HAPE: A high-level area-power estimation framework for FPGA-based accelerators

Mariem Makni, Smail Niar, Mouna Baklouti, Mohamed Abid

Microprocessors and Microsystems 63 (2018) 11–27

HAPE: A high-level area-power estimation framework for FPGA-based accelerators

Mariem Makni*,[a], Smail Niar[a], Mouna Baklouti[b], Mohamed Abid[b]

[a] LAMIH, University of Valenciennes, France
[b] National Engineering School of Sfax, Tunisia

Keywords:
HLS
SoC
Hardware accelerator
RTL
FPGA
DSE

ABSTRACT

Recent embedded applications are widely used in several industrial domains such as automotive and multimedia systems. These applications are critical and complex, involving more computing resources and therefore increasing the power consumption of the system. Although performance still remains an important design metric, power consumption has become a critical factor for several systems, particularly after the increasing complexity of recent System-on-Chip (SoC) designs. Consequently, the whole computing domain is being forced to switch from a focus on high performance computation to energy-efficient computation. In addition to the time-to-market challenge, designers need to estimate, rapidly and accurately, both area occupation and power consumption of complex and diverse applications. High-Level Synthesis (HLS) has been emerged as an attractive solution for designers to address this challenge in order to explore a large number of design points at a high-level of abstraction. In this paper, we target FPGA-based accelerators. We propose HAPE, a high-level framework based on analytic models for area and power estimation without requiring register-transfer level (RTL) implementations. This technique allows to estimate the required FPGA resources and the power consumption at the source code level. The proposed models also enable a fast design space exploration (DSE) with different trade-offs through HLS optimization pragmas, including loop unrolling, pipelining, array partitioning, etc. The accuracy of our proposed models is evaluated by using a variety of synthetic benchmarks. Estimated power results are compared to real board measurements. The area and power estimation results are less than 5% of error compared to RTL implementations.

1. Introduction

Embedded System-on-Chips (SoCs) have often conflicting constraints such as time and energy which considerably harden the design of those systems. In addition, complex embedded applications have to cope with an increasing demand of functionalities, which require increasing processing capabilities [1,2].

With the introduction of heterogeneous computing systems such as the Xilinx Zynq UltraScale+ multiprocessor system-on-chip (MPSoC) [3], different processing units can be embedded in the SoC to meet the growing requirements of the applications (performance/power constraints). Complex applications include different computing-intensive functions with multiple nested loops. This leads to significantly increased power consumption as well as higher processing requirements to ensure the respect of constraints expected in such systems. Consequently, designers need to estimate the various design metrics (execution time, area, power) of the embedded system at the earliest step in the design flow.

High-Level Synthesis (HLS) [4,5] tools have been developed in the recent years to address this challenge. These tools are used to automatically generate circuit specifications in hardware description language from high-level languages (e.g., C/C++) without the need for time-consuming manual register-transfer level (RTL) generation [6]. The utilization of these tools significantly saves time and programming effort.

In addition, HLS tools provide various optimization pragmas such as loop unrolling, pipelining, array partitioning, etc. [7,8]. This enables designer to explore the large number of potential design points for an application with these pragmas while optimizing for performance and/or area constraints. Unfortunately, the large design space resulting from the different pragma combinations makes exhaustive design space exploration (DSE) a time-consuming task. Consequently, the runtime of

DOI : 10.1016/j.micpro.2018.08.004

the HLS tools is prohibitively long to exclude the possibility of exhaustive design space, especially for complex designs.

Analytic models have been proposed to address these challenges by enabling a rapid design space exploration for Field-Programmable Gate Array (FPGA) based accelerators at a high-level of abstraction. By designing at a higher level of abstraction, the designer can work more productively and achieve faster time-to-market than using manual RTL designs.

In addition to performance estimation, it is critical for designers to evaluate whether their implementation meets the area requirements on an FPGA platform. Hence, there is a clear need for an area analytic model that allows to quantitatively estimate the FPGA resources (LUTs, BRAMs, etc.) required to map a given application.

In this paper, we target FPGA-based accelerators. These circuits have recently gained popularity for systems that demand the programmability and customization offered by the reconfigurable FPGA fabric. They are considered as a promising alternative due to the energy efficiency compared to other architectures, such as NVIDIA Tegra [9] and Kalray MPPA [10]. In this work, we have selected the Xilinx ZC702 (ZYNQ) [11], which is an FPGA-based Processor/Accelerator system. FPGAs have the benefits of being high speed and adaptable to the application constraints at a reduced performance per watt if compared to the General Purpose Processors (GPP). Since FPGA will be in charge of executing a large portion of the system power consumption, an accurate power estimation model for FPGA circuits is necessary.

This work has three-fold contributions:

- Modeling accurately hardware resource utilization (LUTs, FFs, etc.) of different applications mapped on FPGA-based accelerators (Section 3.4).
- Modeling accurately power consumption of FPGA-based accelerators (Section 3.5).
- Estimating the impact of optimization pragmas on both the area occupation and the power consumption for FPGA-based accelerators under FPGA resource constraints (Section 3).

In [12], we proposed a high-level area estimation tool based on an analytic model without requiring RTL implementations. Compared to the work we presented in [12], this paper is different in the following points:

- We give an overview of a wide selection of HLS tools that are currently available.
- We elaborate on all aspects of the proposed framework, including an analytic power consumption estimation model.
- We describe the target hardware architecture and the FPGA-based accelerator interconnection approach in further detail.
- We provide additional information on the different benchmarks and optimization pragmas used in this work.

In this paper, we introduce **HAPE**, a **H**igh-level **A**rea and **P**ower **E**stimation framework for FPGA-based accelerators. We use the Advanced eXtensible Interface (AXI4) stream [13] to communicate between the main processor and the Hardware Accelerators (HAs) for diverse applications. The area and power estimation results are less than 5% of error compared to RTL implementations. In addition, our proposed framework provides these estimates faster than existing HLS tools, such as Xilinx Vivado HLS [8] and without invoking HLS.

The remainder of this paper is organized as follows: Section 2 presents background and related work. Section 3 introduces our proposed framework and the target hardware architecture. An experimental evaluation appears in Section 4. Conclusions and suggestions for future work are given in Section 5.

2. Background and related work

The complexity of SoC designs has significantly increased in recent years. Furthermore, streaming embedded applications are widely used in several industrial domains such as automotive, multimedia and surveillance systems. Several Multiprocessor System-on-Chip (MPSoC) designs, integrating multiple cores or processors on a single die [14], have been proposed to cope with the application requirements. As an example of famous commercial platforms based on such architecture, we quote the NVIDIA Tegra [15] processor which integrates a quad-core ARM Cortex A15. Unfortunately, such architectures present larger area and higher power consumption because no single type of processor can be well suited to every application; hence, they are more suitable for general-purpose systems rather than embedded systems, which require more performance and energy efficiency.

FPGA-based Processor/Accelerator systems including Xilinx Zynq-7000 All Programmable SoC [11], have emerged in parallel as a privileged target platform to implement intensive processing applications. In fact, they have the benefits of being high speed and adaptable to the application constraints at a high performance per watt ratio.

In the following subsections, we give some background information related to the different HLS pragmas used in this work as well as an overview of some existing works and HLS tools.

2.1. HLS optimization pragmas

HLS tools provide various optimization pragmas such as loop unrolling, loop pipelining and array partitioning. The designer is responsible for exploring the large number of potential design choices available for an application with these pragmas while optimizing for performance and/or area constraints. These pragmas have a great impact not only on performance but also on resource utilization and power consumption. Applying multiple pragmas produces various implementations with different performance/energy trade-offs. Hence, our proposed framework supports these pragmas while enabling a fast architectural exploration to identify the best design for a given application. The optimization pragmas, considered in this work, are detailed as follows [4]:

(A) Loop pipeliningis an optimization pragma applied at the loop level, allowing parallel executions of loop iterations. When enabled, the hardware performance is determined by a constant Initiation Interval (II) of the loop.
II is defined as the number of clock cycles between the start times of consecutive loop iterations [16]. This pragma provides higher throughput with less execution time. Fig. 1 illustrates the basic concept of loop pipelining. In sequential languages such as C/C++, the operations in a loop are executed sequentially and the next

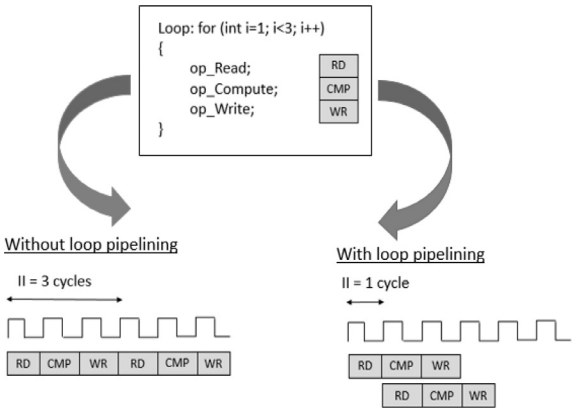


Fig. 1. Loop pipelining pragma.

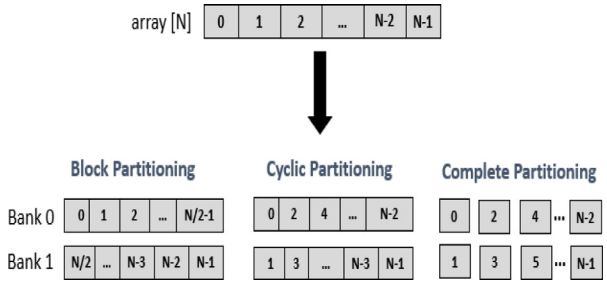


Fig. 2. Array partitioning pragma for the three strategies with partitioning factor of 2 [4].

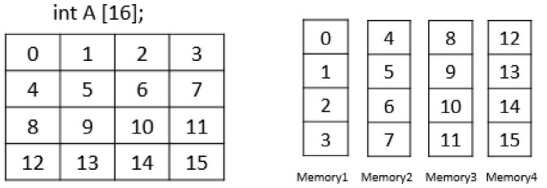
iteration of the loop can only begin when the last operation in the current loop iteration is complete. As shown in Fig. 1, without pipelining, there are three clock cycles between the two read operations and it requires six clock cycles for the entire loop to finish. However, with pipelining, there is only one clock cycle between the two read operations and it requires four clock cycles for the entire loop to finish.

(B) Array partitioningallows to split program arrays into multiple smaller arrays stored in multiple memory banks in the Block RAMs (BRAMs). Without loss of generality, we assume that each memory partition has two read and one write ports. In Xilinx Vivado HLS, the array partitioning strategies include three types, *block, cyclic* and *complete*, as shown in Fig. 2.

Fig. 3 shows an example of array partitioning pragma. In this example, the original array *A* is partitioned into multiple memory banks with block partitioning and partitioning factor of 4. As each memory bank has two read ports, memory load operations for array *A* can be executed in the same cycle. As depicted in Fig. 3, array partitioning has the advantage of improving the memory bandwidth by increasing the number of load/store ports.

(C) Loop unrollingis another technique to exploit parallelism between loop iterations. It can reduce the loop overhead by reducing the number of iterations and replicating the body of the loop. With loop unrolling, we can transform an M-iteration loop into a loop with M/N iterations. Fig. 4 shows an example of loop unrolling pragma. In this example, the loop is unrolled by a factor of 2 to have N/2 iterations. The loop unrolling can also remove the dependences between loop index variables by completely unrolling loops to execute several iterations in parallel. However, for large loop bounds, loop unrolling leads to high FPGA resource requirements as well as high power consumption.

Designers can use HLS tools to explore diverse hardware implementations by inserting different optimization pragmas. Loop pipelining, unrolling and array partitioning are the most prominent pragmas in modern HLS tools, such as Xilinx Vivado HLS [8]. When enabled, these pragmas have the advantage of improving performance in FPGA-



(a) without array partitioning: reading all values of array A can take 8 clock cycles

(b) with array partitioning: reading all values of array A can take only 2 clock cycles

Fig. 3. Array partitioning example for block partitioning strategy with factor of 4).

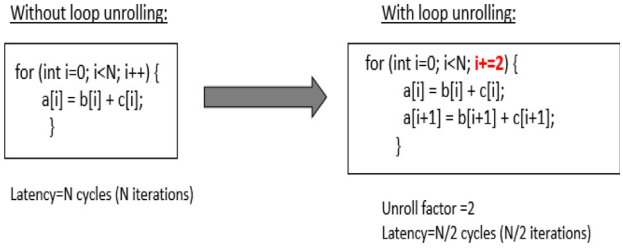


Fig. 4. Loop unrolling pragma.

based accelerator systems. However, more logic resources are required, as will be demonstrated by experiments, presented in Section 4.

