FPGAs offer a faster processing speed, a lower-cost solution and more functionalities to support more innovative characteristics.

Nevertheless, the increasing complexity of algorithms and the rising integration scale on FPGAs triggered designers into drastically improving design methodologies. In addition, the effort to design complex applications on FPGA is generally much more complicated than impelementing them on programmable processors.

The real challenge, as far as the embedded systems designers are concerned, is how to increase performances (execution time, area and energy consumption) of complex systems and reduce their complexity, and refinement time.

Many interesting design methodologies are presented. Some designers have based their methodologies on reducing development time to implement complex embedded systems. Among the many approaches that have been adapted there is first the automatic transformation of the behavioral system description into structural netlist system components using high level input language such as SpecC

(Fujita and Nakamura 2001) and Bluespec (Dave et al. 2005; Gruian and Westmijze 2008; Talpin et al. 2003). Second, there is Hardware In the Loop (HIL) technique which increase the tractability and earlier testability of the design product (Washington and Dolman 2010). The automation of the hardware/software partitioning step on the co-design methodologies using low-level specification presents the third approach (Stitt et al. 2003).

Other designers have based their methodologies on minimizing the design complexities. One approach is the use of Intellectual Proprieties (IPs) blocs and cores (Mcloone and Mccanny 2003) provided by vendors or designers (Lach et al. 1999). An other is the automating of the hardware code generation HDL (Hardware Description Language) from a high level specification (Samarawickrama et al. 2010; Ku and De Mitcheli 1992). This specification can be defined as language (C, SystemC, etc.) or models (Matlab, Sycos etc.) or UML (Unified Modeling Language) diagrams, called HLS (High Level Synthesis) approach (Lingbo et al. 2006; Wakabayashi and Okamoto 2000).

During last decades, early designers' works have been focused on new contributions of the existing design methodologies, which allow both the high level specification and the automation of the design process to decrease the systems complexities, reduce the development time to enlarge the time reserved to the optimization and increase their performance. None of these approaches deals with the impact of the best configuration selection of soft-cores processors performance in terms of computation acceleration.

The goal of this chapter is to actively contribute to the existing co-design approaches including hardware/software partitioning step using high level specification. The chapter also aims at adding a step to the selection f the best soft-cores processors configuration. These contributions permit the increase of embedded systems performance (soft-cores computation) and the reduction of systems complexities. The remaining parts of this paper are organized as follows: Sect. 2 illustrates the related works and background of design methodologies. Section 3 presents the MicroBlaze soft-core processor. Section 4 depicts our co-design approach. Section 4 presents the performance evaluation techniques and the lightweight cryptographic algorithm. Section 5 determines results of our co-design approach. In Sect. 6, we will discuss results. Finally, Sect. 7 summarizes our study, and gives our perspectives.

2 Related Works and Background

In this section, the different steps of design methodologies will be presented in reverse. We will begin by defining the different architecture of implementation. We will proceed with the design methodologies approaches specifically HLS approach and hardware/software partitioning approach. Finally, system level specification of embedded systems will be dealt with.

2.1 Design Methodologies, Challenges

Embedded applications require increasingly sophisticated functionalities and severe constraints. They incorporate many application areas such as telecommunication, avionics, automotive, medical implants, domestic appliances, etc. These increasing complexities require functional constraints (computation capacities, reduced power consumption, miniaturization of the implementation area, etc.) and non-functional constraints (minimum time-to-market, reduced cost, maximum life, growth in the amount of productions, etc.). To increase the embedded systems performances, researchers and industry have focused on two areas of research. First, technological area that is based on the evolution of the integration level of integrated circuits. Second, methodological area, which is based on refinement of design methodologies. Faced with the physical limits of technical evolution, manufacturers of embedded systems had to demonstrate a different reactivity. They had to continuously improve their techniques and design approaches to increase the embedded systems performances.

In our study, we examine different design architectures of complex embedded systems. Our contribution lies in the hardware/software partitioning step starting from high level specification. Also, a new step has been added to the hardware/software partitioning which is the selection of the best configuration of the used soft-core processor.

2.1.1 Design Implementation Architectures

Traditional hardware FPGA design approaches are complex. This reduces FPGA productivity. Hardware implementation uses a low-level specification, VHDL or Verilog languages or combination of both, to implement embedded applications. Their implementation process consists of the (a) definition of application at a low-level specification (b) synthesis, (c) implementation, (d) simulation and (e) tests and verification steps. With the integration of soft-cores processors into FPGA, designers become able to implement complex systems on software architecture. As input, they employ a high-level specification, compile it and implement it into soft-cores processors.

Several researches demonstrate that software implementation of embedded systems allows flexibility (ability to modify specifications), ease of integration, reduction of design time and bad performances. However, hardware implementation of the same application greatly achieves high performance constraints in a long design time.

Now, FPGA offers many advantages. It can be used in all embedded systems fields (image processing, aerospace systems, security and industrial applications, etc.). It can be implemented on different architectures (hardware, software or both hardware/software) using different design methodologies (Joven et al. 2011) as illustrated on Fig. 1.

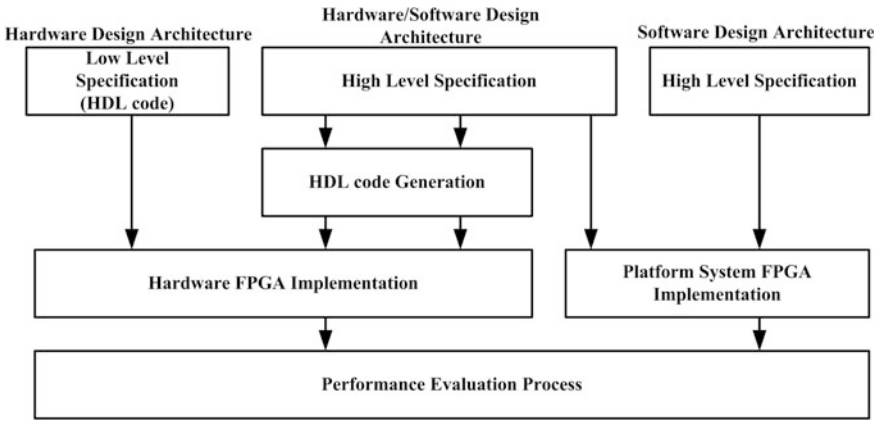

Fig. 1 FPGA architectures and methodologies design

Recently, designs approaches can be implemented using both hardware/software architectures using co-design methodology to accelerate the design process. Using this methodology, designers can incorporate co-processors, hard-cores processors and soft-cores processors. This decision of integration is taken after a hardware/software partitioning step. Hardware/Software partitioning is usually related to physical constraints (computing time, energy consumption, level of integration, area utilization) and economic constraints (cost, flexibility, design time and Time-To-Market) embedded systems constraints, as described in Table 1.

Recently, Reconfigurable devices, such as FPGAs, become highly appealing circuits for co-design methodology as they provide flexibility and ability to easily implement complex embedded applications. Using co-design methodologies, designers permit the integration of both hardware and software architectures into FPGA (Kalomiros and Lygouras 2008). Xilinx proposes its own co-design methodology using Xilinx EDK (Integrated Development Kit) environment. EDK includes both an integrated development environment (IDE) named Xilinx Platform Studio (XPS) and Software Design Kit (SDK). XPS tool allows the implementation on hardware architecture and the creation of a Microprocessor Hardware Specification (MHS) file. SDK tool permit the implementation of software architecture and the creation of the Microprocessor Software Specification (MSS) file. The MHS file defines the embedded system processor, architecture and peripherals. The MSS file defines the library customization parameters for peripherals, the processor customization parameters, the standard I/O devices, the interrupt handler routines, etc. Figure 2 depicts the co-design flow of Xilinx EDK tool.

2.1.2 Hardware/Software Partitioning Approaches

FPGAs present powerful circuits for prototyping embedded system applications, supporting both software and hardware architectures. The choice of architecture is

Table 1 Comparative studies of the software/hardware design architectures

		Software	Hardware
Physical Constraints	Execution time		*
	Energy consumption		*
	Integration		*
	Area		*
Economic Constraints	Cost	*	(Except high volume)
	Flexibility	*	
	Design time	*	
	Time-to-market	*	

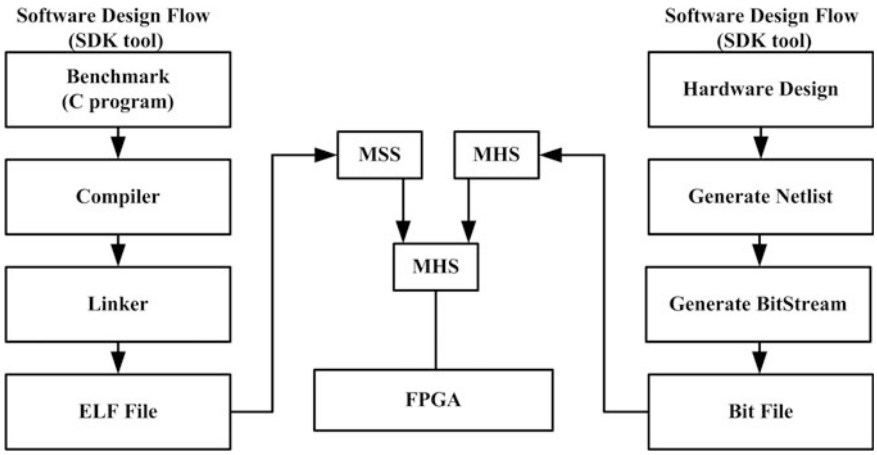

Fig. 2 Co-design flow using Xilinx EDK tool

based on the hardware/software partitioning step. The goal of that partitioning step is to determine which components of the application are suitable for hardware or software implementation. Hardware implementation is desirable to design efficient embedded systems in term of execution time and computation (co-processors). However, software implementation gives less performance in a reduced time. This partitioning is depending on embedded systems constraints such as cost, efficiency and speed. The real paradigm of co-design methodology is the great choice of hardware and software sections.

