An Accurate Energy Model for Streaming Applications Mapped on MPSoC Platforms

Jelena Spasic and Todor Stefanov
Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands
Email: {jspasic, stefanov}@liacs.nl

***Abstract*—In this paper, we propose a very accurate energy model for streaming applications modeled as Polyhedral Process Networks (PPN) and mapped onto tile-based MPSoC platforms with distributed memory. The energy model is based on the well-defined properties of the PPN application model. To guarantee the accuracy of the energy model, values of important model parameters are obtained by real measurements. The proposed energy model is applicable to different types of processors and communication infrastructures within an MPSoC platform. The energy model is evaluated on FPGA-based MPSoC platforms against real measurements of energy consumption from the FPGA. The obtained energy estimates are highly accurate with an average error of 4% and a standard deviation of 3%. The average model evaluation time per design point takes 2.5 minutes for considered cases, which is very good given the high accuracy of the model.**

I. Introduction

The continuous increase in user demands and very fast technology improvement have led to more and more complex embedded systems. Nowadays, most embedded systems are based on Multi-Processor System-on-Chip (MPSoC) platforms. To exploit the parallel nature of these platforms, application behavior is usually specified using a certain parallel Model of Computation (MoC), in which the application is represented as parallel executing and communicating tasks. Finding an efficient application-to-platform mapping is the key issue for optimizing the energy consumption and performance of these systems. Since there are many possible application-to-platform mapping combinations forming a design space, this design space should be efficiently explored by using high-level system performance/energy models. Early in the design process of a system with certain performance/energy requirements, the design space is very large and decisions taken at higher level of abstraction have greater impact on the final design in terms of system performance and energy consumption. Therefore, high-level performance/energy models of a system should be accurate enough to steer the selection of optimal design points (under given constraints) in the right direction. Model accuracy is usually traded-off for modeling and evaluation effort. Especially accuracy of energy models is very important as the International Technology Roadmap for Semiconductors (ITRS) [1] reports that the power/energy consumption is the dominating constraint in the new generations of embedded systems.

In the embedded systems domain the research and results on performance modeling are very mature, while the research on system-level power/energy modeling and estimation has received attention only in recent years. So far, research on power/energy modeling has been mainly done for a single system component in isolation [2]–[12]. Only in a few cases, the power/energy consumption of the whole system has been modeled [13]–[18]. However, in most cases power/energy consumption due to the contention on shared resources is not considered. Moreover, in most cases, characterization and validation of the models have been done by using lower-level simulators or data-sheet values [2], [6], [7], [11]–[13], [15], [18], with usually questionable accuracy. Therefore, in order to find accurately an energy optimal application-to-platform mapping: 1) the energy model should describe the system as a whole and take into account the parallel nature of MPSoCs and possible energy consumption due to contention on shared resources; 2) the energy modeling and estimation should be done with high level of accuracy and efficiency.

For the above mentioned reasons, in this paper, we address the problem of accurate and efficient energy modeling of an application-to-platform mapping. We solve the problem in case when a streaming application is modeled using the Polyhedral Process Network (PPN) [19] MoC and mapped onto a tile-based MPSoC platform with distributed memory. Our energy model describes the system as a whole as well as it considers and models accurately the energy consumption due to data communication among the processors in a platform and the contention on non-contention-free communication infrastructures. The model is based on the well-defined properties of the PPN application model and the values of important energy model parameters are obtained by real measurements of energy consumption for the accuracy reason. The energy model is integrated in the existing Daedalus design flow [20], enabling a system designer to explore a large design space starting from a high-level description of the system behavior and having energy consumption as a primary design constraint.

A. Paper contributions

The major contribution of this paper is a novel energy model for streaming applications mapped onto MPSoC platforms that: 1) considers the system as a whole; 2) is very accurate; 3) is applicable to contention-free and non-contention-free communication infrastructures; 4) models total (static and dynamic) energy consumption; and 5) is applicable to different types of processors.

B. Related work

Research on power/energy modeling has been mainly done for individual system components in isolation – processors [2]–[7], memories [8], interconnections [9]–[12]. In contrast, our energy model models the system as a whole and thus enables more accurate energy estimation and exploration of different application-to-platform mappings.

978-1-4799-0103-6/13/$31.00 ©2013 IEEE

Only a few papers deal with power/energy modeling of the whole system. [13] analyzes power distribution among components in a homogeneous shared bus based MPSoC platform. There is no accuracy information for any model of a component in the system. In contrast, our energy model is more general in the sense that it can model platforms with contention-free and different configurations of non-contention-free communication infrastructures. In addition, we provide accuracy information concerning the obtained energy estimates.

[14] presents the performance and power modeling of multi-programmed multi-core systems. It is assumed that there is no data dependency between the processes running on a platform. The model is characterized and validated by real measurements. However, real applications usually consist of data dependent processes, and thus the energy consumption due to communication between the processes should be considered. In contrast to [14], our model considers the data dependency between the processes, and hence the energy consumption due to interprocessor communication is modeled.