2.2. An overview of High-Level Synthesis (HLS) tools

Hardware designers require writing complex RTL code to generate implementations for mapping the applications on heterogeneous computing systems featuring FPGAs. This process is error prone and can be difficult to debug. HLS tools [6,17,18], on the other hand, are more straightforward, simple programming and are easily accessible.

Several HLS tools have been developed for targeting specific applications. LegUp [6] is an open source HLS tool that compiles automatically a C program to target a hybrid FPGA-based hardware/software system. It can synthesize a design in C language to a custom hardware design. However, LegUp relies on standard commercial HLS tools (e.g., eXCite[19], Altera's PowerPlay power analyzer tool [20]) to measure speed, area and energy of the generated RTL implementations. This can be costly and difficult, making large design space a highly time-consuming process for designers.

ROCCC [18] is an open source HLS tool that can generate custom HAs from C programs. ROCCC is designed to accelerate critical kernels that perform repeated computation on streams of data, such as FIR filters. However, ROCCC does not support several commonly-used aspects of the C language, such as generic pointers, shifting by a variable amount and non-for loops, and the ternary operator. Therefore, ROCCC's strict subset of C is insufficient for compiling any of the CHStone and Polybench benchmarks used in this study and described in Section 4. In addition, it does not support advanced optimization pragmas such as array partitioning, loop pipelining, etc. Therefore, their work has limited design space.

On the commercial front, there is Altera's C2H tool [21]. This tool allows designers to partition a high-level source code (C program'functions) into custom HAs. The software segments are executed on a Nios II soft processor. The C2H system architecture is similar to that targeted by Canis et al. [6]. eXCite tool [19] is another commercial HLS tool. It compiles a standard C program to a hybrid processor/accelerator architecture. Catapult [22] is a HLS tool acquired from Mentor Graphics. This tool accepts a large subset of C, C++ and SystemC, targeting ASICs and FPGAs. In contrast to [19,21], the synthesized RTL generated by Catapult, is optimized for power, area, and speed. Loops can be unrolled completely or partially, or they can be pipelined with a certain initiation interval. However, memory accesses are not optimized by the tool, array elements that are reused in subsequent iterations are fetched from memory on every use.

GAUT [5] is a HLS tool that is designed for DSP applications. GAUT accepts a C program as an input to be synthesized into an architecture with a processing unit, a memory unit, and a communication unit, and requires that the user supply specific constraints, such as the pipeline initiation interval. eXCite and GAUT tools have not been under active development for several years and are no longer maintained.

With regard to commercial tools, there has been considerable activity in recent years, both in start-ups and major EDA vendors. Vivado HLS is a commercial HLS tool, released by Xilinx [8]. This tool starts from a high-level programming language (e.g., C/C++) to

automatically generate a circuit specification in hardware description language that performs the same function. It can also optimize the area, speed and power consumption of the hardware implementation.

Another commercial HLS tool is Altera SDK for Open Computing Language (OpenCL). At a high level, Altera SDK [30,31] translates an OpenCL kernel to a hardware circuit that executes on the FPGA. Similar to Xilinx Vivado HLS tool [8], recent Altera FPGA devices support optimization pragmas for different application domains, such as loop unrolling, pipelining, etc.

Since the proposed work deals with area/power estimation of FPGA-based accelerators during high level synthesis, hence this paper will focus only on related approaches at higher abstraction levels and not lower levels.

2.3. An overview of high-level performance/area/power estimation tools

Several works have investigated the high-level estimation aspect for different FPGA-based accelerators but mostly from a performance perspective [28,32,33]. In [29], the authors propose a high-level estimation model to estimate the power consumption of FPGA-based systems. Their estimation model is only based on Logic Slices and Block RAMs (BRAMs) parameters. They use a test application to validate the proposed model, which is a 720p video frame standard. However, this work has limited design space since it only focuses on one application, which is not sufficient to validate an analytic model. Our approach on the contrary uses different sets of applications to validate the proposed estimation models. Moreover, in [29], the framework does not support any optimization pragma to improve the performance of their system. Therefore, their proposed power estimation model is not validated for our work.

Most of the high-level estimation models are aimed towards specific entities like IP Cores [34] or arithmetic operators [35] or softcore processors [36] etc. and are not generic for FPGA-based architectures. The model presented in [37] only works on streaming functions (video streaming applications), and cannot be easily expanded to large design space with different application domains, such as signal processing applications, etc.

The authors of Shao et al. [17] propose *Aladdin*, a pre-RTL, power-performance accelerator modeling framework. Aladdin allows for several design space exploration options such as loop unrolling, pipelining and array partitioning. It estimates performance, power, and area of accelerators within 0.9%, 4.9%, and 6.6% with respect to RTL implementations. However, Shao et al. [17] is mainly oriented towards Application Specific Integrated Circuit (ASIC) accelerators. This limits the target hardware architectures to ASIC-based accelerators.

Unlike the aforementioned works, our proposed pre-RTL models do not rely on any commercial HLS tools. Furthermore, we use different sets of applications (data mining, signal processing, image processing, etc.) to validate our proposed framework.

In [28], the authors perform more extensive design space through several HLS optimization pragmas, such as loop pipelining, unrolling, etc. The Lin-Analyzer tool [28] is primarily focused on streaming applications and implementation of these applications on FPGA-based accelerators.

The majority of the current state-of-the-art methods [17,28], propose high-level estimation models to estimate the computation cost and ignore the data communication cost between the different system components. Moreover, Makni and co-workers [16,23,26,27] consider only loop unrolling pragma and ignore the other two prominent pragmas (loop pipelining and array partitioning) that have significant impact on system performance and power consumption.

Existing works [23,24,28] that present analytic area estimation models for FPGA-based systems often ignore the hardware resource constraints of the target FPGA platform in their models. In this work, we estimate the area occupation for FPGA-based accelerator systems under FPGA resource constraints.

Canis and co-workers [6,27,38] use commercial HLS tools to perform area, performance and power estimations. However the usage of commercial HLS tools in their frameworks significantly increases the exploration time to hours or even days in some cases. Schafer et al. [38] propose a divide and conquer algorithm for solving HLS design space exploration problems. They first parse kernels into a set of clusters which consist of loops, functions and arrays. Then they exhaustively search each cluster by invoking HLS tools with all possible configurations to find the local Pareto-optimal points. Finally, they combine the local Pareto-optimal configurations and invoke HLS tools again to find the global Pareto-optimal points. Performing full synthesis at each design iteration can become quite time-consuming. Therefore, their method suffers from long simulation/synthesis runtime.

Several authors [23–27,38] rely on a static program analysis to estimate the performance of SoC designs while exploring the massive design space. However, the static analysis causes false dependences between the operations and therefore introduces large inaccuracies in the estimated performance due to the lack of memory information during the static analysis. In contrast, our proposed methodology is based on a dynamic analysis, which is built with runtime information to avoid false data dependences that restrict algorithmic parallelism.

In addition, our proposed Pre-RTL framework can estimate resource utilization and power for larger SoC configurations and more general C/C++ source code than those supported by Shao and co-workers [17,28]. Moreover, it supports several options for design optimization, such as array partitioning, loop unrolling and pipelining to explore large design space. In this work, we use Advanced eXtensible Interface (AXI4) stream interface [13,39] to communicate between the processor and the custom HAs. Based on several experimental results [16], we found that the suitable interconnect solution for the recent streaming applications is the AXI4 stream protocol. AXI4 is the latest revision of the Advanced Microcontroller Bus Architecture (AMBA) 4.0 standard [39]. Table 1 summarizes the different HLS tools surveyed above. We compare various HLS tools based on different criteria. In this paper, we

Table 1
Current state-of-the-art techniques vs. proposed approach.

	Analysis (Static/ dynamic)	Accuracy	Pragma exploration	Target	Model outputs
Bilavarn [23] [TCAD'06]	Static	Medium	Loop Unrolling	FPGA	Computation cost Area
Smith [24] [FPL'09]	Static	Medium	N/A	FPGA	Area
Villarrea [18] [FCCM'10]	Static	Medium	N/A	FPGA	HLS tool
Canis [6] [FPGA'11]	Static+ HLS	High	Loop Pipelining	FPGA	HLS tool
Liu [25] [DAC'13]	Static + HLS	High	Loop Unrolling + Loop Pipelining + Array Partitioning	ASIC	Computation cost Area
Boucle [26] [DSD'13]	Static	Medium	Loop Unrolling+ Array to registers	FPGA	Computation cost
Shao [17] [ISCA'14]	Dynamic	High	Loop Unrolling+ Loop Pipelining+ ArrayPartitioning	ASIC	Performance Area Power
Zhong [27] [ICCD'14]	Static + HLS	High	Dataflow+ Loop unrolling	FPGA	Computation cost Area
Zhong [28] [DAC'16]	Dynamic	High	Loop Unrolling + Loop Pipelining + Array Partitioning	FPGA	Computation cost
Makni [16] [PDP'17]	Dynamic	High	Dataflow+ Loop Pipelining	FPGA	Data communication cost
Sharma[29] [ARC'17]	Static	High	N/A	FPGA	Power
Our proposed framework	Dynamic	High	Loop Unrolling + Loop Pipelining + Array Partitioning	FPGA	Data communication cost Area Power

propose a high-level framework based on different analytic models to rapidly estimate area occupation and power consumption for a large set of SoC configurations. Our goal is to assist the design space exploration to reduce the number of invocations of HLS tools. In addition, the proposed framework supports different optimization pragmas with multi-level parallelism on FPGAs. We propose a dynamic analysis method that exploits runtime information to obtain true dependences between operations and therefore accurately estimates area occupation and power consumption. This also obviates the need to use HLS tools, resulting in a rapid and reliable design space framework.

3. Proposed approach: a high-level area-power estimation framework

In this section, we describe our proposed high-level estimation framework. We also give an overview of the HLS flow and the Low-Level Virtual Machine (LLVM) framework.

3.1. High-Level Synthesis: HLS

Several advantages arise from the use of HLS in the design flow. First of all, the amount of code to be written by designers is reduced dramatically, which saves time, programming effort and reduces the risk of mistakes. HLS can also optimize a design by applying several optimization pragmas to increase the performance of the system, resulting an extensive design space. This is particularly relevant for the design of complex FPGA-based systems.

As shown in Fig. 5a, HLS creates an RTL implementation from C/C++ level source code. HLS has traditionally been divided into three important steps [4]: *Allocation, Scheduling* and *Binding*. Fig. 5b illustrates the different steps of the HLS flow.

3.1.1. Allocation

After analysis of the source code, *allocation* determines the types of operators and the amount of hardware resources available for use (e.g., the number of multipliers, adders, etc.). This step also manages other hardware constraints (e.g., time, area, and power) at a high-level of abstraction.

3.1.2. Scheduling

As shown in Fig. 6, *scheduling* determines in which clock cycles an operation will occur. It takes into account the different hardware resources extracted from C/C++ source code at the top level. The allocation of resources can be constrained to ensure that the design does not exceed FPGA's capacity.

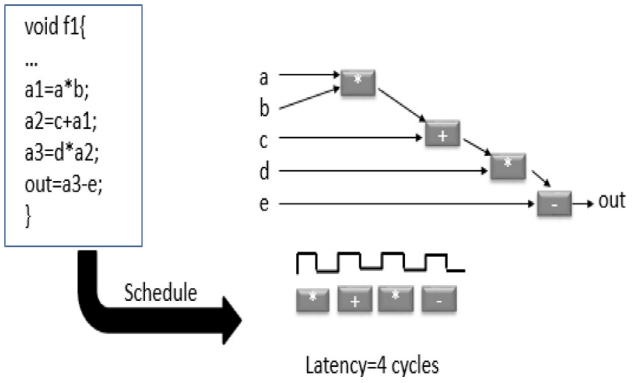


Fig. 6. Scheduling.