Co-design approaches promote the implementation of efficient embedded systems in a low development time by integrating hardware co-processors into software design process. During the design process, fundamental decisions have dramatically influenced the quality and the cost of the final solution. Design decisions have an impact of about 90 % of the overall cost. The most important decision is that of hardware/software partitioning.

Therefore, Partitioning is a well-known problem. During the last years, many partitioning approaches have been proposed to automate the partitioning process decision of hardware and software components (De Michell and Gupta 1997; Wiangtong et al. 2005). The feasibility of these approaches depends essentially on the system-level specification, the target architecture and the constraints parameters (hardware size, power consumption, execution time, computation, etc.). Several works were focused on the automation of the hardware/software partitioning using co-design methodologies. Many interesting approaches are presented. Some of them are described on Table 2.

As described in the table above, many partitioning hardware/software approaches exist (Madsen et al. 1997; Boßung et al. 1999; Chatha and Vemuri 2000). From the many co-design approaches, we will examine some of these. A hardware/software partitioning approach is proposed by Lysecky and Vahid (2004). This approach uses a relaxed cost function to satisfy performance in an Integer Linear Programing (ILP); it handles hardware minimization in an outer loop. Lysecky and Vahid (2004), presents a binary constraint search algorithm which determines the smaller size constraint. Vahid partitioning approach minimizes hardware, but not execution time. Kalavade and Lee (1994), proposed also a different hardware/software partitioning approach. It is based on GCLP algorithm to determine for each node iteratively the mapping to hardware or software. The used GCLP algorithm selects its appropriate objective according to critical time measure and another measure for local optimum.

Table 2 Hardware/software partitioning approaches

Approaches	Cosyma	Vulcan	Polis	CoWave	GrapeII
Specification	SDL language	HardwareC	FSM, esterel	DFL, C, etc.	DFL
Internal model	No	No	Yes	Yes	Yes
Support Y chart model?	No, Semi-automatic	Yes, with migration	No, manual	No	Yes
Support automating partitioning?	No	No	Yes (Y-chart like)	No	Yes (Y-chart like)
Supports the exploration of the design space?	Low	Low	Very high	High	Medium
Level specification of approach	No	No	Yes	No	No
Support for synthesis	No	No	Yes	No	Yes
Target architecture	Mono-processor: CPU + Co-processor	Mono-processor: CPU + ASIC with buses	Mono-processor	Multi-processors with ASICs	Multi-processors with FPGAs

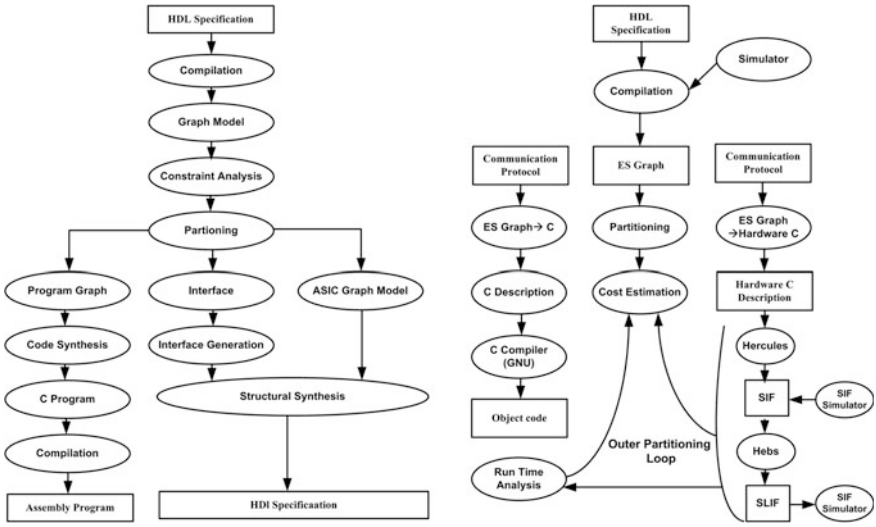

Fig. 3 Vulcan and cosyma approaches

Two representative approaches directly affecting the research of this chapter are Vulcan (Wolf 2003) and Cosyma (Co-synthesis embedded architecture) approaches (López-Vallejo and López 2003). Both Vulcan and Cosyma use partitioning approach, which iterates over hardware synthesis and software generation. Iteration, in these approaches, is necessary because there are no approaches known to accurately estimate the results of optimizing compilers and high-level synthesis tools with advanced techniques. While Vulcan is hardware oriented, starting with an all hardware implementation and moves operations to software on a given processor until time constraints are verified, COSYMA is software oriented, starting with an all software implementation on a given processor and moves operations to hardware until no time constraint is verified any more. Several studies employ these approaches to automate their co-design approach (Henkel and Ernst 1998; Gupta et al. 1992). Figure 3 illustrates these two co-design approaches.

The automation of hardware/software partitioning process allows the classification of embedded specification to determine which components can benefit from the transformation to hardware and the best configuration for getting an optimal gain of performance. Transformations of hardware nodes are provided using HLS approaches. In the next sub-section, we will introduce the HLS design approaches.

2.1.3 HLS Approaches

Using HLS approaches, complexities are managed by (a) starting the design process at a higher level of abstraction, (b) automating the hardware code generation, and (3) reusing intellectual components (IPs). Reducing the migration time from a high-level

language specification to a hardware specification language presents the main objective of designers. Early works are focused on how to faster prototyping speed with automatization of the Register Transfer Level (RTL) generation process from the high level behavioural description using commercial tools (Feng et al. 2009).

During the last decades, HLS design approaches have been the main subject for research. Their principal objective is to simplify the accelerators hardware design by describing applications at high abstraction levels and generating the corresponding description of a low-level implementation. Different studies were focused to qualify the benefits of implementing HLS methodologies in terms of time-to-market, execution time and area consumption.

Thangavelu et al. (2012) evaluate the Model-Based Design approach, using XSG (Xilinx System Generator). However, Abhinvar compares the HLS approach (C-Based Design), using Catapuls, with Bluespec design approach, in order to prove that HLS is the most efficient in stage of the design development for fast prototyping complex systems (Dave et al. 2005). Indeed, it offers reduced design time and provides a generic design compared to the Bluespec design flow that generates hardware code adapted to the performance constraints and resources. The Rapid prototyping of complex systems are founded on HLS approaches such as C-Based Design approach (Dave et al. 2005), Model-Based Design approach and Architecture Based Design approach (Cherif et al. 2010) to raise their productivity (from higher levels of abstraction) and their reliability (from automatic code and test bench generation and more robust test technologies).

Model Based Design Approach

The Model-Based Design approach accentuates the use of models to increase the abstraction level of the complex systems (Lingbo et al. 2006; Wakabayashi and Okamoto 2000). This approach represents a real process of evolution in the embedded systems design. The model used, in the systems engineering, includes safety critical areas such as aerospace, automotive, etc. It is applied not only for the explanation of algorithms, but likewise, for the generation of VHDL code. The Model-based design approach level of abstraction is very high, which allows the flexibility to add, delete and modify applications in a short design time. Using this approach, designers can automate the generation, from a model to a synthesized hardware code (VHDL or Verilog), ready to be implemented on FPGA. Mode-Based Design approach emphasizes the usage of models to increase the level of abstraction to design complex systems. It allows the modelization and verification of each function separately using a low-level language or blocks. The ability to plot the progress of the application using Model-Based Design presents an advantage to detect the wrong behaviour. The design model used in the systems engineering, includes also safety critical domains like aerospace and automotive. One of the most widely used tools in these domains is Sicos-HDL, FPGA-module (LabView), Syndex-Ic and XSG.

Architecture-Based Design Approach

The Architecture Based Design approach permits an automatic generation of a synthesized hardware code (VHDL or Verilog), ready to be implemented in FPGA, from UML diagrams. The rapid prototyping approach should provide a way to accelerate the hardware language generation. It must satisfy the following features: (i) Flexibility analysis to produce different results with minimum changes such as the computing precision. (ii) Accuracy of results. The abstraction level of the Architecture Based Design approach, compared to a code written with C in the C-Based design approach and a model described in the Model-Based Design, is very high. This allows the flexibility to add, delete and modify the applications in a short design time. The efficient implementation of complex algorithms (such as a light-weight cryptographic application) in a hardware circuits (FPGA) allows a faster processing speed (parallelism) and more functionalities to support more advanced features.

C-Based Design

This approach consists in the automatic generation of hardware code like VHDL or Verilog, from a C/C++ language, ready to be implemented on FPGAs (Dave et al. 2005). Recent development of C-to-HDL tools technology has minimized the gap between software developer's experience-level, and the expertise needed to produce hardware applications. Many commercial and academic C-Based Design tools can be found in the literature: Catapult-C (Mentor Graphics), CoDeveloper™, C2H, SPARK. In this study, we chose the CoDeveloper™ tool to implement complex embedded application using hardware architecture.