[15] presents a multi-core power modeling and estimation tool flow which consists of two tools: *PowerMixer*^{IP}, an IP power model builder, and *PowerDepot*, a power estimation tool which generates and embeds power monitors into a SystemC simulation environment. Power model characterization and validation are done by using transistor-level and gate-level simulations. Authors report average power estimation error of 2% compared to gate-level simulations which accuracy is not known. In addition, the contention on shared resources is not discussed in [15]. In contrast, our energy model considers contention on different kinds of non-contention-free communication infrastructures with the energy estimates close to real energy measurements.

The FLPA power estimation methodology for MPSoCs is presented in [16]. The power consumption estimation consists of two parts: 1) power model development – a system is divided into functional blocks, and the power consumption is evaluated for selected activity parameters; 2) activity estimation and power calculation – a transaction level SystemC simulator and an Instruction Set Simulator (ISS) are used for detection of the activities. Models are characterized and validated by real measurements. Power modeling of shared resources and the contention on shared resources are not discussed in detail. In contrast, we give a general methodology for modeling the energy consumption for both contention-free and non-contention-free communication infrastructures. By considering the energy consumption due to the contention on non-contention-free communication infrastructures we achieve the energy estimates close to real energy measurements.

[17] proposes a top-down power and performance estimation methodology for MPSoCs. The system architecture is modeled by a set of resources – processors, memories, interconnects and dedicated hardware resources. Each resource is characterized by power and performance attributes. Power costs of the power attributes are extracted from measurements. There is no information about the accuracy of the proposed model and modeling of contention on non-contention-free communication infrastructures is not considered. In contrast, our energy model is more detailed, and consequently highly accurate with accuracy numbers obtained by comparison to real measurements. In addition, our work considers various kinds of communication infrastructures in the energy modeling.

In terms of application and platform models, the closest work to ours is [18]. An application is modeled by a Kahn Process Network (KPN) where every process has *read*, *execute* and *write* events. The proposed power modeling technique estimates the power consumption of an application-to-FPGA MPSoC mapping based on "event signatures". The "event signatures" for *execute*, *read* and *write* events are used together with a micro-architecture description, lower-level simulators and some additional parameters obtained from literature and through synthesis to calculate the power consumption of an application-to-MPSoC mapping. The model is validated by comparison to measurements. However, it is not clear how the "scaling factors" used for pre-calibration of the power models for interconnections and memories are obtained and what the relation is between these factors and application/MPSoC properties. This fact does not give high credibility to the accuracy of the model. Moreover, the authors assume that the data communication transactions performed by the KPN application model are not interleaved at the architecture level. In contrast, our energy model considers contention on shared resources and its parameters are extracted from measurements, which make the model very accurate. In addition, we do not use scaling factors and thus the accuracy of our model is highly credible.

The reminder of this paper is organized as follows: Section II introduces the considered application and platform models. The energy model formulation and the procedure to extract the parameters of the energy model are given in Section III. Section IV presents the results of empirical evaluation of the proposed energy model. Finally, Section V concludes the paper.

II. System model

Since our energy model is based on the well-defined properties of the PPN application model and the MPSoC platform model, in this section, we first present the PPN application model, followed by the MPSoC platform model we consider.

A. Application model

An application modeled as a PPN [19] is a directed graph $A = (\mathcal{P}, \mathcal{C})$ that consists of a set of processes $\mathcal{P} = \{P_1, P_2, ..., P_m\}$, which communicate with each other via a set of communication channels $\mathcal{C} = \{CH_1, CH_2, ..., CH_k\}$. Each channel in the PPN is a bounded FIFO and represents one direction of data communication between two processes. Here, a channel $CH_l = (P_i, P_j)$ represents a data dependency between processes P_i and P_j where P_i is the producer and P_j is the consumer process. A blocking read/write on an empty/full FIFO is the synchronization mechanism in the PPN MoC. An example of a PPN and the structure of its process P_3 is given in Fig. 1. Each process has a set of channels it reads from, a set of channels it writes to, and a function that represents a computation performed on input data that generates output data. A read/write from/to a channel is realized by blocking read/write primitives implemented in *software* (SW) or *hardware* (HW). Fig. 2 gives the structure of the read primitive implemented in software and hardware. In case of the SW read primitive, blocking FIFO access is implemented in software: *check for data*, see Fig. 2(a) lines 1, 3 and 4, *read data*, see Fig. 2(a) lines 5 and 6, and *release space*, see Fig. 2(a) lines 7 and 8. In case of the HW

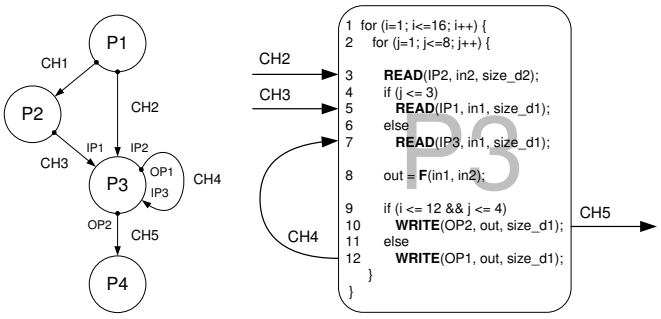

Fig. 1. Example of a PPN (a) and the structure of process $P3$ (b).