3.1.3. Binding

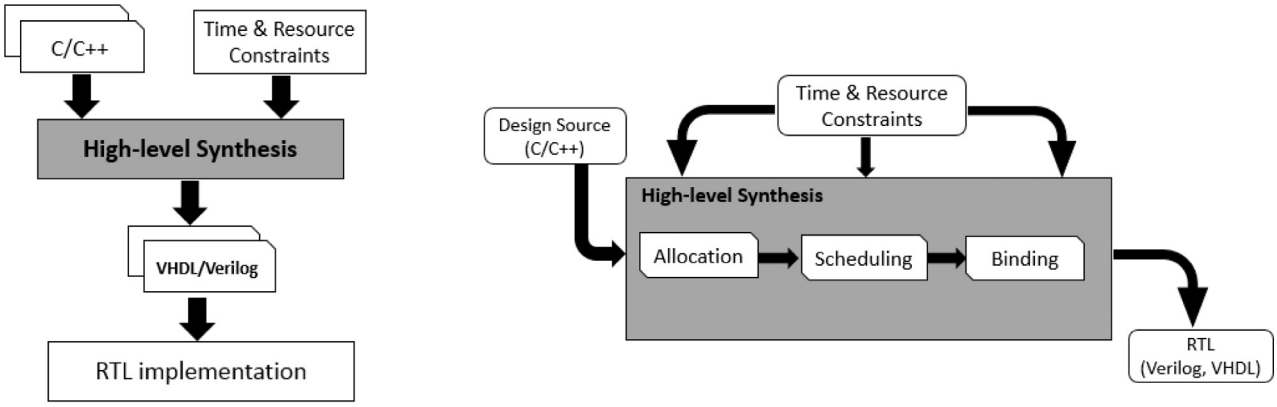
Binding determines which hardware unit is used to implement each operation. It saves area by sharing functional units between operations and sharing registers/memories between variables. Given for example the schedule presented in Fig. 6, there are two binding decisions: (a) binding may decide to share the multipliers since both are in a different cycle, (b) binding may decide to use two different multipliers because the cost of sharing (muxing) would impact timing. Based on user constraints e.g., for latency and FPGA resource usage, scheduling and binding are determined.

3.2. The proposed framework

3.2.1. Modeling methodology

The proposed framework, shown in Fig. 8a, takes a high-level specification (C/C++) of an algorithm in the form of nested loops, and FPGA resource constraints as inputs. It automatically estimates the total execution time, the FPGA resource utilization and the power consumption for a hybrid architecture containing an FPGA-based accelerator. An overarching goal of the proposed framework is to provide area and energy benefits of a hardware design, while retaining the ease-of-use associated with software. Figs. 8a and b illustrate the detailed flow. The proposed framework primarily consists of four important steps: computation cost estimation, data communication cost estimation, area occupation estimation and power consumption estimation. Computation cost estimation has been implemented using Lin-Analyzer [28], while data communication cost estimation has been implemented using an analytic model detailed in our previous work in [16].

Integrating the proposed power model with the existing performance and area models allows designers to analyze and evaluate the



(a) High-Level Synthesis Flow

(b) High-Level Synthesis: Allocation, Scheduling and Binding

Fig. 5. High-Level Synthesis (HLS).

performance/power trade-off for complex applications within the strict time-to-market and non-recurring engineering constraints. Furthermore, our proposed framework allows designers to select the most efficient pragmas in their SoC designs. This gives the designers multiple implementation choices (e.g., loop unrolling factors, array partitioning factors) to improve system performance.

The foundation of the proposed framework infrastructure is the use of DDDG graph generated from a dynamic execution trace to represent program behaviors. This technique avoids the false data dependences created by the static analysis used in most existing HLS tools. Besides, the features of the dynamic trace coupled with the dataflow nature of accelerators makes DDDG a good candidate for modeling hardware behavior. Fig. 8a illustrates the overall structure of our framework, starting from an unmodified C/C++ description of an application and passing through an *optimization* phase, described in Section 3.2.2, where the sub-trace is extracted and constructed. The sub-trace then passes to a *generation* phase, discussed in Section 3.2.3, where the DDDG is constructed and optimized to derive a realistic and accurate representation of the application. The outcome of these two phases is a pre-RTL, performance-area-power estimation of HAs across a wide range of design alternatives.

In contrast to dynamic approaches, HLS tools use program dependence graphs (PDG) that statically capture both control and data dependences. Static analysis is inherently conservative in its dependence analysis, because it is used for generating code and hardware that works in all circumstances and is built without run-time information. The following subsections give details about the optimization phase (Section 3.2.2) and the generation phase (Section 3.2.3) of HAPE, presented in Fig. 8a.

3.2.2. Optimization phase

The main goal of the optimization phase is to represent the fundamental dependences of the application by removing operations that are not required for the hardware implementation. For example, our framework can remove additional load/store operations by buffering data in internal registers within HAs. This phase applies typical hardware optimizations for the dynamic execution trace, respecting three requirements:

- Only represent necessary computation and memory instructions in the program's trace.
- Remove all false memory dependences between dynamic instructions, keeping only true read-after-write dependences within the DDDG graph.
- Remove unnecessary dependences and redundant load/store operations generated due to stack overheads and register spilling [17] in order to save memory bandwidth.

Our proposed framework leverages the LLVM compiler framework [17,40] for execution trace collection. The core of LLVM is an intermediate representation (IR), which is essentially machine independent assembly language. As shown in Fig. 7, high-level source code (C/C++) is translated into LLVM's IR then analyzed and modified by a series of compiler optimization passes. In this subsection, we highlight the important key elements of the LLVM framework, as well as the Dynamic Data Dependence Graph (DDDG) graph representation [17].

The LLVM IR is a single static assignment (SSA) form, which prohibits variable re-use, guaranteeing a 1-to-1 correspondence between an instruction and its destination register. As illustrated in Fig. 7, register names in the IR trace are prefixed by %. Types are explicit in the IR. For example, i64 specifies a 64-bit integer type.

From Fig. 7, we can note that LLVM instructions (IR trace) are simple enough to directly correspond to hardware operations (e.g., a load from memory, or an arithmetic computation). Canis and coworkers [6,17,28] operate directly with the LLVM IR, scheduling the instructions into specific clock cycles. Scheduling operations in hardware requires knowing data dependencies between operations.

As illustrated in Fig. 7, a DDDG is a directed, acyclic graph, where nodes represent computation and edges represent dynamic data dependences between nodes. The DDDG graph is generated from an execution trace (IR) to represent program behaviors. This technique avoids the false data dependences created by the static analysis.

3.2.3. Generation phase

Respecting the above requirements, our proposed framework generates the DDDG graph from an optimistic execution trace. It uses a high-level, machine independent Intermediate Representation (IR) provided by the open-source LLVM compiler [40]. Code analysis, optimization and modification are performed on IR via LLVM passes in order to remove the instructions that are not part of the program [17].

In contrast to existing works, the proposed framework only focuses on the relevant sub-trace based on the pragma settings provided by designers, instead of analyzing the entire program trace (Fig. 8a). This makes the estimations very fast even for applications with relatively large input size. The goal of the *generation* phase is to generate an optimized DDDG from the dynamic execution sub-trace. The outcome of these two phases is a pre-RTL, performance-area-power estimation of HAs across a wide range of design alternatives. From the generation phase, we can get observations that will later help us to understand how the hardware resources are consumed in the system.

Based on the optimistic program's IR trace, we can easily detect the different functional units (memory/computation operations) and then generate the DDDG graph to represent HAs. Streaming interfaces are supported by our framework to simplify HA integrations in FPGA-based SoCs. As our approach is based on dynamic analysis, all the data dependences are known after obtaining the execution trace.

3.3. Target hardware architecture

With FPGA-based accelerators, designers have the opportunity to assign the right processing unit to the right task in order to achieve the application constraints. This level of control enables the target platform to meet the demands of modern applications requiring high performance while meeting a low power consumption. The Zynq-7000 All programmable SoCs [3,41] are examples of such platforms in the current embedded market.

To explore various configurations that can be efficiently used for the diverse applications, an FPGA-based accelerator platform is used as it can provide efficient hardware implementations. Such platforms typically contain a processor for executing the software segments of the application and other HAs that can be used to accelerate the critical components in the application. FPGA has been widely used to implement the HAs for applications.

The Zynq-7000 SoC [41] consists of a processing system (PS) and a programmable logic (PL), as shown in Fig. 9. Xilinx Zynq platforms typically integrate an application processor such as the dual-core Cortex A9 from ARM, with a highly reconfigurable FPGA fabric. These platforms are becoming increasingly complex and will integrate more and more processors and logic elements. The communication between the processing system and the programmable logic is achieved by AXI4 interconnection [13]. In our FPGA implementations, HAs are attached to the AXI4 interconnection via the AXI master interface. The communication and the synchronization between the main processor and the different HAs are done through the AXI4 stream interface [13] using the Direct Memory Access (DMA) to exchange data between the main memory (DDR) and the local memories (BRAMs) of the HAs. The advantage of using a DMA is that the processor can execute other computations while the accelerator performs its work. In addition, this interconnect provides a pipelined control that enables the software running on the processor to queue multiple tasks requests, reducing its latency.

Table 2 lists the different symbols used in the modeling of the area/power metrics.

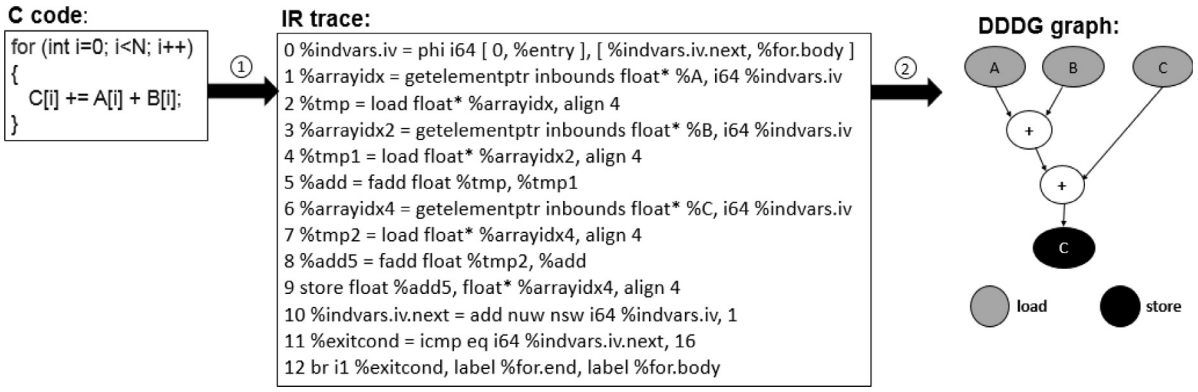
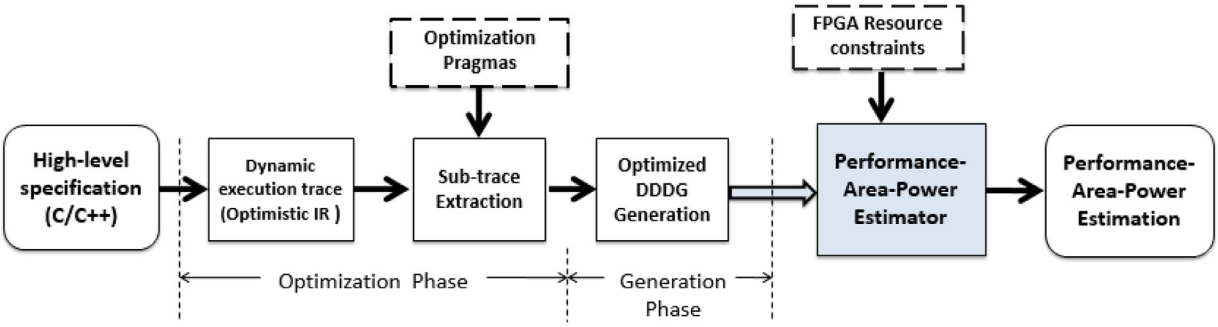


Fig. 7. C code, IR trace and its corresponding DDDG graph.