2.2 System Level Specification

The choice of hardware/software partitioning, using co-design approach, presents a trade-off among various design metrics such as performance, cost, flexibility and time-to-market (López-Vallejo and López 2003; Joven et al. 2011). Several approaches of hardware/software partitioning are presented. Their classification is based on their input specifications which is it based on models or languages.

2.2.1 Model Specification

Stoy and Zebo (1994) groups indicate that initial specification can be defined as models of components such as a Finite State Machine (FSM), Discrete-Event Systems, Petri Nets, Data Flow Graphs, Synchronous/Reactive Model, and Heterogeneous Models. These models are described in the next sub-sections.

Finite State Machine (FSM)

Finite State Machine (FSM) models contain sets of states, inputs, outputs, output functions, and next-state functions. Embedded applications are described as a set of states and input values, which can activate a transition from one state to another. FSMs are usually used for modeling the control-flow dominated systems. To avoid limitations of the classical FMS, researchers have proposed several derivatives of the FSM. Some of these extensions are used in several tools such as SOLAR (Ismail et al. 1994), Hierarchical Concurrent FSM (HCFSM) (Reynari et al. 2001) and Co-design Finite State Machine (CFSM) (Cloute et al. 1999).

Discrete-Event Systems

In a Discrete-Event System, the occurrence of discrete asynchronous events triggers the transitioning from one state to another. An event is defined as an instantaneous action, and has a timestamp representation when the event took place. Events are sorted globally according to their time of arrival. A signal is defined as a set of events, and it is the main method of communication between processes (Stoy and Zebo 1994). Discrete Event modeling is often used for hardware simulation. For example, both Verilog and VHDL use Discrete Event modeling as the underlying model of Computation. Discrete Event modeling is expensive since it requires all events according to their timestamp.

Petri Nets

Petri Nets is widely used for modeling systems. Petri Nets consists of places, tokens and transitions where token are stored in places. Transition causes tokens are stored in places. Transition causes tokens to be produced and consumed. Petri Nets supports concurrency and is asynchronous; however, they lack the ability to model hierarchy. Therefore, it can be difficult to use Petri Nets to model complex systems due to its lack of hierarchy. Variation of Petri Nets has been devised to address the lack of hierarchy, such as the Hierarchal Petri Nets (HPNs) proposed by Dittrich. Hierarchical Petri Nets (HPNs) supports hierarchy in addition to maintaining the major Petri Net's features such as concurrency and asynchronously. HPNs use directed graphs as the underlying model. HPNs are suitable for modeling complex systems since they support both concurrency and hierarchy.

Data Flow Graphs

Data Flow Graphs (DFG) systems are specified using a directed graph where nodes (actors) represent inputs, outputs and operations and edges represent data paths between nodes (Reynari et al. 2001). The main usage of Data Flow is for modeling

data flow dominated systems. Computations are executed only where the operands are available. Communication between processes is done via unbounded FIFO buffering Scheme (Stoy and Zebo 1994). Data Flow models support hierarchy since the nodes can represent complex functions or other Data Flow.

Several variations of Data Flow Graphs have been proposed in the literature such as Synchronous Data Flow (SDF) and Asynchronous Data Flow (ADF). In SDF, a fixed number of tokens are consumed, where in ADF the number of tokens consumed is variable.

Synchronous/Reactive Models

Synchronous modeling is based on the synchrony hypothesis. Outputs are produced instantly in reaction to inputs and there is no observable delay in the outputs. Synchronous models are used for modeling reactive real-time Systems. Stoy and Zebo (1994) mentioned two styles for modeling reactive real time systems. First multiple clocked recurrent systems (MCRS) which are suitable for data dominated by real time systems. Second, state base formalisms which are suitable for control dominated real time systems. Synchronous languages, such as Esterel, are used for capturing Synchronous/Reactive model computation.

Heterogeneous Models

Heterogeneous Models combine features of different models of computations. Two examples of heterogeneous models are presented: Programming languages and Program State Machine (PSM). Programming languages provide a heterogeneous model that can support data, activity and control modeling. Two types of programming languages are presented, imperative language such as C, and declarative languages such as LISP and PROLOG. In imperative languages, statements are executed in the same order specified in the specification. On the other hand, execution order in not specified in declarative languages since the sequence of execution is based on a set of logic rules or functions.

Program State Machine (PSM) is a merger between HCFSM and programming languages. The Spec Charts language, which was designed as an extension to VHDL, is capable of capturing the PSM model. The SpecC is another language capable of capturing the PSM model. The following Table 3 attempts to set a comparison between different models of computation.

2.2.2 Specification Using Language

The goal of a specification using language is to describe the intended functionality of non-ambiguous systems. A large number of specifications using languages are currently being used in embedded system design since there is no language that is

Table 3 Comparison of various models of computation

MOC	Origin MOC	Main application	Clock mechanism	Orientation	Time	Communication method	Hierarchy
SOLAR	FSM	Control oriented	Synch	State	No explicit time	Remote procedure call	Yes
HCSFM state charts	FSM	Control oriented/reactive real time	Synch	State	Min/Max time spent in state	Instant broadcast	Yes
CFSM	FSM	Control oriented	Async	State	Events w/t time stamp	Wire signals	Yes
Discret event	N/A	Real time	Synch	Timed	Globally sorted events w/t time stamp	Wired signals	No
HPN	Petri net	Distributed	Async	Activity	No explicit timing	N/A	Yes
SDF	DFG		Synch	Activity	No explicit timing	Unbounded FIFO	Yes
ADF	DFC		Async	Activity	No explicit timing	Bounded FIFO	Yes

MOC Model of Compilation

the best for all applications. Below is a brief overview of the widely used language specification.

- Formal Description Languages such as LOTOS (based on process algebra, and used for the specification of concurrent and distributed systems) and SDL (used for specifying distributed real time systems, and based on extended FSM).
- Real Time Languages such as Esterel (a synchronous programming language based on the synchronous hypothesis. They are used for specifying real time reactive systems. Esterel is based on FSM, with constructs to support hierarchy and concurrency) and StateCharts (the graphical specification using languages used for specifying a reactive system). StateCharts extend FSM by supporting hierarchy, accuracy and synchronization.
- Hardware Description Languages: Commonly used HDL are a VHDL (IEEE standardized hardware description language), Verilog (hardware description language, which has been standardized by the IEEE) and HardwareC (a C based language designed for hardware synthesis). It extends C by supporting structural hierarchy, concurrency, communication and synchronization.

3 FPGA Cores Processor

The emergence of soft-cores processors (implemented using logic General Purpose programmable and synthesized onto FPGA) and hard-core processors (available as embedded blocks in the silicon next to the FPGA) inside FPGA increases their efficacy. FPGAs can include various embedded processors, different communications buses, many peripherals and network interfaces. It is possible now to create a complete hardware/software system with I/O and control interfaces on a single chip (SoC). This coexistence improves the embedded system performances by reducing the communication between external processors and FPGA circuit.

Embedded systems architectures allow the coexistence between hardware and software processors working together to perform a specific application. Usually, they can contain embedded processors who are often in the form of soft-core processors (described at a higher level of abstraction, implemented and synthesized to target a given FPGA or ASIC technology) and hard-core processors. Despite the advantages of the use of hardware processors (small area and power consumption), designers of embedded systems choose the implementation using soft-core processors due to their many advantages and their different configurations. Soft-core processors offer many hardware configurations to accelerate the execution time (e.g. adding floating-point hardware as hardware components into the soft-core processor) in terms of cost, flexibility, configuration, portability and scalability.

Atmel (FPGA vendors) and Triscend firms began introducing hard-core processor on their FPGA circuits, basing on an efficient communication mechanism between hard-core and FPGA components. More recently, Altera has offered Excalibur devices hard-core using ARM9 processor, NIOS and recently NIOS II soft-cores.

Xilinx firm has proposed the Virtex II Pro device with two or more PowerPC and tens millions of programmable gates and both PicoBlaze and MicroBlaze soft-cores. OpenCore has presented OpenRISC soft-core (Bolado et al. 2004) and Gaisler Research has given LEON and LEON2 soft-cores (Denning et al. 2004). In our study, the partitioning of software/hardware components was tested on Virtex-5 FPGA circuit, which allows the integration of various MicroBlaze soft-cores processors. In this chapter, embedded soft-core processor architecture, as being examined, consists of the Xilinx MicroBlaze soft-core processor.

3.1 Xilinx MicroBlaze Soft-Core Processor Architecture

Embedded processors can be defined as software cores implemented in hardware circuits using Logic General Purpose Programmable. The most used soft-cores processors, in the designing of embedded system for Xilinx FPGA, is the Xilinx's MicroBlaze soft-core processor. MicroBlaze is a 32-bit Reduced Instruction Set Computer (RISC) architecture optimized for synthesis and implementation into Xilinx FPGAs with a separate 32-bit instruction and data buses to execute programs and access data from both on-chip and external memory at the same time. This processor includes 32-bit general-purpose registers, virtual memory management, cache software support, and FSL interfaces. It has Harvard memory architecture and uses: Two Local Memory Busses (LMB) for instruction and data memory, two-Block RAMs (BRAM) and two peripherals connected via On-chip Peripheral Bas (OPB). Three memory interfaces are supported: Local Memory Bus (LMB), the IBM Processor Local Bus (PLB), and Xilinx Cache Link (XCL): The LMB offers single-cycle access to on-chip dual-port block RAM. The PLB interfaces offer a connection to both on-chip and off-chip peripherals and memory. The CacheLink interface is proposed for use with specialized external memory controllers. The architecture of the Xilinx MicroBlaze FPGA processor, the interfaces, buses, memory, and peripherals are shown in Fig. 4.

The major advantage of choosing MicroBlaze soft-core processor, in our researches, is its higher performance and its various configurations.

3.2 Xilinx MicroBlaze Soft-Core Processor Features

The MicroBlaze Xilinx processor offers tremendous flexibility during the design process. It allows different configurations to meet the needs of their design embedded applications by adding or removing some setting parameters such as:

- Integer Multiplier Units: Add the Integer multiplication as a co-processor.
- Barrel Shifter Units: Add the Shift by bit operations as a co-processor.
- Integer Divider Units: Add the Division of Integer as a co-processor.

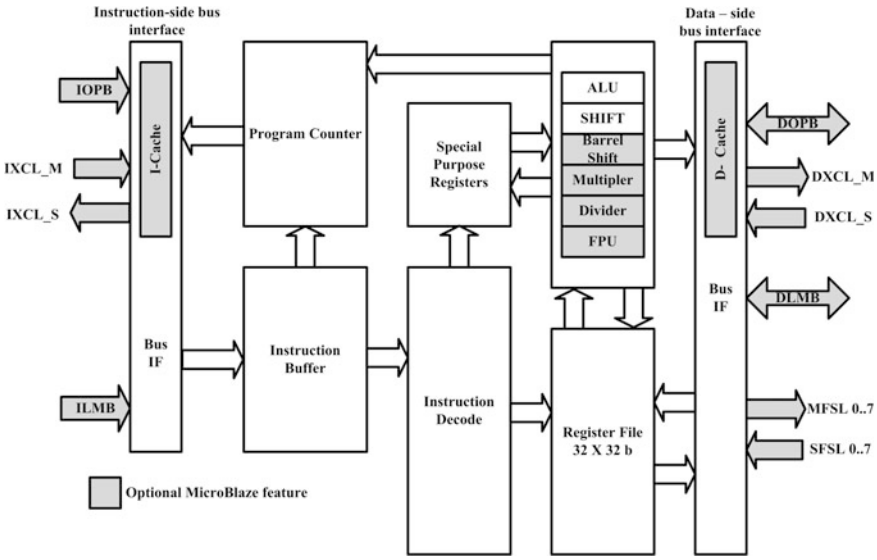

Fig. 4 Microblaze functional block diagram

- Floating-Point Units: Add Basic and Extended precision as a co-processor.
- Machine Status Register Units: Add Set and clear machine status register as a co-processor.
- Pattern Compare Unit: Add the String and pattern matching as a co-processor.

However, the designers need to select an appropriate configuration according to the application to improve the system performances. Thus, performance evaluation main function is to help embedded systems designers to answer the following questions: Does design methodology influence on the embedded system performances? Does a particular configuration affect the performance of the embedded system? How fast is the design process? What are the limits of the improvement of the design process? In the next sub-section, we will start t present our evaluation design approaches.

4 Proposed Design Approaches

The great issue of FPGA designers is that they are faced with the various problems for selecting the best architecture, the greatest hardware/software partitioning and the finest configuration of the selected soft-core processor. All these difficulties choice are constrained by execution time and area consumption. To take a decision about the final architecture design, designers need to proceed to a performance evaluation step. In our work, we propose to accelerate co-design methodology by

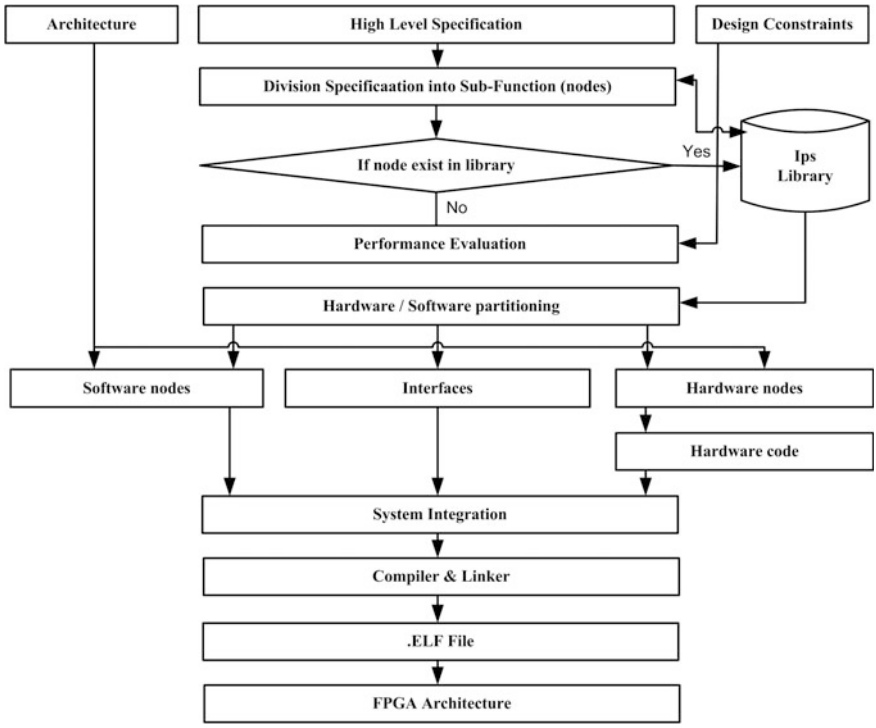

Fig. 5 Proposed co-design methodology approach

automating hardware/software partitioning step (basing of the hardware/software costs) using a high-level specification. Figure 5 illustrates our design methodology.

The low-level specification, proposed practically by all hardware/software partitioning approaches is replaced, in our approach, by a high-level specification. This high-level specification is divided into functional nodes (C functions) defined as nodes to make possible its integration on the hardware or software architecture. Beginning with a high-level specification, in the hardware/software partitioning step permits the classification of nodes on software or hardware without specifying the implementing target which allows the portability of our design process. Before partitioning, designers have to evaluate the costs of nodes (in term of execution time and area consumption) and the time taken for communication between software and hardware nodes. For software nodes, these costs are computed using profiler (e.g. compiling c code on MicroBlaze using the directive -pg, permit the generation of the profiling of each C function). However, hardware costs are measured after hardware synthesis of the high-level specification using HLS approaches. As hardware/software algorithms partitioning, we selected the Integer Linear Programing (ILP) algorithm. Figure 6 details our approach on hardware/software partitioning.

To evaluate our approach, software nodes are executed on the MicroBlaze soft-core processor with its different configurations. However, the implementation of hardware tasks is carried out, using HLS approaches. This tool allows fast prototyping of Intellectual Proprieties (IPs) that will be added to the MicroBlaze by the Fast Simplex Link (FSL) interface using Xilinx EDK tool.

5 Performance Evaluation Process

The performance evaluation of embedded systems has multiple aspects depending on the application that the system is made off. Hence, performance measurement is involved in several stages of the design process. In this chapter, we propose to evaluate the performance of the MicroBlaze FPGA soft-core processor features, in a first time, then that of our proposed design methodology, in a second time, using a lightweight cryptographic application.

5.1 Performance Evaluation Technique

Performance evaluation is the process of predicting whether the designed system satisfies the performance goal defined by the user such as area consumption and execution time (Mysore et al. 2005; Monmasson and Cristea 2007; Li and Malik 1995). Performance evaluation can be classified into two categories: Performance modeling and performance measurements as mentioned on Table 4.

5.1.1 Performance Modeling

Performance modeling approach is concerned with architecture-under-development. It can be used at an early stage of the design process where the processor is not available, or it is very expensive to prototype all possible processors architectures choices. Performance modeling may be classified into analytical-Based approach and Simulation-Based approach.

Analytical-Based Approach

The analytical modeling approach is based on probabilistic methods. Petri nets or Markov models create mathematical models of the designed embedded systems. The results of this approach are not often easy to construct. It allows predicting mainly user performance, time execution of sub-functions rapidly without compilation or execution. There has not been much study on the analytic approach for processors. Processors' structures are so complex that few analytical models can be provided for

Table 4 Performance evaluation techniques

CPU benchmarks	Synthetic benchmarks	
	Application based benchmarks	
	Algorithm based benchmarks	
Performance measurement	MP-on chip performance monitoring counters	
	Off-chip Hw monitoring	
	SW monitoring	
	Micro-coded instrumentation	
Performance modeling	Simulation	Trance driven simulation
		Execution driven simulation
		Complete system simulation
		Even driven simulation
		Software profiling
	Analytical model	Probabilistic models
		Queuing models
		Markov models
		Petri net models

them. Some research efforts are presented by Noonburg and Shen (1997) using a Markov models to model a pipelined processor, when Sorin et al. (1998) used probabilistic techniques to model a Multi-processor composed by superscalar processors.