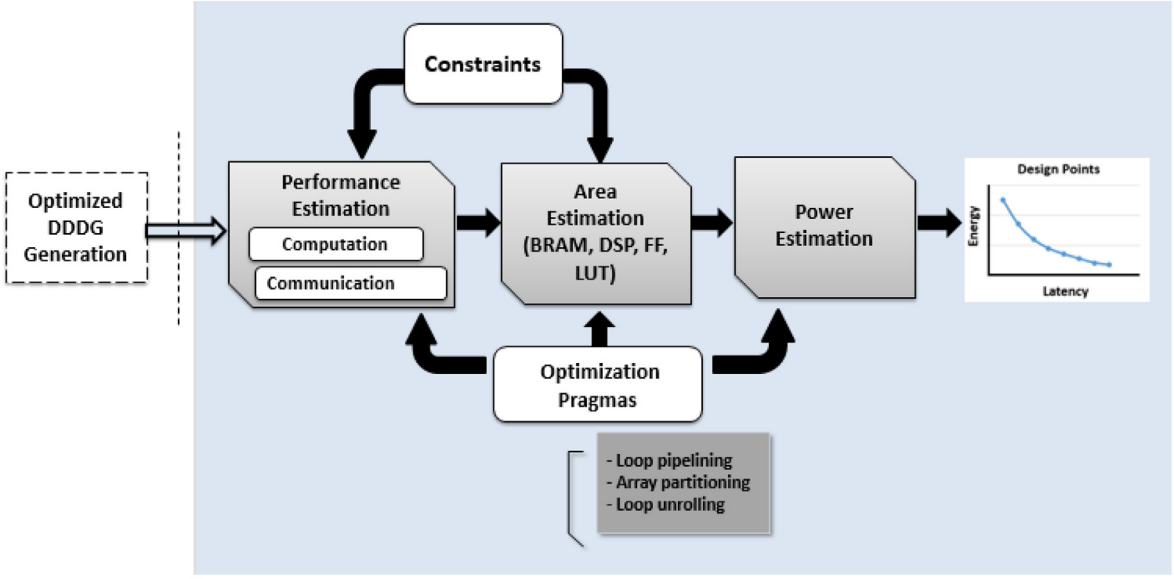
3.4. Area analysis: an analytic area estimation model

The performance/power trade-off is an important challenge in the design space exploration process. Achieving the high performance is constrained by the number of available resources that can be synthesized on FPGA. Area occupation is measured in terms of the required FPGA resources: *Look-Up Tables (LUTs), Flip-Flop registers (FFs), BRAMs* and *DSPs*. As illustrated in Fig. 8b, the area estimation model uses the pragma settings provided by users, to construct the execution sub-trace and generate the optimized DDDG graph.

In this paper, we estimate the area occupation of the HAs with the following assumptions: (a) hardware functional units associated with nodes (DDDG) follow the default setting of Xilinx Vivado HLS [8] including FPGA resource occupation. For instance, we assume that a 32-bit floating-point addition node is mapped to a pipelined floating-point add (FA) unit, which consumes two DSPs and zero BRAM; (b) Each memory bank has two reads and one write ports; (c) Resource constraints are modeled for DSP, BRAM, LUT and FF.



(a) Performance-area-power estimation framework



(b) The proposed framework for performance-area-power estimator

Fig. 8. The HAPE overview.

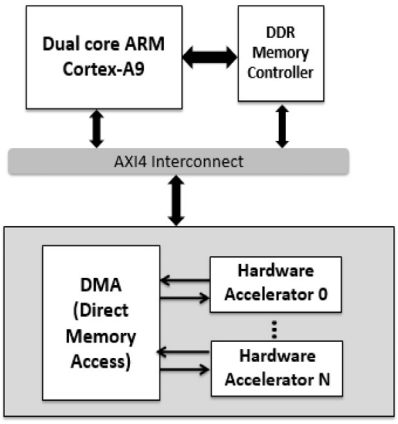


Fig. 9. Simplified block diagram of ZYNQ architecture.

Our developed area estimation model applies optimization pragmas as well as resource constraints to the DDDG graph to explore a large design space with different trade-offs (Fig. 8a).

In this subsection, we develop an efficient analytic model to estimate the area occupation of the application based upon different parameters generated from the DDDG graph, such as number of loop levels, loop bounds, input data size, etc. Our analysis allows then to estimate the application's hardware usage in order to reject unfeasible designs. In addition, the AXI4 stream interface can also consume a significant number of FPGA resources. This quantity depends on the number of input data arrays, denoted *nIA* and can be estimated by Eq. (1). DMA_{area} represents the FPGA resources required by a DMA controller.

$$Com_{area} = (nIA)*DMA_{area} \quad (1)$$

In this work, we use *nFA, nFS, nFM* and *nFD* to represent respectively the total number of floating addition, floating subtraction, floating multiplication and floating division operations required for the application execution. These values are obtained from the DDDG graph generated from the dynamic execution trace (Fig. 8a). N_{op} represents the number of *computation* operation nodes (multiplication, addition, etc.), while N_m represents the total number of *memory* operations extracted from the DDDG graph. N_{load} and N_{store} represent the number of memory *load* and *store* operations respectively.

Let's consider a nested loop $K = \{K_1, .,K_i, .,K_L\}$, where K_L is the innermost loop level. B_K is the bound of the nested loop K. The Iteration Latency, *IL*, is the number of clock cycles required to perform a single iteration of the loop. L_K is the number of loop levels in the nested loop K. The number of single-level loops in an application is represented by S_l. In this paper, the total number of nested loops in a given application is represented by n. The constants used in analytic equations, denoted *Ci*, are determined by analyzing the results of the RTL designs generated from Vivado HLS tool. These constants depend on the used FPGA family. They are collected from the default setting of Vivado HLS tool after its hardware resource estimation. The values of these constants are summarized in Table 3. With the generated DDDG (Fig. 8a), our framework estimates the different hardware resources (LUTs, FFs, etc.) of the FPGA-based accelerator for the given algorithm without generating RTL implementations.

Loop unrolling can be applied at any loop level. We can handle both

Table 2
List of symbols.

Symbol	Description	Obtained by/from
n_{IA}	Number of input data arrays	Application profiling
DMA_{area}	FPGA resources required by a DMA controller	Proposed area model
N_{load}, N_{store}	Number of memory load and store operations respectively	DDDG generation
N_{op}	Number of computation operation nodes (multiplication, addition, etc.)	DDDG generation
N_m	Total number of memory operations extracted from the DDDG graph	DDDG generation
n_{FA}	Total number of floating addition operations	Application profiling
n_{FS}	Total number of floating subtraction operations	Application profiling
n_{FM}	Total number of floating multiplication operations	Application profiling
n_{FD}	Total number of floating division operations	Application profiling
C1, C2, C3, C4	Constants, which represent the number of LUT resources required to perform respectively a single FA, FM, FD and FS operation	Default setting of Vivado HLS
C5,C6, C7, C8	Constants, which represent the FF resources required to perform respectively a single FA, FM, FD and FS operation	Default setting of Vivado HLS
C9, C10, C11	Constants, which represent the number of DSP resources required to perform respectively a single FA, FM and FS operation	Default setting of Vivado HLS
T	Array partitioning type, T = {cyclic; complete; block}	Pragma settings
P_f	Partition factor	Pragma settings
S_A	Size of an array A, measured in words	Application profiling
S_{BRAM}	Size of a BRAM (in bits) in FPGA	User settings
n_A	Number of arrays in a given application	Application profiling
$BRAM_b$	Number of BRAMs per memory bank	Proposed area model
$BRAM_T$	Estimated total amount of BRAM required to implement the application	Proposed area model
K_L	Innermost loop level	Application profiling
B_K	Bound of the nested loop K	Application profiling
IL	Number of clock cycles required to perform a single iteration of the loop	Application profiling
L_K	Number of loop levels in the nested loop K	Application profiling
S_l	Number of single-level loops in the application	Application profiling
n	Total number of nested loops in a given application	Application profiling
BRAMs	Generated BRAM utilization, rounded to power of two	Proposed area model
$\alpha = 32$; $\beta = 14$	Two empirical values	Default setting of Vivado HLS
e	A simple exponential constant	Default setting of Vivado HLS
U	Loop unrolling factor	Pragma settings
V, V1, V2, V3	Four constants	Default setting of Vivado HLS
n_{BRAMs}	Total number of the block RAMs, measured in %	Proposed area model
n_{FF}	Total number of Flip-Flops, measured in %	Proposed area model
n_{LUT}	Total number of Look-Up tables, measured in %	Proposed area model
C_F, C_L, C_B	Coefficients representing the individual effects of Flip-Flop registers, LUTs and BRAMs respectively	Proposed power model
p_i	Pipelining loop level, disabled if i = 0; otherwise, i represents the pipeline level indicating that this pragma is applied to the loop level i of the nested loop	Pragma settings
u_j	Loop unrolling factor, disabled if j = 0; otherwise, j represents the unrolling factor	Pragma settings
a_k	Array partitioning factor, disabled if k = 0; otherwise, k represents the number of factor	Pragma settings

Table 3
The default area estimated by Vivado HLS for the different operations for the Xilinx ZC702 platform.

Operation type (32 bits)	LUT	FF	DSP
FA	C1 = 390	C5 = 205	C9 = 2
FM	C2 = 321	C6 = 143	C10 = 3
FD	C3 = 994	C7 = 761	0
FS	C4 = 390	C8 = 205	C11 = 2

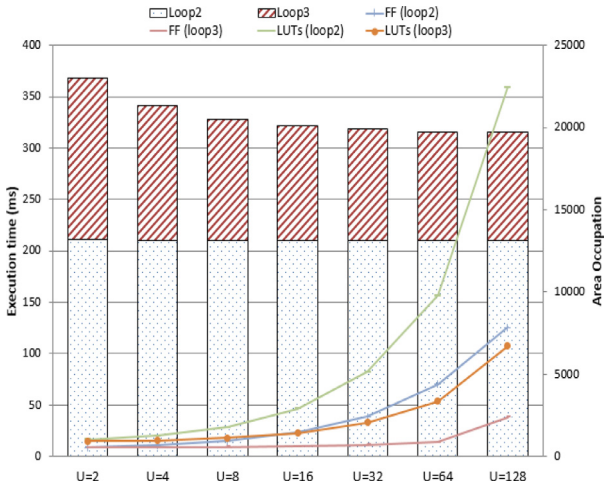


Fig. 10. Execution time and area occupation for MM benchmark with different loop unrolling factors. Here, $L_K = 3$ and $U = \{2, 4, 8, 16, 32, 64, 128\}$.

nested and single loops in the application. Given an unrolling factor $U = \{U_1, .,U_i, .,U_L\}$, where U_i is the unrolling factor of loop level K_i in the nested loop K. Loop unrolling removes dependences between loop index variables in the DDDG. In fact, when enabling an unrolling factor U, U loop iterations can be executed in parallel if there is no loop carried dependences across the different loop iterations.

Fig. 10 shows execution time and resource usage results of the MM benchmark obtained from Xilinx Vivado HLS. The MM benchmark includes one nested loop with 3 loop levels. In this example, loop3 is the innermost loop and unrolling pragma is applied at loop level2 and loop level3 with different factors.

As illustrated in Fig. 10, loop unrolling pragma has an impact on the FF and LUT resources as well as total execution time in FPGA-based accelerators. Furthermore, unrolling the innermost loop of the nested loop K, achieves a high performance with less area occupation.

The results, presented in Fig. 10, show that applying loop unrolling gives a good performance/area trade-off. In fact, when increasing the unrolling factor, the performance boost associated with loop unrolling pragma comes at the cost of increased FPGA resource consumption. However, unrolling the innermost loop "*loop level3*" of the nested loop "*loop3*", achieves a high performance with less FPGA resources compared to the other configurations.

3.4.1. BRAM estimation

The array partitioning pragma has a great impact on the BRAM resource utilization. The memory bandwidth can be improved by splitting up the original arrays into multiple independent memory banks. Our developed BRAM Resource Estimation (BRE) algorithm allows to rapidly estimate the required BRAM resources to map the application on a HA implementation. By increasing array partitioning factor, HLS tool can exploit more parallelism in the multi-kernel application. However, it requires more FPGA resources. It takes the parameters *nA, T, Pf,* S_A and S_{BRAM} as inputs and generates $BRAM_T$, which is the estimated total amount of BRAM required to implement the application. When applying the array partitioning pragma, we should specify the array address A, the partition factor Pf, and the array partitioning type T, where $T = \{cyclic, complete, block\}$. Varying the partition factor Pf as well as the arrays data size, may increase or decrease the required BRAM resources. The S_A, respectively S_A**Bits*, parameter represents the size of an array A, measured in *words*, respectively in *bits*. *nA* represents the number of arrays in a given application. S_{BRAM} is the size of a BRAM (in *bits*) in FPGA. It is a generic parameter that can be set by the designer. $BRAM_b$ is the number of BRAMs per memory bank.