Simulation-Based Approach

Simulation-Based approach presents the best performance modeling method in the performance evaluation of processor architectures. Model of the processor being simulated must be written in a high-level language, such as C or Java and running on some existing machine. Simulators give performance information in terms of cycles of execution, cache bit ratios, branch prediction rates, etc. Many commercial and academics simulators are presented: The SinOS simulator which presents a simple pipeline processor model and a powerful superscalar processor model. The SIMICS simulator simulates uni-processor and multi-processor models. Results of simulation approaches are not very interested in the performance evaluation of the MicroBlaze Xilinx soft-core processor because they are not exact.

5.1.2 Performance Measurement

Performance measurement approach is used for understanding systems that are already built or prototyped. Two major purposes for performance measurement approach can be used to tune systems to be built in order to understand the

bottlenecks of such system. Performance measurement adjusts the application if its source code or algorithms can still be changed in order to understand the applications. This application can run on the system and tune the different design configurations. This kind of performance evaluation approach can be done using the following means:

- Microprocessor on-chip performance monitoring: can be used to understand performance of high microprocessors (Intel's Pentium III and Pentium IV, IBM Power3 and Power4 processors, AMD's Athlon, Compaq's Alpha and Sum's Ultra SPARC). Several tools are available to measure performance monitoring counters: Intel's Vtune software can be used to perform measurement when the Intel performance counters. The P6Pref utility presents a plug-in for Windows NT performance monitoring. The Compaq DIGITAL Continuous Profiling Infrastructure (DCPI) presents a very powerful tool used to profile program on the Alpha processors.
- Off-Chip hardware monitoring: Instrumentation using hardware wherewithal can be done by attaching off-chip hardware. Example Speed Tracer from AMD and Logic analyser. AMD developed hardware-trading platform to help in the design of X86 microprocessors. However, Poursepanj and Christie used a logic analyser to analyze 3D graphics workloads on AMD-K6-2 based systems.
- Software monitoring: is an important mode of performance evaluation used before the advent of on-chip performance monitoring counters. The primary advantage of software monitoring is that it is easy to execute.
- Mircocoded instrumentation: is a technique lying between trapping information on each instruction using hardware interrupts (traps) or software interrupts (traps). The tracing system modified the VAX microcode to record all instructions and data references in a reserved portion of memory.

5.1.3 CPU Benchmarks

Designers of FPGA processor have to use the CPU Benchmarks approach to get a fixed measurement of the processors 'performance, which is attempting to implement and verify the architectural and the timing behavior under a set of benchmark programs. Several open sources and commercial benchmarks are presented. Some of them are: Mibench, Paranoia, LINPACK, SPEC (Standard Performance Evaluation Corporation), and EEMBC (Embedded Microprocessor Benchmark Consortium). These Benchmarks are divided into three categories depending on the application (Korb and Noll 2010). The first category is Synthetic Benchmark (with the intention to measure one or more features of systems, processors, or compilers). The second category is application based benchmarks or "real world" benchmarks (developed to compare different processors' architectures in the same fields of applications). Finally, the third category is Algorithm Based Benchmarks (developed to compare systems architectures in special (synthetic) fields of application).

Synthetic Benchmarks

Synthetic Benchmarks are developed to measure processor specific parameters. Synthetic benchmarks are created with the intention to measure one or more features of systems, processors, or compilers. It tries to mimic instruction mixes in real-world applications. However, it is not related to how that feature will perform in a real application. Dhrystone and Whetstone benchmarks are the most-used synthetic benchmarks.

Application Based Benchmarks

Application Based Benchmarks or "real world" benchmarks are developed to compare different processor architectures in the same fields of applications. Application based or "real world" benchmarks use the code drawn from real algorithms, and they are more common in system-level benchmarking requirements.

Algorithms Based Benchmarks

Algorithm Based Benchmarks: (a compromise between the first and the second type) developed to compare systems architectures in special (synthetic) fields of application. Several studies are based on this approach to evaluate the processors' performances. Bolado et al. (2004) evaluated three soft-cores processors namely LEON2, MicroBlaze and OpenRISC to measure the execution time and the area consumption, using Dhrystone and Standford benchmarks. Berkeley Design Technology, Inc. evaluated the performance of the Texas Instruments' DSCs processors to compute the execution time using the Fast Fourier Transform (FFT) algorithms using fixed-point and floating-point data precision. Korb and Noll (2010) examined the performance of both DSPs and MCUs basing on the execution time of a number of benchmark codes included fixed-point and floating-point math operations, logic calculation, digital control, FFT, conditional jumps and recursion test algorithms. In our paper, we have chosen to adopt the performance measurement method using freely benchmark solutions. We used lightweight cryptographic secure application as a benchmark.

In the next section, we will introduce our used benchmark: The lightweight cryptographic application: Quark Hash Algorithm.

5.2 Lightweight Cryptographic Benchmarks: Quark Hash Algorithm

The need for Lightweight cryptographic applications have been frequently expressed by embedded systems designers, to implement a secured application such

as the authentication, the password storage mechanisms, the Digital Signal Standard (DSS), the Transport Layer Security (TLS), the Internet Protocol Security (IPSec), the Random number generation algorithms; etc. Several Lightweight cryptographic algorithms are presented. Lightweight cryptographic algorithms have been designed to fit with a very compact hardware. Each algorithm can be adapted for a specific field (Korb and Noll 2010; Bogdanov et al. 2013).

- SHA family: Secure SHA Algorithms are a family of Hash Algorithms published by NIST since 1993. SHA has many derivative standards such as SHA-0, SHA-1, SHA-3
- MDA/MD5/MD6: Message-Digest Algorithm is a family of broadly used cryptographic hash function developed by Ronald Rivest that produces a 128-bit for MD4 and MD5, 256-bit for MD6.
- Quark: Family of cryptographic functions designed for resource-constrained hardware environments.
- CubeHash: A very simple cryptographic hash function designed in University of Illinois at Chicago, Department of Computer Science.
- Photon: A lightweight hash function designed for very constrained devices.
- SQUASH: Not collision resistant, suitable for RFID applications.

According to its complexity, Quark presents the most appropriate algorithm to evaluate the performance of the soft-core FPGA processor architecture. Quark can minimize area and power consumption, it offers strong security guarantees. These Hash algorithms that are efficiently implemented in low cost embedded devices are important components for securing new applications in ubiquitous computing. Quark Hash algorithm is a family of lightweight cryptographic "sponge" algorithms designed for resource-constrained hardware environments, as RFID tags. It combines a number of innovations that make it unique and optimized. In the design of Quark, designers opt for an algorithm based on bit shift. It combines a sponge construction with a capacity e equal to the digest length n, and a core permutation inspired by preceding primitives. Quark algorithm proposes three instances: u-Quark, d-Quark and s-Quark. Quark is a family of cryptographic "sponge" functions intended for resource-constrained hardware environments (Bogdanov et al. 2013). It minimizes area and power consumption, yet offers strong security guarantees.Quark function includes four functions: (1) permute function, (2) init function, (3) update function and (4) final function. These instances are parameterized by a rate r, a capacity c, an output length n and a b-bit permutation (b = r + c). Table 5 demonstrates the parameters of each instance of the Quark algorithms.

Table 5 Parameters of Quark hash algorithms instance

	Rate (r)	Capacity (c)	With (b)	Digest (n)
u-Quark	8	128	136	136
d-Quark	16	160	176	176
s-Quark	32	224	256	256

6 Results

As mentioned before, the main topic of this study is to evaluate and validate the effect of the Xilinx MicroBlaze features and the proposed hardware/software partitioning approach on the embedded system performances.

6.1 Experimental Setup

6.1.1 Hardware Experimental Setup

Performance evaluation was estimated on a first time by a basic measurement of the different MicroBlazesoft-core configurations implemented on Xilinx Virtex-5 development board (XUPV5-LX110T, xc5vlx110t, grade ff1136, speed-1), illustrated in Figure 6.

Processor performance can be measured in different metrics such as execution time, energy consumption and area utilization. The most common metric is the time required for a processor to accomplish the defined task. In some architecture using an internal CPU clock driver, execution time presents the clock driver multiplied by the total instruction cycle count. In our case, execution time is measured using a Logic Analyser to have a high-precision measurement.

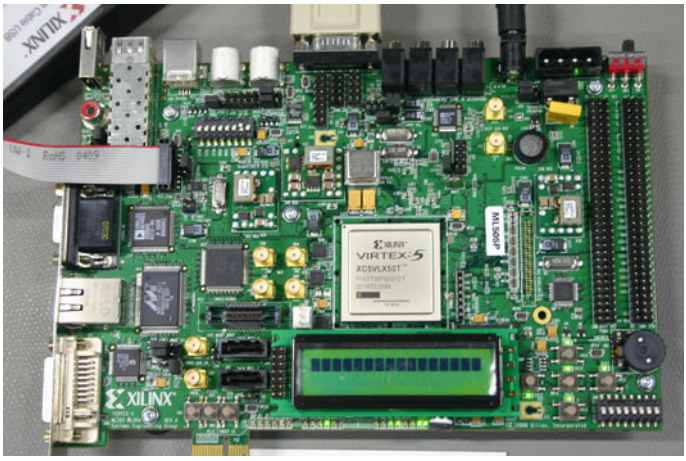

Fig. 6 Our plateform

6.1.2 Software Experimental Setup

In this chapter, we propose to immediately generate a cryptographic application as a co-processor (Hardware part) that will be added to a MicroBlaze using FSL interface (Software part). EDK tool will be used to perform the integration of both Hardware and Software in our architecture design. To implement Virtex-5 embedded applications, we use CoDeveloppper™ tool to generate co-processors or IPs (HLS methodology) and Xilinx Project Studio (XPS) to configure the FPGA including one MicroBlaze soft-core.

EDK Development Kit

The Xilinx EDK contains both an integrated development environment (IDE) named Xilinx Platform Studio (XPS) to create the Microprocessor Hardware Specification (MHS) file and the Software Design Kit (EDK) to create a Microprocessor Software Specification (MSS) file. The MHS file defines the embedded system processor, architecture and peripherals. The MSS file defines the library customization parameters for peripherals, the processor customization parameters, the standard I/O devices, the interrupt handler routines, etc.

CoDeveloper™

CoDeveloper™ is a commercialized by Impulse Accelerated Technologies in the CAD market. It allows designers to compile C applications directly into optimized logic ready for use with Xilinx FPGAs, in few times. ImpulseC code, the input language of CoDeveloper™, can be written and debugged in any ANSI standard C environment. The implemented algorithm can use both fixed and floating-point data point types. Impulse C is a library of functions and related data types that give a programming environment, and a programming model, for parallel applications targeting FPGA-based platforms. It has been optimized for mixed software/hardware targets, with the goal of abstracting details of inter-process communication and can allow relatively platform-independent application design. CoDeveloper™ includes the Impulse C libraries and associated software tools that help designers use standard C language for the design of highly parallel applications targeting FPGAs.

6.2 Application of the Proposed Design Approach for Quark Benchmark

6.2.1 Effect of MicroBlaze Soft-Core Configuration on Embedded Systems Performance

For embedded application, different MicroBlaze configurations can be provided. In real-time complex applications, both execution time and area consumption determine the efficiency and the high performance of the configured embedded soft-core processor.

The evaluation of hardware area presents one of the metric to select embedded configurations, which requires an optimal area. In a software design methodology, area consumption is independent from the implemented application. We can evaluate the performance of the soft-core processor for each configuration directly after the hardware specification step. Results prove that the average number of slices (a group of logic cell resources in FPGA) without using optimization option is very important. Table 6 depicts the area consumption recorded for some possible MicroBlaze configurations.

To evaluate the performance of the MicroBlaze soft-core processor, we have estimated the execution time in order to choose the most efficient configuration, which takes the minimum execution time onto the smaller hardware area. In our work, we compute the execution time of the configuration described in the table for three Quark hash functions (u-Quark function, d-Quark function and s-Quark function). Figure 7 illustrates the Quark hash functions execution time measurement for the 17 configurations of Xilinx MicroBlaze.

6.2.2 Automation of Partitioning Process

Designers have to specify the target architecture early in the design by defining the configuration of the software nodes to synthesize hardware nodes. Moreover, designers have, also, to determine the design constraints, performance constraints (timing) and resource constraints (area, memory). In this study, we choose to evaluate the proposed approach for a lightweight cryptographic s-Quark benchmark. We divide the C-high-level specification into four functional units (C functions) presented as nodes. We compute than nodes costs for all hardware and software possible architecture. For Software nodes, cost computation will be assured by profiling. For Hardware nodes, C functions will be transformed into Hardware specification using HLS approach, synthesized and analysed using Logical synthesis to get its costs. We propose MicroBlaze soft-core as software architecture with its different configurations (presented above). With our approach; we have to specify the costs (execution time and resources utilization) for each s-Quark node which can be implemented using soft-core (within all configurations).

Table 6 Area consumption of MicroBlaze processor synthesis

Configuration	With optimization synthesis		Without optimization synthesis	
MB + Units	LUTs	F-Fs	LUTs	F-Fs
1: Basic	1,210	1,452	1,657	1,693
2: BS	1,570	1,247	1,818	1,727
3: FPU	1,620	2,153	2,395	2,105
4: Mul	1,456	1,232	1,714	1,709
5: ID	1,581	1,326	1,801	1,805
6: MSRU	1,458	1,214	1,675	1,690
7: BS + mul + ID	1,727	1,380	1,964	1,867
8: BS + mul + FPU	2,307	1,674	2,511	2,162
9: BS + ID + FPU	2,433	1,769	2,668	2,258
10: BS + mul + MSRU	1,609	1,267	1,830	1,749
11: BS + ID + MSRU	1,734	1,365	1,966	1,846
12: BS + FPU + MSRU	2,313	1,659	2,533	2,142
13: mul + ID + FPU	2,358	1,755	2,575	2,241
14: mul + ID + MSRU	1,628	1,351	1,864	1,829
15: mul + FPU + MSRU	2,207	1,645	2,418	2,127
16: ID + MSRU + FPU	2,359	1,740	2,554	2,224
MSRU + FPU	2,202	1,625	2,417	2,107

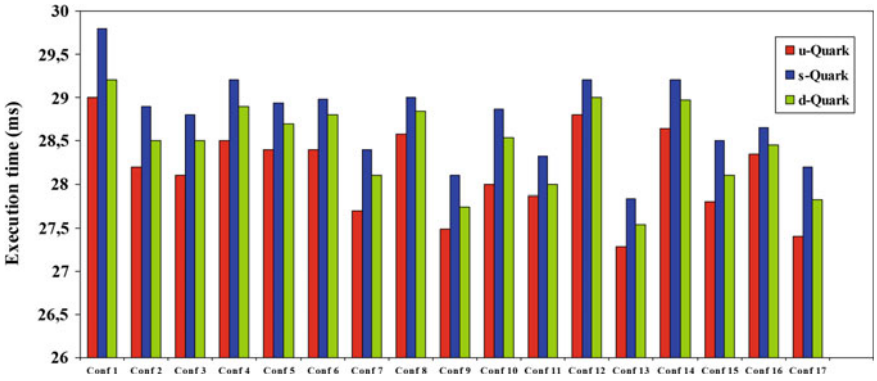

Fig. 7 Quark benchmark execution time usage for different MicroBlaze configurations

In order to select the greatest hardware/software architecture (partitioning process), we used the Integer Linear Programing (ILP) algorithm. Under the ILP algorithm, Gains of execution time and resource consumption are computed (as described in Table 7) using these two formulas:

Table 7 Temporal and Resources gain of s-Quark implementation

Task	1		2		3		4	
Gain	Gt	Gr	Gt	Gr	Gt	Gr	Gt	Gr
G1	20.94	68	0.84	10	2.91	14	6.98	145
G2	18.94	88	0.67	18	2.66	23	6.18	148
G3	19.14	109	0.72	10	2.71	21	5.78	178
G4	18.74	118	0.71	12	2.7	20	6.58	187
G5	19.94	128	0.64	14	2.63	17	5.32	198
G6	20.94	130	0.72	13	2.71	21	3.08	197
G7	19.94	132	0.82	17	2.91	19	4.18	197
G8	19.34	136	0.74	15	1.73	23	6.18	199
G9	20.14	139	0.82	17	2.81	22	4.08	201
G10	21.81	143	0.72	18	2.71	20	6.73	206
G11	18.26	146	0.71	20	2.70	22	6.18	205
G12	21.14	140	0.70	23	2.69	21	4.08	197
G13	20.74	143	0.72	14	2.71	23	6.18	207
G14	19.14	136	0.66	26	2.65	26	4.08	185
G15	18.14	148	0.64	30	2.63	27	6.73	192
G16	17.59	151	0.7	29	2.69	28	7.18	195
G17	18.14	155	0.62	26	2.7	24	6.48	206

(1) Gt = Execution time before Hw migration−Execution time after Hw migration
(2) Gr = Resources before Hw migration−Resources after Hw migration

In the designing of Quark cryptographic application, the designer has to satisfy temporal constraints while minimizing the number of the used resources. Partitioning process is based on the assignment of tasks on software and hardware units. This partitioning will be modified, with new hardware/software assignments, until the designer got the partition that meets the requirements of execution time and area consumption. The interesting parameter for partitioning is the number of nodes, which have to be partitioned. Using both hardware/software implementation, the time taken to transfer data between the soft-core and IPs (or co-processors) will be added. The cost of hardware/software communications are computed based on the width of transmitted data (8, 16 or 32 bits) and the rate of the communication buses.

As seen above, Xilinx MicroBlaze soft-core processor implements Harvard architecture. It means that it has separate bus interface for data and instruction access. The OPB interface gives a connexion to both on- and off-chip peripherals and memory. The MicroBlaze soft-core also provides 8 input and 8 output interfaces to Fast Simplex Link (FSL) buses. This FSL buses, 32 bits wide, are unidirectional non-arbitrated dedicated communication channels. In our study, we used the FSL interface due to its high performance (can reach up 300 Mb/S). EDK provides a set of Macros for reading and writing to or from FSL interface. Our

Table 8 S-Quark implementation results

	Execution time (ms)	Resources (Slices)	Design time : From architecture model to design implementation
Node 1 (Hw)	75	832	1 h
Node 2 (SW7)	32	774	5 min
Node 3 (SW5)	52	853	
Node 4 (SW1)	124	954	
Total Hw/Sw nodes	283	3413	1 h/15 min

purposed partitioning solution will determine the best partition that will reduce the number on nodes implemented on hardware and increase the number of nodes implemented on software to reduce the design time and the hardware area.