Without loss of generality, we assume that each memory bank has two read and one write ports. The developed BRE algorithm (Algorithm 1) is useful for *cyclic* and *block* types. In fact, no BRAMs are required for the *complete* type, since in this type, the array is completely split into individual registers. As a result, we use Flip-Flops (FF) instead of BRAMs. It should also be noted that the generated BRAM utilization *BRAMs* is always rounded to power of two, as shown in lines 9 and 14 of Algorithm 1.

3.4.2. LUT estimation

FF and LUT usage become more crucial in the area estimation metric. To estimate the total number of LUT resources (LUT_T) of a given application, we sum three important parameters: LUT_m, LUT_{op} and LUT_{ex}, as presented in Eq. (2). LUT_m (Eq. (3)) corresponds to the LUT consumed by the multiplexer resources. LUT_{op} (Eq. (4)) represents the number of LUT consumed by the computation operation nodes, while LUT_{ex} (Eq. (5)) represents the LUT resources used by any expressions such as multipliers, adders and comparators. These information are automatically generated from our proposed area estimation model.

Based on empirical method, we use $\alpha = 32$ and $\beta = 14$. These two empirical values follow the default setting of Vivado HLS and have been tested for different configurations. B_K is the bound of the nested loop K, where B_K can be represented as $B_K = 2^e + c$. e is a simple exponential constant. The term U represents the loop unrolling factor. In this paper, the unrolling factors correspond to the divisors of the loop bound B_K. V, $V1$, $V2$, $V3$ are four constants, which are represented by the following equations.

$$LUT_T = LUT_{op} + LUT_m + LUT_{ex} \tag{2}$$

$$LUT_m = (N_{store} + N_{op})^*\alpha + \left(\sum_{K=1}^{K=n} L_K + S_l\right)^*V + N_{load}^*\beta \tag{3}$$

Where $N_{op} = nFA + nFM + nFS + nFD$
$V = e + 1$

$$LUT_{op} = nFA^*C1 + nFM^*C2 + nFD^*C3 + nFS^*C4 \tag{4}$$

$C1$, $C2$, $C3$ and $C4$ are four constants, which represent the number of LUT resources required to perform respectively a single floating addition (*FA*), floating multiplication (*FM*), floating division (*FD*) and floating subtraction (*FS*) operation, as presented in Table 3.

$$LUT_{ex} = \left(S_l + \sum_{K=1}^{K=n} L_K\right)^*(V1 + V2 + V3) + (U - 1)^*V1 \tag{5}$$

Where $B_K = 2^e + c$; $V1 = e + 1$
$V2 = e^*2$; $V3 = e + 2$

3.4.3. FF Estimation

As FPGAs can exploit diverse types of parallelism within applications, HLS-based techniques typically generate higher performance accelerators at the cost of more area occupation. Quantifying the necessary number of FF resources depends on various parameters such as the type of an operation (load/store, computation, etc.), the total number of operations required to execute an application, the number of the loop levels, etc. Each operation is represented by a node in the DDDG graph, as illustrated in Fig. 8a. Based on an empirical study, the total number of FF resources (FF_T) can be estimated using Eq. (6).

Inputs: nA, T, Pf, S_A, S_{BRAM}
Output: $BRAM_T$ ▷ The estimated BRAMs.

1: **begin**
2: $BRAM_T \leftarrow 0$;
3: $BRAM_b \leftarrow 0$;
4: $R \leftarrow 0$;
5: $BRAMs \leftarrow 0$;
6: **for** $i \leftarrow 1$ to nA **do** ▷ For each array.
7: **if** $T == cyclic$ or $T == block$ **then**
8: $BRAM_b \leftarrow (S_A * Bits)/(pf * S_{BRAM})$;
9: $R \leftarrow Round(BRAM_b)$; ▷ Rounding to nearest integer.
10: $BRAMs \leftarrow pf * R$;
11: **if** $BRAMs == power(2, x)$ **then** ▷ Verify if $BRAMs$ is a power of two.
12: $BRAM_T \leftarrow BRAMs$;
13: **else**
14: $BRAM_T \leftarrow 2^{round[log_2(BRAMs)]}$; ▷ Find the nearest power of two.
15: **end if**
16: **else**
17: $BRAM_T \leftarrow 0$; ▷ The type T is complete.
18: **end if**
19: **end for**
20: return $BRAM_T$;

Algorithm 1. BRAM Resource Estimation (BRE) algorithm.

DOI : 10.1016/j.micpro.2018.08.004

$$FF_T = FF_r + FF_{op} \quad (6)$$

FF_{op} (Eq. (7)) is the FF consumed by the arithmetic operations extracted from the DDDG graph, where $C5$, $C6$, $C7$ and $C8$ are constants (Table 3). These constant values follow the default setting of Vivado HLS.

$$FF_{op} = nFA*C5 + nFM*C6 + nFD*C7 + nFS*C8 \quad (7)$$

FF_r (Eq. (8)) represents the FF consumed by the register resources used to map an application into an FPGA. It also includes the FF consumed by the memory load/store instructions.

$$FF_r = (N_m + N_{op})*\alpha + IL + \left(2*V* \sum_{K=1}^{K=n} L_K\right) + S_l*V \quad (8)$$

Where $N_{op} = nFA + nFS + nFD + nFM$
$N_m = N_{load} + N_{store}$; $B_K = 2^e + c$
$V = e + 1$; $\alpha = 32$

3.4.4. DSP estimation

The DSP resource consumption depends on the total amount of compute operations (*subtraction, multiplication, addition* and *division*) required to execute an application. To estimate the total DSP resources of a specific SoC design, we measure the computation operation nodes within the generated DDDG graph. $C9$, $C10$ and $C11$ (Table 3) are three constants, which represent the number of DSP resources required to perform respectively a single *FA, FM* and *FS* operation. The total DSP resources (DSP_T) required to implement an application on an FPGA is estimated using Eq. (9).

$$DSP_T = nFA*C9 + nFM*C10 + nFS*C11 \quad (9)$$

In this paper, we estimate the area occupation for FPGA-based accelerators within resource budget. The Area Efficiency, denoted *AE*, is defined using Eq. (10), where *BRAM, DSP, FF* and *LUT* represent the available BRAM, DSP, FF and LUT resources of a given FPGA platform, while *bram, dsp, ff* and *lut* are FPGA resources consumed by the current implementation.

$$AE = max\left(\frac{bram}{BRAM}, \frac{dsp}{DSP}, \frac{ff}{FF}, \frac{lut}{LUT}\right) \quad (10)$$

A given HA implementation can fit into the FPGA if and only if $AE \leq 1$. Consequently, generated area results are equal to the FPGA resources required by the current configuration. Otherwise, the design exceeds the FPGA resource capacity, and the corresponding configuration is automatically rejected from the design space.

3.5. Power analysis: an analytic power estimation model with different pragma combinations

With the increasing complexity of recent SoC designs, the development of a high-level, simple and accurate power estimation model for the FPGA-based accelerators thus becomes inevitable. High-level analytic models guide the design space exploration by estimating the various metrics (performance, area, power) of the different design points on FPGAs. This helps to reduce the total runtime to perform large and complex design space.

In modern FPGA designs, reconfigurable resources can be mainly categorized into Flip-Flops registers (FFs), Block RAMs (BRAMs), Digital Signal Processing (DSP) slices and Look-Up Tables (LUTs). Based on the implementation results, we note that the different SoC designs, used in this work, consume mainly FFs, BRAMs and LUTs resources. It is important for designers that use the recent FPGAs to understand how a design consumes the various FPGA resources. In addition, with the advance of hardware acceleration devices such as FPGAs, it is possible to achieve performance/power improvement, while offering more flexibility than an ASIC can provide. In this paper, we propose an approach for accurately estimating the power consumption of different FPGA-based accelerators. This estimation can be made based on an analytic model that describes the dependence of power consumption of the embedded system on certain parameters such as the number of FPGA resources (FFs, LUTs, etc.). The different symbols used in the power modeling, are listed in Table 2. According to the target system components presented in the previous subsection, the total power consumed by the system (P_{Tot}) when it executes a software task can be expressed by Eq. (11):

$$P_{Tot}(mW) = P_{PS} + P_{DDR3} + P_{PL} + P_{BRAM} \quad (11)$$

where P_{PS} corresponds to the power consumed by the processing system including the dual ARM cortex cores while executing a program. P_{PL} corresponds to the power consumed by the programmable logic or FPGA. P_{BRAM} represents the power consumed by the Block RAMs available in the Zynq Programmable Logic. The term P_{DDR3} represents the power consumed by the DDR3 external memory.

In our system, the total power consumption of PL, BRAM, PS, DDR3 has been measured using the different rails that are available in the Xilinx ZC702 platform [42,43]. Subtracting the static power of the board from the total power measured for a SoC design provides the dynamic power consumption of Zynq for that design.

In this paper, the ARM processor only controls the whole system, while the HAs are used to execute the different applications. Based on empirical method, we use P_{PS} = 354mW and P_{DDR3} = 586mW. These two empirical values are obtained from the above power measurement method. The following general equation represents the model for P_{Tot} for Xilinx ZC702. The term n_{BRAM} represents the total number of the block RAMs, while n_{FF} and n_{LUT} represent the total number of Flip-Flops and Look-Up tables, respectively, used to execute the application on a HA implementation. In this paper, n_{BRAM}, n_{FF} and n_{LUT} are extracted from our framework's area model (Section 3.5) and measured in %. P_{Tot} is expressed in milliwatts (mW).

$$P_{PL} = C_F*n_{FF} + C_L*n_{LUT} \quad (12)$$

$$P_{BRAM} = C_B*n_{BRAM} \quad (13)$$

In Eqs. (12) and (13), C_F, C_L, and C_B are coefficients representing the individual effects of Flip-Flop registers, LUTs and BRAMs respectively. In this paper, the proposed linear equations represent the relation between power consumption and number of FPGA resources estimated by the proposed area model.

E_{Tot} is measured using Eq. (14), where T_{Tot} is the total time that correspond to the execution of the application on the HA implementation.

$$E_{Tot} = P_{Tot}*T_{Tot} \quad (14)$$

HLS technique offers various architectural design options with different trade-offs via pragmas (loop unrolling, pipelining, array partitioning). The hardware resource utilization and the total power consumption are dependent on the used pragmas. Therefore, it should estimate the total power consumption of different design points with various pragma combinations. In the following subsections, we present the different equations used to estimate the total power consumption of an FPGA-based accelerator system with/without pragmas.

3.5.1. With loop pipelining

HLS tools provide optimization pragmas for users to explore and evaluate diverse hardware architectures. Loop pipelining, unrolling and array partitioning are among the prominent pragmas that have significant impact on hardware resource utilization and power consumption. Applying optimization pragmas presents a performance/area trade-off.

When loop pipelining pragma is applied, P_{PL} and P_{BRAM} are estimated using Eqs. (15) and (16).

$$P_{PL} = 93*n_{FF} - 41*n_{LUT} \quad (15)$$

$$P_{BRAM} = 0.07*n_{BRAM} \quad (16)$$

The first and second terms in Eq. (15), represent the individual effect of Flip-Flops and LUTs on power consumption. P_{PL} and P_{BRAM} are estimated in milliwatts (mW).

3.5.2. With array partitioning

Similar to Vivado HLS [8], our proposed framework supports *block, cyclic* and *complete* array partitioning strategies (Fig. 2). It enables array partitioning by mapping addresses of memory nodes (load and store) in the DDDG graph to memory banks.

When array partitioning pragma is applied, P_{PL} and P_{BRAM} are estimated using Eq. (17) and Eq. (18), respectively.

$$P_{PL} = -40*n_{FF} + 32*n_{LUT} \quad (17)$$

$$P_{BRAM} = 0.03*n_{BRAM} \quad (18)$$

n_{BRAM}, n_{FF} and n_{LUT} are measured in %.

3.5.3. With loop unrolling

Loop unrolling is a technique to exploit instruction-level parallelism inside loop iterations, while loop pipelining enables different loop iterations to run in parallel.

Considering loop unrolling pragma, P_{PL} and P_{BRAM} are estimated using the following Equations.

$$P_{PL} = -16*n_{FF} + 18*n_{LUT} \quad (19)$$

$$P_{BRAM} = 0.06*n_{BRAM} \quad (20)$$

3.5.4. With loop pipelining, unrolling, array partitioning

To assist designers in finding good-quality accelerator designs through appropriate pragma settings, it is vital to obtain performance/power estimations early in the design stage without the need for time-consuming manual RTL creation. Optimizing the application using pragmas, such as loop pipelining, unrolling and array partitioning, considerably reduces the total execution time compared to the results without pragmas.