After hardware/software partitioning, we have to implement our s-Quark benchmark. Hardware nodes are implemented using HLS approach (CoDeveloper™ tool). Software nodes are executed using XPS tool. The integration of the hardware nodes (co-processors or IPs) with MicroBlaze soft-core processor is achieved using EDK tool. Table 8 illustrates results of s-Quark implementation.

7 Discussions

Increasing complexities of embedded systems application sunders core the need to take design decisions at an early stage. In our study, we are based on two important decisions related to the automation of the choice of both hardware/software partitioning and soft-core processor configurations.

7.1 Soft-Core Processor Configuration

MicroBlaze is a 32-bit embedded soft-core processor with a reduced instruction set computer (RISC) architecture. It is highly configurable and specifically optimized for synthesis into Xilinx field programmable gate arrays (FPGAs). The MicroBlaze soft-core processor is available as HDL source code or structural netlist. Itcan also be integrated into ASICs. As described in Fig. 6, one of the advantages of Xilinx MicroBlazesoft-core processors is its flexibility: it uses various configurations (more than 17 configurations) required for a specific application. Another advantage is its ability to integrate customized IP cores, which can result in a dramatic acceleration in software execution time (difference between configuration 1 and configuration 17) due to applications being executed in parallel with hardware and not sequentially in software.

Quark hash functions do not use huge values. They are dominated by barrel shifter, integer arithmetic, logic decisions, and memory accesses intended to reflect the CPU activities in computing applications. It takes a huge time for memory access. As described in the Fig. 6, selecting the best configuration enables a huge gain perspective of execution time and area consumption. The performance of implemented embedded systems using basic configuration (config. 1) is very low compared to the performance using Barrel Shifter Units (BS), Integer multiplier (Mul) and Floating-Point Units (FPU) configuration (config. 8). The execution time using the basic configuration takes 29 mS (for u-Quark); 29.8 mS (for s-Quark) and 29.2 mS (for d-Quark). 8) takes 28.58 mS (for u-Quark); 29 mS (for s-Quark) and 28.84 mS (for d-Quark). Area consumption constraint has also an effect on the embedded systems performance when modifying the configuration. Using basic configuration (config. 1), with optimization, takes 1,210 LUTs and 1,452 F-Fs. However, using Barrel Shifter Units (BS), Integer multiplier (Mul) and Floating-Point Units (FPU) configuration (config. 8) takes 2,307 LUTs and 1,674 F-Fs. If the application is area-critical, the user should select the best area/execution time constraints. In real-time embedded systems, area consumption constraint is not very important compared to the execution time.

Results prove that modifying configuration have an important effect on the embedded system performances. These results are interesting to make an optimized architecture for software design, designers of embedded systems can also benefit of FPGA hardware resources to more accelerate execution time and minimize the energy consumption. Hardware/software architecture has to be used to satisfy embedded systems constraints.

The results obtained from these different configurations require approximately 20 min per configuration, so, 60 % of the time is spent by the synthesis to choose the best configuration. Automate this step using time estimation approach allows the acceleration of the design time. Also, area synthesis results can be used on the designing of other embedded application, which reduce the design time.

7.2 Hardware/Software Partitioning

Partitioning an application among software solution on a soft-core processor (MicroBlaze) and hardware co-processors (IPs) in on-chip configurable logic has been shown to improve performance in embedded systems.

The used partitioning algorithm ILP is software oriented, because it starts with only software nodes. For this reason, the initial specifications were written in a high-level language (C functions). These functions are divided into functional units named nodes (node1, node2, node3 and node4 for Quark function). The first step in hardware/software partitioning step is the computation of both nodes and communication (between hardware and software nodes) costs. The costs can be defined as the execution time and the resources using hardware implementation (Hw1) or software implementation with different configuration of Microblaze (Sw1–Sw17).

Table 9 Benefits of our design approach compared to the existing ones

		Traditional design approaches	Recent design approaches	Our design approaches'
Design time	Time required for soft-core configuration	Time required for specifying all configurations + 20 mS for synthesizing each configuration	Time required for specifying all configurations + 20 mS for synthesizing each configuration	Time required for specifying all configurations + 20 mS for synthesizing each configuration (if node is not in the library)
	Time required for hardware/software architecture selection	Manually: long time: long decision time	Long time: specification has to be written in a low-level specification	Medium: specification is written in a high-level specification
	Time required for hardware/software partitioning	Long time: time required for the manually coding for both software and hardware architecture	Medium: time required for the automatic generation of codes for a specified architecture	Medium: time required for the automatic generation of codes for choosing architecture
Flexibility		Ye	No	Yes
Portability		Yes	No	Yes
Execution time		Very high	Medium	Medium
Area consumption		High	Reduced	Reduced

Choosing the greatest partitioning is verified by ILP algorithm. As result, we select to implement the node 1 (permute C function) as hardware. Permute function (node) is dominated by barrel shifter, integer arithmetic and logic decision. Implement it as a hardware node allows the designer to minimize area and execution time at least to 1.95 % for LUTs resources and 0.86 % for execution time comparing to software implementation. In addition, the integration of hardware nodes in soft-core MicroBlaze processor did not require to inline assembler code because the FSL interface has predefined C-macros that can be used for sending and receiving data between hardware and software nodes. Results of s-Quark benchmark (illustrated on the Table 8) prove that implementing complex applications on hardware/software architecture with automatic hardware/software partitioning are better than implementing these applications on software architectures (using MicroBlaze Soft-core processors). As summary, Table 9 illustrates features of our design approach compared to the existing ones.

8 Conclusions and Perspectives

FPGA presents an interesting circuit for implementing embedded applications. The purpose of this chapter to illustrate the impact of co-design approach, on the design acceleration and architecture performance. Based on the proposed co-design approaches of hardware/software partitioning, we are contributing to specification in order to increase its level. We, also, added a step to select the finest soft-core processor configuration in order to facilitate the co-design process, improve embedded systems' performance and reduce design time.

The presented results demonstrate that the choice of the good configuration has a significant impact on the system performance. The same approach can be used to evaluate the performance of other embedded systems or other architectures. Design methodologies of embedded systems, as mentioned in this paper, can be software, hardware or both software/hardware. Using co-design methodology helps the designer to obtain a good performance in a short time-to-market based on a good hardware/software partition. In this chapter, we have also introduced the hardware/software partitioning problem from a high-level specification. Several partitioning algorithms are presented in this study: One of them is based on ILP, which is used in our empirical tests. The ILP algorithm works efficiently for graphs with several hundreds of nodes and yield optimal solutions. As perspective, we can validate our proposed approach for more complexes embedded applications using FPGA devices for other vendors such as Altera, Actel, etc. We can also study the performances and design time benefits using time estimation approach instead of real performance evaluation.

References

Arulmozhiyal, R. (2012). Design and implementation of fuzzy PID controller for BLDC motor using FPGA. In *IEEE International Conference on Power Electronics*, *Drives and Energy Systems* (*PEDES*) (pp. 1–6), December 16–19, 2012. doi:10.1109/pedes.2012.6484251.

Bogdanov, A., Knezevic, M., Leander, G., Toz, D., Varici, K., & Verbauwhede, I. (2013). Spongent: The design space of lightweight cryptographic hashing. *IEEE Transactions on Computers, 62*(10), 2041–2053.

Bolado, M., Posadas, H., Castillo, J., Huerta, P., Sanchez, P., Sanchez, C., Fouren, H., & Blasco, F. (2004). Platform based on open-source cores for industrial applications. In *Europe Conference and Exhibition on Design*, *Automation and Test* (pp. 1014–1019), February 16–20, 2004. doi:10.1109/date.2004.1269026.

Boßung, W., Huss, S. A., & Klaus, S. (1999). High-level embedded system specifications based on process activation conditions. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology, 21*(3), 277–291.

Chatha, K. S., & Vemuri, R. (2000). An iterative algorithm for hardware-software partitioning, hardware design space exploration and scheduling. *Design Automation for Embedded Systems, 5*(3–4), 281–293.

Cherif, S., Quadri, I. R., Meftali, S., & Dekeyser, J. (2010). Modeling reconfigurable systems-on-chips with UML MARTE profile: An exploratory analysis. In *13th Euromicro Conference on Digital System Design: Architectures*, *Methods and Tools* (*DSD*) (pp. 706–713), September 1–3, 2010. doi:10.1109/dsd.2010.58.

Cloute, F., Contensou, J. N., Esteve, D., Pampagnin, P., Pons, P., & Favard, Y. (1999). Hardware/software co-design of an avionics communication protocol interface system: An industrial case study. In *7th International Workshop on Hardware/Software Codesign* (*CODES '99*) (pp. 48–52). doi:10.1109/hsc.1999.777390.

Dave, N., Ng, M. C., & Arvind. (2005). Automatic synthesis of cache-coherence protocol processors using Bluespec. In *3rd ACM and IEEE International Conference on Formal Methods and Models for Co-Design* (pp. 25–34), July 11–14, 2005. doi:10.1109/memcod.2005.1487887.

De Michell, G., & Gupta, R. K. (1997). Hardware/software co-design. *Proceedings of the IEEE, 85*(3), 349–365.

Denning, D., Irvine, J., Stark, D., & Delvin, M. (2004). Multi-user FPGA co-simulation over TCP/IP. In *15th IEEE International Workshop on Rapid System Prototyping* (pp. 151–156), June 28–30, 2004. doi:10.1109/iwrsp.2004.1311110.

Feng, W., Yuan, X., & Takach, A. (2009). Variation-aware resource sharing and binding in behavioral synthesis. In *Asia and South Pacific Design Automation Conference* (*ASP-DAC*) (pp. 79–84), January 19–22, 2009. doi:10.1109/aspdac.2009.4796445.

Fujita, M., & Nakamura, H. (2001). The standard SpecC language. In *Proceedings of the 14th International Symposium on Systems synthesis* (pp. 81–86).

Gruian, F., & Westmijze, M. (2008). VHDL vs. Bluespec system verilog: A case study on a java embedded architecture. In *Proceedings of the 2008 ACM Symposium on Applied Computing* (pp. 1492–1497).

Gupta, R. K., Coelho, C. N., & De Micheli, G. (1992). Synthesis and simulation of digital systems containing interacting hardware and software components. In *29th ACM/IEEE Design Automation Conference* (pp. 225–230), June 8–12, 1992. doi:10.1109/dac.1992.227832.

Henkel, J., & Ernst, R. (1998). High-level estimation techniques for usage in hardware/software co-design. In *Asia and South Pacific Design Automation Conference* (pp. 353–360), February 10–13, 1998. doi:10.1109/aspdac.1998.669500.

Ismail, T. B., Abid, M., O'brien, K., & Jerraya, A. (1994). An approach for hardware-software codesign. In *5th International Workshop on Rapid System Prototyping Shortening the Path from Specification to Prototype* (pp. 73–80), June 21–23, 1994. doi:10.1109/iwrsp.1994.315907.

Jianzhuang, W., Youping, C., Jingming, X., Bing, C., & Haiping, L. (2008). System structure for FPGA-based SOPC design using hard tasks. In *6th IEEE International Conference on Industrial Informatics*, *INDIN 2008* (pp. 1154–1159), July 13–16, 2008. doi:10.1109/indin.2008.4618277.

Jing-Jie, W., & Rui, H. (2011). A FPGA-based wireless security system. In *Third International Conference on Multimedia Information Networking and Security* (*MINES*) (pp. 512–515), November 4–6, 2011. doi:10.1109/mines.2011.82.

Joven, J., Strict, P., Castells-Rufas, D., Bagdia, A., De Micheli, G., & Carrabina, J. (2011). HW-SW implementation of a decoupled FPU for arm-based cortex-M1 SOCS in FPGAS. In *6th IEEE International Symposium on Industrial Embedded Systems* (*SIES*) (pp. 1–8), June 15–17, 2011. doi:10.1109/sies.2011.5953649.

Kalavade, A., & Lee, E. A. (1994). A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem. In *3rd International Workshop on Hardware/Software Codesign* (pp. 42–48), September 22–24, 1994. doi:10.1109/hsc.1994.336724.

Kalomiros, J. A., & Lygouras, J. (2008). Design and evaluation of a hardware/software FPGA-based system for fast image processing. *Microprocessors and Microsystems, 32*(2), 95–106.

Kikuchi, H., & Morioka, K. (2012). Development of wireless image sensor nodes based on FPGA for human tracking in intelligent space. In *IECON 2012—38th Annual Conference on IEEE Industrial Electronics Society* (pp. 5529–5534), October 25–28, 2012. doi:10.1109/iecon.2012.6388950.

Korb, M., & Noll, T. G. (2010). LDPC decoder area, timing, and energy models for early quantitative hardware cost estimates. In *International Symposium on System on Chip* (*SoC*) (pp. 169–172), September 29–30, 2010. doi:10.1109/issoc.2010.5625546.

Ku, D. C., & De Mitcheli, G. (1992). Relative scheduling under timing constraints: Algorithms for high-level synthesis of digital circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 11*(6), 696–718.

Lach, J., Mangione-Smith, W. H., & Potkonjak, M. (1999). Robust FPGA intellectual property protection through multiple small watermarks. In *Proceedings 36th Design Automation Conference* (pp. 831–836), 1999.

Li, Y.-T. S., & Malik, S. (1995). Performance analysis of embedded software using implicit path enumeration. *ACM SIGPLAN Notices, 30*(11), 88–98.

Lingbo, Z., Fuchun, S., & Zengqi, S. (2006). Cloud model-based controller design for flexible-link manipulators. In *IEEE Conference on Robotics*, *Automation and Mechatronics* (pp. 1–5), December 2006. doi:10.1109/ramech.2006.252742.

López-Vallejo, M., & López, J. C. (2003). On the hardware-software partitioning problem: System modeling and partitioning techniques. *ACM Transactions on Design Automation of Electronic Systems (TODAES), 8*(3), 269–297.

Lysecky, R., & Vahid, F. (2004). A configurable logic architecture for dynamic hardware/software partitioning. In *Design*, *Automation and Test in Europe Conference and Exhibition* (pp. 480-485), February 16–20, 2004. doi:10.1109/date.2004.1268892.

Madsen, J., Grode, J., Knudsen, P. V., Petersen, M. E., & Haxthausen, A. (1997). LYCOS: The Lyngby co-synthesis system. *Design Automation for Embedded Systems, 2*(2), 195–235.

Mcloone, M., & Mccanny, J. V. (2003). Generic architecture and semiconductor intellectual property cores for advanced encryption standard cryptography. *IEE Proceedings on Computers and Digital Techniques, 150*(4), 239–244.

Monmasson, E., & Cirstea, M. N. (2007). FPGA design methodology for industrial control systems—A review. *IEEE Transactions on Industrial Electronics, 54*(4), 1824–1842.

Mysore, N., Akcakaya, M., Bajcsy, J., & Kobayashi, H. (2005). A new performance evaluation technique for iteratively decoded magnetic recording systems. In *Digests of the IEEE International Magnetics Conference* (*INTERMAG*) (pp. 1603–1604), April 4–8, 2005. doi:10.1109/intmag.2005.1464235.

Nasreddine, N., Boizard, J. L., Escriba, C., & Fourniols, J. Y. (2010). Wireless sensors networks emulator implemented on a FPGA. In *International Conference on Field-Programmable Technology* (*FPT*) (pp. 279–282), December 8–10, 2010. doi:10.1109/fpt.2010.5681484.

Noonburg, D. B., & Shen, J. P. (1997). A framework for statistical modeling of superscalar processor performance. In 3rd *International Symposium on High-Performance Computer Architecture* (pp. 298–309), February 1–5, 1997. doi:10.1109/hpca.1997.569691.

Reynari, L. M., Cucinotta, F., Serra, A., & Lavagno, L. (2001). A hardware/software co-design flow and IP library based of simulink. In *Design Automation Conference* (pp. 593–598), 2001. doi:10.1109/dac.2001.156209.

Samarawickrama, M., Rodrigo, R., & Pasqual, A. (2010). HLS approach in designing FPGA-based custom coprocessor for image preprocessing. In *5th International Conference on Information and Automation for Sustainability* (*ICIAFs*) (pp. 167–171), December 17–19, 2010. doi:10.1109/iciafs.2010.5715654.

Sorin, D. J., Pai, V. S., Adve, S. V., Vemon, M. K., & Wood, D. A. (1998). Analytic evaluation of shared-memory systems with ILP processors. In *The 25th Annual International Symposium on Computer Architecture* (pp. 380–391), June 27–July 1, 1998. doi:10.1109/isca.1998.694797.

Stitt, G., Lysecky, R., & Vahid, F. (2003). Dynamic hardware/software partitioning: A first approach. In *Proceedings of the 40th Annual Design Automation Conference* (pp. 250–255).

Stoy, E., & Zebo, P. (1994). A design representation for hardware/software co-synthesis. In *The 20th EUROMICRO Conference on System Architecture and Integration* (pp. 192–199), September 5–8, 1994. doi:10.1109/eurmic.1994.390391.

Talpin, J., Le Guernic, P., Shukla, S. K., Gupta, R., & Doucet, F. (2003). Polychrony for formal refinement-checking in a system-level design methodology. In *3rd International Conference on Application of Concurrency to System Design* (pp. 9–19), June 18–20, 2003. doi:10.1109/csd.2003.1207695.

Thangavelu, A., Varghese, M. V., & Vaidyan, M. V. (2012). Novel FPGA based controller design platform for DC–DC buck converter using HDL co-simulator and Xilinx system generator. In *IEEE Symposium on Industrial Electronics and Applications* (*ISIEA*) (pp. 270–274), September 23–26, 2012. doi:10.1109/isiea.2012.6496642.

Wakabayashi, K., & Okamoto, T. (2000). C-based SoC design flow and EDA tools: An ASIC and system vendor perspective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 19*(12), 1507–1522.

Washington, C., & Dolman, J. (2010). Creating next generation HIL simulators with FPGA technology. In *IEEE AUTOTESTCON* (pp. 1–6), September 13–16, 2010. doi:10.1109/autest.2010.5613618.

Wiangtong, T., Cheung, P. Y., & Luk, W. (2005). Hardware/software code sign: A systematic approach targeting data-intensive applications. *IEEE Signal Processing Magazine, 22*(3), 14–22.

Wolf W. (2003). A decade of hardware/software codesign. *Computer*, *36*(4), 38–43.

Xiaoyin, S., & Dong, S. (2007). Development of a new robot controller architecture with FPGA-based IC design for improved high-speed performance. *Industrial Informatics, IEEE Transactions on, 3*(4), 312–321.