When the above optimization pragmas are applied, P_{PL} and P_{BRAM} are estimated using Eq. (21) and Eq. (22), respectively.

$$P_{PL} = 14*n_{FF} - 3*n_{LUT} \quad (21)$$

$$P_{BRAM} = 0.14*n_{BRAM} \quad (22)$$

3.5.5. With loop pipelining and unrolling

According to optimization pragmas provided by the designer, our proposed framework estimates the total power consumption of the different FPGA-based accelerator systems. The total power consumption of PL (P_{PL}) and BRAMs (P_{BRAM}), are estimated using Eqs. (23) and (24).

$$P_{PL} = 0.875*n_{FF} + 5.12*n_{LUT} \quad (23)$$

$$P_{BRAM} = 0.06*n_{BRAM} \quad (24)$$

3.5.6. With loop pipelining and array partitioning

The total power consumed by FPGA (P_{PL}) and the block RAMs (P_{BRAM}) are estimated using Eqs. (25) and (26), while considering loop pipelining and array partitioning pragmas.

$$P_{PL} = -1.2*n_{FF} + 7.7*n_{LUT} \quad (25)$$

$$P_{BRAM} = 0.03*n_{BRAM} \quad (26)$$

3.5.7. With loop unrolling and array partitioning

Our proposed framework can rapidly estimate the power consumption for other combinations of pragmas, such as loop unrolling and array partitioning pragmas with different unrolling and array partitioning factors.

In this case, P_{PL} and P_{BRAM} are estimated using Eqs. (27) and (28).

$$P_{PL} = 3.7*n_{FF} + 2*n_{LUT} \quad (27)$$

$$P_{BRAM} = 0.48*n_{BRAM} \quad (28)$$

3.5.8. Without pragmas

Our proposed framework also allows designers to use the default implementation of the application without applying any pragmas. This can reduce the hardware resource utilization. However, it does not optimize the source code. P_{PL} and P_{BRAM} are estimated using Eq. (29) and Eq. (30), respectively.

$$P_{PL} = 164*n_{FF} - 80*n_{LUT} \quad (29)$$

$$P_{BRAM} = 0.1*n_{BRAM} \quad (30)$$

4. Experimental results

The main goal of our proposed framework is to efficiently explore the design space and provide accurate area and power estimates for FPGA-based accelerators. In order to validate our approach, we have implemented fourteen (14) benchmarks from Polybench suite [44] and CHStone [45]. These benchmarks represent kernels of real applications in wireless communications, video processing, signal processing, etc. In addition, all these applications operate on large matrices with regular data access patterns. In this paper, the different benchmarks have been mapped on several HA architectures, running at 100 MHz, over a wide range of input data sizes. We use Xilinx Vivado HLS version 2014.4 and Xilinx ZC702 Evaluation Kit [43] to validate our estimations.

Table 4 lists the various benchmarks used in this work, specifying their application domain (column 2) as well as their different input data size (column 3). For each benchmark, the input data size is chosen such that the benchmark can fit into the available resources (LUT, BRAMs, DSP, etc.) of our target FPGA device, Xilinx ZC702 Evaluation Kit [43]. In this work, the different input data sizes are measured in *words*. The size of the data bus is 32-bit.

Fig. 11 shows the instruction distribution (in %) for the different benchmarks. These values are obtained from the program's IR trace, as explained in Section 3.3. The IR trace includes different instruction information such as the type of instructions (e.g., memory, computation) required by the program to generate then the DDDG graph that models HA behaviors.

4.1. Power measurement

Recent Zynq devices combine a dual-core ARM Cortex A9 processor and an FPGA fabric in the same die and in different power domains. According to device vendors, recent 28nm FPGAs consume 50% less power than previous generations (e.g., 65-nm FPGA devices) [46].

Many modern FPGA boards include Power Management Bus (PMBus) Controllers [47]. The PMBus is a serial interface specifically designed to support power management protocol. It facilitates the communication with power converters and other devices in a power system [42]. This technology means that software or hardware running

Table 4
Used benchmarks.

Benchmarks	Domain	Input data size
CONV2D CONV3D	Convolution	4, 8, 32, 64, 128
MM ATAX BICG GEMM MVT SYR2K SYRK	Linear Algebra	4, 8, 32, 64, 128, 256
CORRELATION COVARIANCE	Data Mining	4, 8, 32, 64, 128
IDCT	Image processing	8
AES	Encryption	32
FFT	Signal processing	8

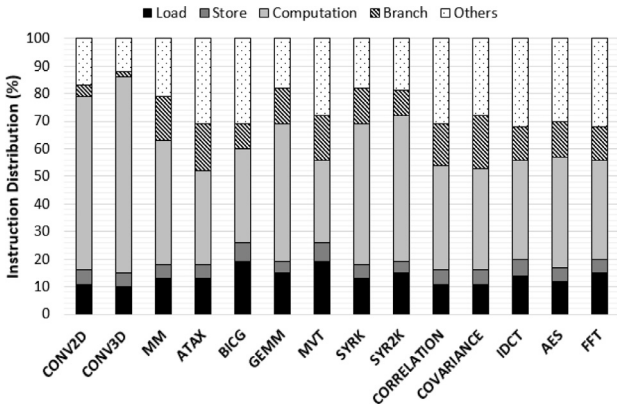


Fig. 11. Instruction distribution for the different benchmarks.

on the device has access to a controllable power supply.

This is the case with the latest Xilinx series 7 FPGAs, including ZC702, that use the Texas Instrument (TI) UCD92xx PMBus controller [48]. The TI UCD92xx series [42] is a family of digital power controller which supports a wide range of commands that allow an external host to configure, control, and monitor the controller through an I2C electrical interface using the PMBus command protocol.

In this paper, we employ the Fusion Digital Power Designer software package provided by TI [48]. This software package has several tools that are able to communicate with the UCD92xx series of controllers from a Windows-based host computer. It requires the use of a USB Interface Adapter EVM to connect the PMBus (I2C) interface of the UCD9248 controller and the USB port in the host computer. The TI Fusion Digital Power tool [48] reads the voltage and current information of the power supply regulators monitored by the UCD9248 Power controllers, calculates the average power of individual supply and finally calculates the total power consumed by the ZC702 board.

In this section, we will discuss the implementation of the different benchmarks. By applying the equations obtained in the previous section, we were able to estimate power consumption for the different SoC designs with various trade-offs through HLS optimization pragmas.

4.2. Rapid estimation

Table 5 shows the exploration time of the proposed framework (Fig. 8a). The exploration time of our approach includes the overhead necessary to execute the application and generate the IR trace. The results have been compared to Vivado HLS tool, for the same design space. Table 5 lists three selected benchmarks (MM, BICG and CONV3D) in column 1, while column 2 shows the input data size used for each one. Column 3 presents the total number of explored configurations for each benchmark varying pipeline options, array partitioning factors and types. As an example, for MM (Matrix Multiplication) benchmark, we explored 119 design points with different pragma combinations in just 25 seconds. Our proposed framework is based on accurate modeling techniques and optimization pragmas, and can provide considerable fast area and power estimates for many designs compared to Vivado HLS tool.

Table 5
DSE time for MM, BICG and CONV3D: Vivado HLS tool vs. proposed framework.

Application	Input size	Design space	DSE time	
			Vivado HLS	Proposed framework
MM	128*128	119	18 h	25 s
BICG	256*256	133	1 day	53 s
CONV3D	32*32*32	120	2.61 days	2 min

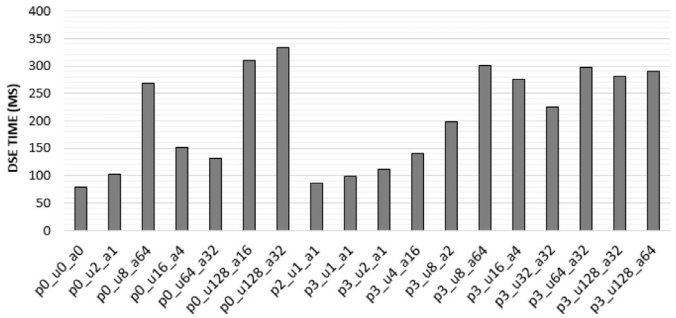


Fig. 12. Execution time in ms for the MM benchmark with different (pipeline_unrolling_arrayPartition) configurations.

An example of MM design space is shown in Fig. 12. The X-Axis shows some selected MM design configurations. The Y-Axis denotes the exploration time of each configuration in milliseconds (ms). Each configuration, denoted (pi_uj_ak), is expressed as follows:

- pi: pipelining loop level, which is disabled when $i = 0$; otherwise, i represents the pipeline level indicating that this pragma is applied to the loop level i of the nested loop.
- uj: loop unrolling factor, which is disabled when $j = 0$; otherwise, j represents the unrolling factor.
- ak: array partitioning factor, which is disabled when $k = 0$; otherwise, k represents the number of factor.

Experiments prove that HAPE can perform a large design space within the strict time-to-market with different pragma combinations. In fact, our proposed framework skips the time-consuming RTL generation, synthesis, and simulation process for different sizes of input data. Consequently, it explores rapidly various hardware configurations in the order of seconds to minutes, over a large multi-level parallelism design space using different pragma combinations.

4.3. Synthesis results

Fig. 13 plots the resource utilization results of ATAX benchmark while varying pipeline levels, unrolling and partitioning factors. The main criteria for area usage are LUTs, BRAMs and FFs resources of an FPGA. Occupation indicates the percentage of FPGA resources utilized by the different configurations.

As illustrated in Fig. 13, the configuration without pragmas (p0_u0_a0) consumes 6% of the available FFs and 11% of LUTs within the FPGA. The configuration (p2_u64_a8) considering loop pipelining, unrolling and array partitioning, occupies 12% of FFs and 19% of LUTs. By increasing partitioning factor, we can exploit more parallelism in the

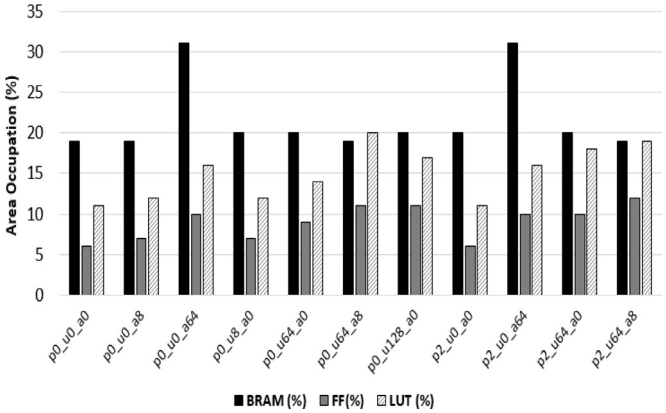


Fig. 13. Area Occupation (%) of ATAX benchmark with different pragma combinations.

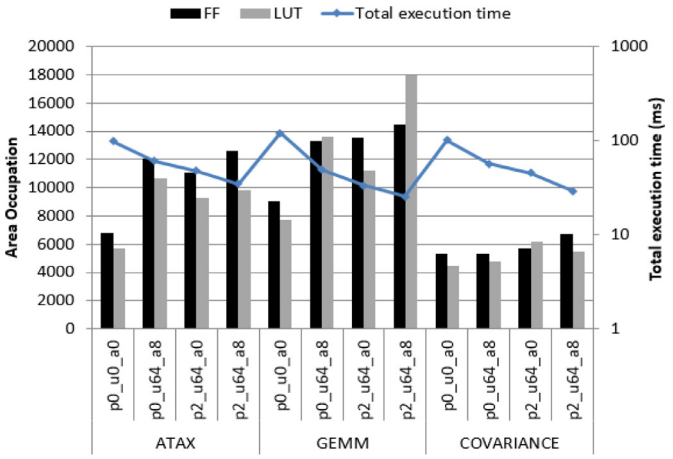


Fig. 14. The trade-off between the estimated area occupation and the total execution time for ATAX, GEMM and COVARIANCE benchmarks with different pragma combinations.

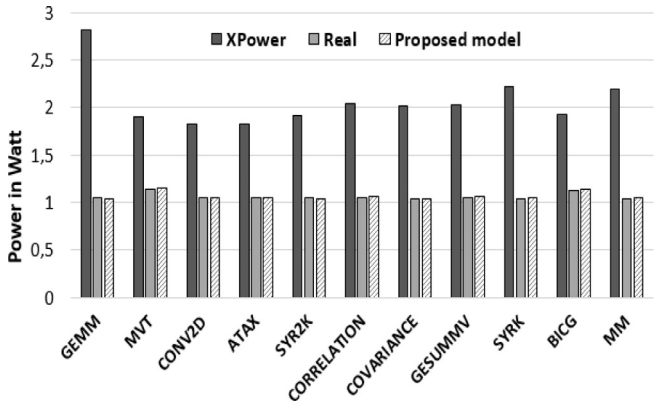


Fig. 16. Our proposed power model vs. XPower Analyzer estimation and measured power with loop pipelining.

function, but this requires more hardware resources. Therefore, it is obvious that the used BRAMs for (p0_u0_a64) is more than that used for (p0_u0_a0) configuration. This occurs because applying array partitioning pragma with the partitioning factor of 64 needs to split the arrays into multiple banks so that several memory accesses can be executed simultaneously.

Based on the experimental results, we demonstrate that applying optimization pragmas has a strong impact on the hardware resource utilization and therefore on the power consumption of SoC designs. An additional optimization pragma significantly increases the resource utilization compared to the configurations without pragmas.

Fig. 14 depicts the estimated area occupation versus the total execution time results in milliseconds (ms) for ATAX, GEMM and COVARIANCE benchmarks with various pragma combinations. The left Y-axis in Fig. 14 denotes hardware resource utilization in each design, while the right Y-axis shows the total execution time. Here, we use the AXI4 stream interface to communicate between the processor and the HA. The measured results demonstrate that applying optimization pragmas presents a performance/area trade-off. Furthermore, it is interesting to observe that the selected pragmas, if exploited carefully, can improve the accelerator performance within area budget. From Fig. 14, we note that the (p2_u64_a8) configuration of GEMM benchmark reduces the total execution time by about 78% compared to the (p0_u0_a0) configuration, but with increased FPGA resources. For the other configurations, we can also see a significant speed-up compared to the configurations without pragmas.

As shown in Figs. 13 and 14, optimizing the application using pragmas, such as loop pipelining, unrolling and array partitioning considerably reduces the total execution time compared to the results without pragmas. However, more logic resources are required. Therefore, it is clear that as the number of pragmas increases, the FPGA resource requirement and the cost of the HA implementation increase as well.

4.4. Estimation accuracy

To evaluate estimation accuracy of the proposed models, we calculate the average error between the measured (by RTL implementation) and estimated results:

$$AverageError = |Measured - Estimated|/Measured*100.$$

We use the power controller UCD9248 mounted on the evaluation board using Fusion Digital Power Designer [42] to understand how the power was consumed by the different hardware resources and validate the obtained power estimates. We use the equations presented in Section 3 to estimate the area occupation and the power consumption for the different benchmarks. The power consumption for each design point was also estimated using Xilinx XPower Analyzer [49] to generate the same configurations as the proposed power model and compare the obtained power results. We then compute the average estimation error for all the configurations.

In this work, we use first three benchmarks from Polybench Suite,

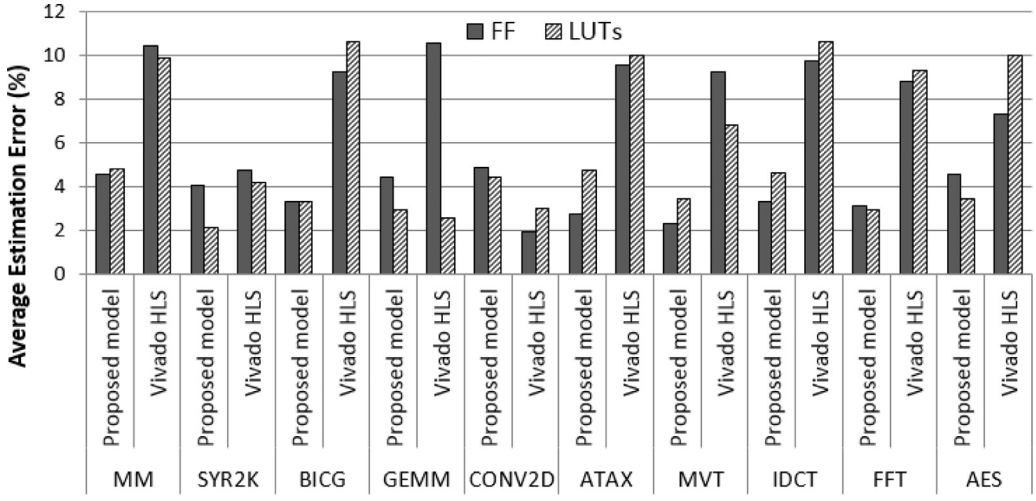
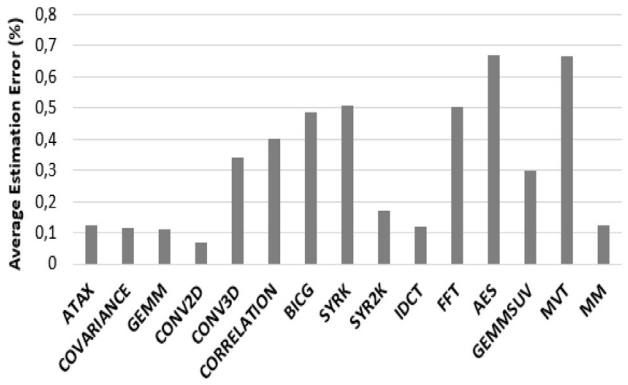
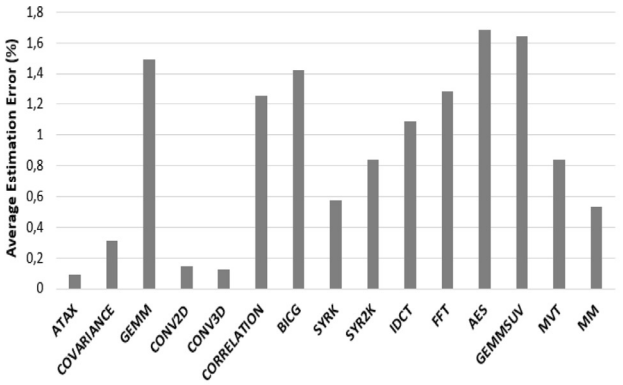


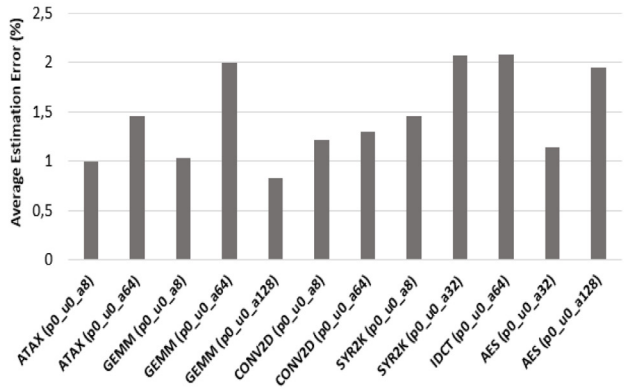
Fig. 15. Our proposed area model and Vivado HLS estimation vs. real hardware implementation with loop pipelining, loop unrolling and array partitioning.



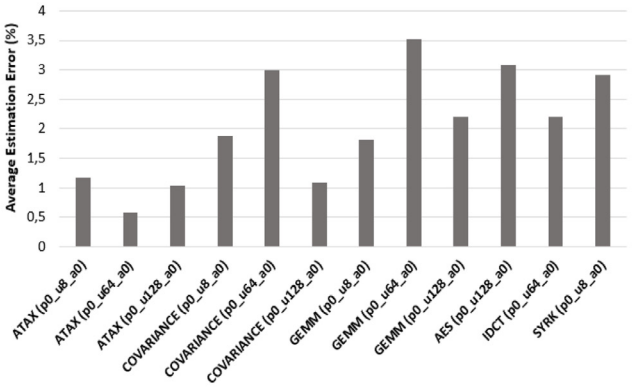
(a) Average estimation error (in%) with loop pipelining, unrolling and array partitioning pragmas



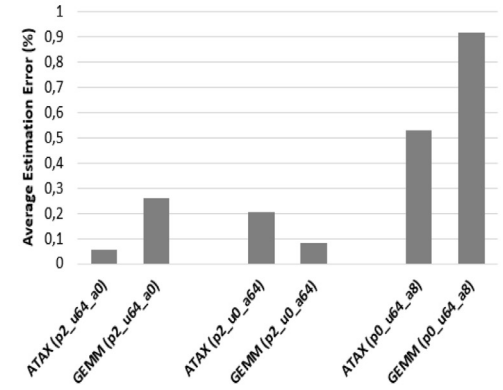
(b) Average estimation error (in%) with *only* loop pipelining pragma



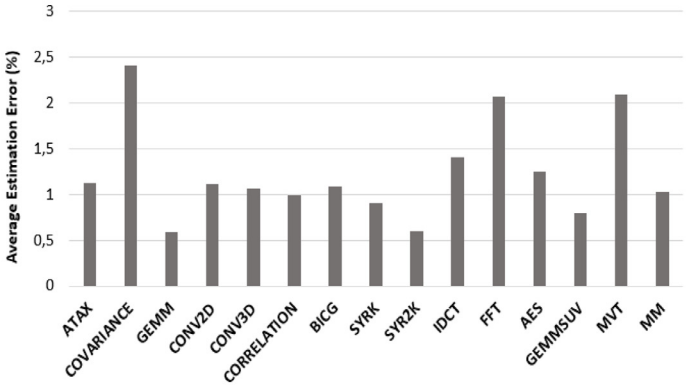
(c) Average estimation error (in%) with *only* array partitioning pragma



(d) Average estimation error (in%) with *only* loop unrolling pragma



(e) Average estimation error (in%) with various pragma combinations



(f) Average estimation error (in%) without pragmas

Fig. 17. Estimation Accuracy: Average estimation error (in%) between the proposed power model and the measured power with different pragma combinations.

which are *ATAX, MM* and *GEMM* as training benchmarks for the proposed power model. Then, we have applied the obtained equations on the other benchmarks from Poybench Suite such as *SYRK, BICG, CONV2D, CONV3D, COVARIANCE, CORRELATION*, etc. After that, in order to explore large design space and validate our proposed model, we have tested the proposed equations on other applications from CHStone Suite with different application domains such as IDCT (image processing), FFT (signal processing), AES (encryption).

Fig. 15 presents the experimental results of our proposed area estimation model and Vivado HLS compared to real hardware implementations for the different benchmarks. We can note that our proposed model results are very close to the ones obtained from the real hardware implementations considering the optimization pragmas (loop pipelining, loop unrolling and array partitioning). For Vivado HLS, the estimation error of FF resources achieves about 10% in GEMM benchmark.

Vivado HLS cannot provide accurate estimates for the required FPGA resources (FF, LUTs) due to its static analysis. This causes false dependencies between the operation nodes within the control data flow graphs. In this work, both BRAM and DSP resources estimated by our proposed area model and Vivado HLS match the real implementation results very closely. They are accurately estimated **(0% estimation error)** in the different benchmarks.

Fig. 16 compares the power results of the measured power, the

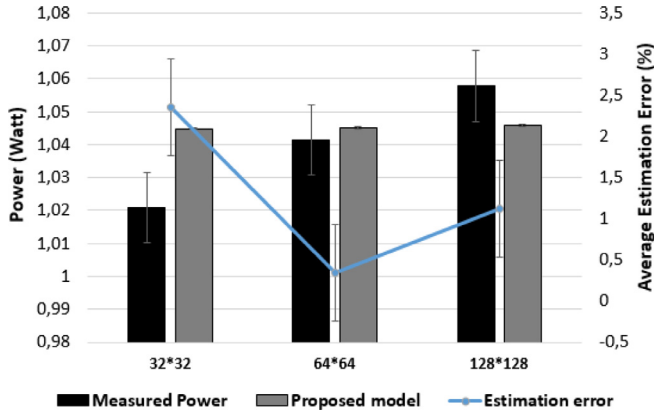


Fig. 18. Our proposed power model vs. measured power for ATAX (p0_u0_a0) configuration.

XPower Analyzer and the proposed power model, considering the loop pipelining pragma. From the obtained results, we note that the XPower Analyzer tool cannot provide accurate estimates for the power consumption due to its static analysis and lack of memory information (number of load/store instructions, etc.). Therefore, it introduces large inaccuracies in the estimated power consumption. As illustrated in Fig. 16, the experimental results show that the proposed power model can accurately estimate the power consumption of FPGA-based accelerators compared to exhaustive HLS-based techniques, including XPower Analyzer.

Fig. 17 depicts the average estimation error (in%) between our proposed power model and the measured power (real implementations) for each design point with/without optimization pragmas. As illustrated in Fig. 17, the proposed model serves well in the different configurations and results in an average estimation error less than 5% compared to the real hardware implementations. After applying the additional pragmas, the average error, shown in Fig. 17, ranges from 0.7% to 4% across all the benchmarks considered, which demonstrates high accuracy of our power model. Among these results, power estimation with loop unrolling pragma has the highest average error that primarily comes from the unpredictable behavior of FF resources while we calculate FF usage based on loop level and its unrolling factor. Although the average error is relatively high, power estimation still follows the measured power trend. This high estimation accuracy of our proposed model can be explained as follows: In contrast to Vivado HLS tool, our proposed framework leverages dynamic analysis approach and exploits run-time information to detect true dependences between operations. In addition, our proposed framework skips the time-consuming RTL generation, synthesis, and simulation process for different sizes of input data. Consequently, it explores rapidly various SoC configurations in the order of seconds to minutes, over a large multi-level parallelism design space using different pragma combinations (Table 5).

The average estimation error between the proposed power model and the measured power considering loop pipelining is depicted in Fig. 17b. This experiment clearly establishes the accuracy of our proposed power model. When only pipelining enabled, the average estimation error is less than 1.8% compared to the real hardware implementations.

Fig. 18 shows the accuracy of our proposed power estimation model of ATAX benchmark with various input data size (32, 64, 128). The power estimation results are less than 1.1% for different sizes of input data, which are very close to the real hardware implementation results using a target FPGA device. This estimation accuracy is explained by the fact of using a dynamic analysis in our model. This also obviates the need for generating RTL implementations to improve the estimation accuracy, thereby resulting in a rapid as well as a reliable design space exploration framework. In addition, it is worth to mention that the proposed models, presented in Section 3, are validated on other computational platforms, such as Xilinx Virtex7 platform (Table 6). Based on the experimental results, we found that our model obtains a good accuracy.

Table 6 shows the estimation accuracy of our proposed area model compared to Vivado HLS results using the Xilinx Virtex7 platform. To evaluate estimation accuracy of the proposed area model, we calculate the average error between the measured (Vivado HLS) and estimated results.

As shown in Table 6, our area model is able to estimate FF and LUT consumption with high accuracy across all the benchmarks considered, with an average area estimation error less than 7.6% and 7%, respectively. DSP and BRAM usage are also accurately estimated with 0% average error.

On the tested benchmarks, **static analysis** might add false loop-carried dependence between operations when considering array partitioning pragma and limit the available parallelism on accelerators. For example, when varying array partitioning factors, Vivado HLS, which relies on static analysis, conservatively adds false loop-carried dependences leading to higher II values dominated by these dependences. Our experiments on the considered benchmarks demonstrated that increasing memory bandwidth by applying array partitioning pragma, therefore, does not help to reduce the II in Vivado HLS. As a result, Vivado HLS is unable to improve the cycle counts by exploiting array partitioning pragma. To resolve this, in the experiments we enabled set directive dependence pragma in Vivado HLS to explicitly disable false dependences.

Table 6
Estimation accuracy of our proposed area model for different applications without pragmas using Virtex7 platform.

Benchmark	Proposed model/ Vivado HLS	FPGA resources				Average error (%)			
		BRAM	DSP	FF	LUT	BRAM	DSP	FF	LUT
MM	Proposed model	96	5	574	516	0	0	3.052	6.172
	Vivado HLS	96	5	557	486				
SYR2K	Proposed model	96	8	796	757	0	0	7.567	6.92
	Vivado HLS	96	8	740	708				
BICG	Proposed model	128	10	1055	946	0	0	2.943	5.345
	Vivado HLS	128	10	1087	898				
GEMM	Proposed model	96	5	620	590	0	0	6.896	2.966
	Vivado HLS	96	5	580	573				
ATAX	Proposed model	32	5	557	468	0	0	4.699	7.093
	Vivado HLS	32	5	532	437				
MVT	Proposed model	128	5	562	436	0	0	2.768	5.060
	Vivado HLS	128	5	578	415				

5. Conclusion

In this paper, we proposed a pre-RTL framework for area and power estimation based on high-level analytic models. The main objectives of this work is to help designers to quickly and accurately estimate the required FPGA resources and the total power consumption of different FPGA-based accelerators in an early design stage. Our framework focuses on applications containing compute-intensive kernels with nested loops. In the proposed approach, we do not consider heterogeneous MPSoC-FPGA architectures. This point could be considered as a possible extension of the work.

The comparison of our estimation results to RTL implementations confirmed the accuracy of our proposed framework with an average power estimation error less than 5%. In addition, the proposed models have the advantage of exploring and evaluating several configurations with different trade-offs through optimization pragmas. Loop pipelining, loop unrolling and array partitioning pragmas are those explored in this paper.

As extensions of this work, we project to extend our framework to support other applications having irregular memory access patterns. In this case, we should integrate machine learning techniques to accurately estimate the different design metrics of large and complex systems. We can also incorporate and explore new pragmas in order to improve the performance of the system, such as operation chaining, latency, etc.

References

[1] R. Jidin, D. Andrews, D. Niehaus, Implementing multithreaded system support for hybrid FPGA/CPU computational components, International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA), (2004), pp. 116–122.
[2] A. Putnam, D. Bennett, E. Dellinger, J. Mason, P. Sundararajan, S.J. Eggers, CHiMPS: a C-level compilation flow for hybrid CPU-FPGA architectures, International Conference on Field Programmable Logic and Applications (FPL), (2008), pp. 173–178.
[3] Zynq Ultrascale+ MPSoC, (http://www.xilinx.com/products/silicondevices/soc/zynqultrascale-mpsoc.html).
[4] Xilinx vivado design suite user guide: high-level synthesis, 2012.
[5] P. Coussy, et al., An introduction to high-level synthesis, IEEE Design Test of Computers, (2009).
[6] A. Canis, et al., LegUp: high-level synthesis for FPGA-based processor/accelerator systems, International Symposium on Field Programmable Gate Arrays (FPGA), (2011).
[7] N. Razvan, et al., A survey and evaluation of FPGA high-level synthesis tools, IEEE Trans. Comput.-Aided Des.Integr. Circuits Syst. (TCAD) 35 (10) (2015) 1591–1604.
[8] Xilinx vivado high-level synthesis, (http://www.xilinx.com/products/design-tools/vivado.html).
[9] Nvidia corporation, 2011, (Bring High-End Graphics to Handheld Devices).
[10] MPPA MANYCORE: a multicore processors family kalray, (http://www.kalray.eu/products/mppa-manycore).
[11] Xilinx Inc, (http://www.xilinx.com).
[12] M. Makni, M. Baklouti, S. Niar, M. Abid, Hardware resource estimation for heterogeneous FPGA-based SoCs, The 32nd ACM Symposium on Applied Computing (SAC), (2017).
[13] Xilinx AXI reference guide, 2012.
[14] G. Martin, Multi-processor SoC-based design methodologies using configurable and extensible processors, J. Signal Process. Syst. 53 (1) (2007) 113–127.
[15] Nvidia, (http://www.nvidia.com/object/tegra-4-processor.html).
[16] M. Makni, et al., A rapid data communication exploration tool for hybrid CPU-FPGA architectures, International Conference on Parallel, Distributed, and Network-Based Processing (PDP), (2017).
[17] Y.S. Shao, et al., Aladdin: a Pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures, International Symposium on Computer Architecture (ISCA), (2014).
[18] J. Villarreal, A. Park, W. Najjar, R. Halstead, Designing modular hardware accelerators in C with ROCCC 2.0, IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), (2010).
[19] Y Explorations, (http://www.yxi.com/products.php).
[20] Powerplay power analyzer support resources, (http://www.altera.com/support/support-resources/operation-and-testing/power/sof-qts-power.html).
[21] Altera corporation, 2009, (Nios II C2H Compiler User Guide).
[22] Catapult high-level synthesis, (http://www.mentor.com/hls-lp/catapult-high-level-synthesis).
[23] S. Bilavarn, et al., Design space pruning through early estimations of area/delay tradeoffs for FPGAImplementations, volume 25, 2006, pp. 1950–1968.
[24] A. Smith, et al., Area estimation and optimisation of FPGA routing fabrics, International Conference on Field Programmable Logic and Applications (FPL), (2009).
[25] H. Liu, et al., On learning-based methods for design space exploration with high-level synthesis, Design Automation Conference (DAC), (2013).
[26] A. Prost-Boucle, et al., A fast and autonomous HLS methodology for hardware accelerator generation under resource constraints, Euromicro Conference on Digital System Design (DSD), (2013).
[27] G. Zhong, V. Venkataramani, Y. Liang, T. Mitra, S. Niar, Design space exploration of multiple loops on FPGAs using high-level synthesis, IEEE International Conference on Computer Design (ICCD), (2014).
[28] G. Zhong, et al., Lin-analyzer: a high-level performance analysis tool for FPGA-based accelerators, Design Automation Conference (DAC), (2016).
[29] D. Sharma, V. Dumitriu, L. Kirischian, Architecture reconfiguration as a mechanism for sustainable performance of embedded systems in case of variations in available power, International Symposium on Applied Reconfigurable Computing (ARC), (2017).
[30] Altera SDK for OpenCL, 2015, (http://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/opencl-sdk/aocl_optimization_guide.pdf).
[31] D. Chen, D. Singh, Fractal video compression in OpenCL: an evaluation of CPUs, GPUs, and FPGAs as acceleration platforms, Asia and South Pacific Design Automation Conference (ASP-DAC), (2013).
[32] T. Kenter, M. Platzner, C. Plessl, M. Kauschke, Performance estimation for the exploration of CPU accelerator architectures, WARP - 5th Annual Workshop on Architectural Research Prototyping, (2010).
[33] C. Pham-Quoc, Z. Al-Ars, K. Bertels, Heterogeneous hardware accelerators interconnect: an overview, NASA/ESA Conference on Adaptive Hardware and Systems (AHS), (2013).
[34] B. Jovanovic, R. Jevtic, C. Carreras, Binary division power models for high-level power estimation of FPGA-based DSP circuits, volume 10, 2014, pp. 393–398.
[35] B. Jovanovic, R. Jevtic, C. Carreras, Triple-bit method for power estimation of nonlinear digital circuits in FPGAs, volume 46, 2010, pp. 903–905.
[36] L. Senn, E. Senn, C. Samoyeau, Modelling the power and energy consumption of NIOS II softcores on FPGA, IEEE International Conference on Cluster Computing Workshops, (2012).
[37] K. Ali, R.B. Atitallah, N. Fakhfakh, J.L. Dekeyser, Using hardware parallelism for reducing power consumption in video streaming applications, 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), (2015).
[38] B. Schafer, K. Wakabayashi, Divide and conquer high-level synthesis design space exploration, ACM Trans. Des. Autom.Electron. Syst. (TODAES) 17 (29) (2012).
[39] ARM, AMBA specification (Rev 2.0), (http://www.arm.com/).
[40] C. Lattner, et al., LLVM: a compilation framework for lifelong program analysis and transformation, International Symposium on Code Generation and Optimization (CGO), (2004).
[41] Xilinx Inc, Zynq-7000 All programmable SoC technical reference manual, 2014, (http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf).
[42] Texas instruments Inc, 2011, (http://www.ti.com/lit/ug/sluu490/sluu490.pdf).
[43] X. Inc, ZC702 Evaluation Board for the Zynq-7000 XC7Z020 All Programmable SoC User Guide, 2013.
[44] L. Pouchet, PolyBench/C 3.2, (http://web.cse.ohio-state.edu/pouchet/software/polybench).
[45] CHStone benchmark, (http://www.ertl.jp/chstone).
[46] 7 series FPGAs Data Sheet, 2014, (https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf).
[47] PMBus, (http://pmbus.org/index.php).
[48] Texas Instruments., Fusion digital power designer GUI for isolated power applications, 2014.
[49] Xilinx Xpower analyzer, (http://www.xilinx.com/products/design_tools/logic).