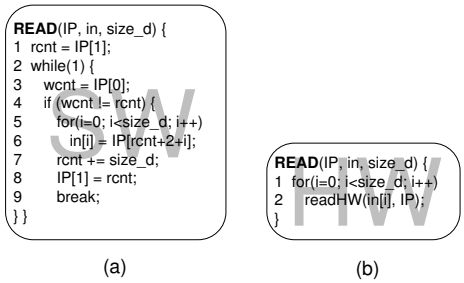

Fig. 2. The read primitive implemented in software (a) and hardware (b).

read primitive, blocking FIFO access is encapsulated in the $readHW$ function and realized in hardware, see Fig. 2(b). The execution of a PPN process is a set of iterations, called *process domain*. The process domain is represented using the polytope model [21]. In the example given in Fig. 1(b), the process domain of process P_3 is given as the polytope $D_{P3} = \{(i,j) \in \mathbb{Z}^2 \mid 1 \leq i \leq 16 \wedge 1 \leq j \leq 8\}$. Accessing an input/output port of the PPN process is represented as a subset of the process domain, called *input port domain*/*output port domain*. Process P_3 reads data from input ports $IP1$, $IP2$ and $IP3$. The input port domain of input port $IP2$ is equal to process domain D_{P3}, while the input port domain of port $IP1$ is given as $D_{IP1} = \{(i,j) \in \mathbb{Z}^2 \mid 1 \leq i \leq 16 \wedge 1 \leq j \leq 3\}$. Process P_3 writes data to output ports $OP1$ and $OP2$. Domain $D_{OP2} = \{(i,j) \in \mathbb{Z}^2 \mid 1 \leq i \leq 12 \wedge 1 \leq j \leq 4\}$ is the output port domain of port $OP2$.

By counting the integer points in the process domain polytope, we can determine the number of iterations each process function is executed. Similarly, by counting the integer points in the corresponding input/output port domain we can determine the number of read/write accesses for each channel of a process. Counting of the integer points in a polytope can be done automatically by using the *Barvinok* library in the `pn` compiler [19]. The counting ability of the PPN model is used in Section III-B for the computation of the so-called *N* energy model parameters N^{r_j}, N^{w_j} and N^{F_j}. In the example given in Fig. 1(b), by counting the integer points (i,j) in the process domain D_{P3} we can see that function F is executed 128 times and by counting the integer points in the port domains D_{IP1} and D_{OP2} we obtain that channel $CH3$ is read 48 times and channel $CH5$ is written 48 times.

B. Platform model

In this work, we consider tile-based MPSoC platforms with distributed memory. The generic architecture template of our platforms is shown in Fig. 3. A programmable processor with its local data and program memory, a timer and a bus bridge constitute a processing tile within the platform. Different processing tiles can have different types of processors. The communication infrastructure consists of a contention-free or non-contention-free communication component and distributed communication memories (every tile has its own communication memory). A contention-free communication component is a point-to-point (P2P) medium where every channel in the PPN application model has its own communication link. Non-contention-free communication components are mediums with shared communication links – a shared bus (ShB) or a crossbar switch (CB). Communication memories are assumed to be dual-port memories. This means that the communication memory can be accessed by its own processing tile and a remote processing tile at the same time. The processing tile produces data to its communication memory, locally accessing it through the local data bus, and consumes data from its own and/or other communication memories remotely through the communication component. Within our platforms, HW read/write primitives are used for P2P communication components, while SW read/write primitives can be used for both P2P and shared (CB, ShB) communication components.

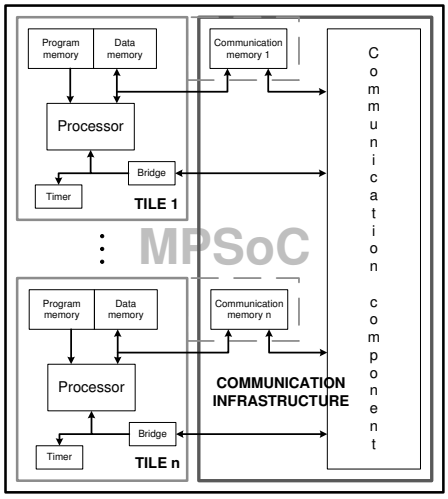

Fig. 3. The architecture template of MPSoC platforms.

More formally, a platform can be represented as a directed graph $P = (\pi, \mathcal{L})$, where $\pi = \{\pi_1, \pi_2, ..., \pi_n\}$ is a set of n processing tiles (homogeneous or heterogeneous) and $\mathcal{L} \subseteq \pi \times \pi$ is a set of physical communication links between the tiles.

C. Application-to-platform mapping

The mapping of an application $A = (\mathcal{P}, \mathcal{C})$ onto a platform $P = (\pi, \mathcal{L})$ can be expressed as a tuple $M = (\mathcal{P}^n, \mathcal{C}^n)$, where $\mathcal{P}^n = \{P_1^n, P_2^n, ..., P_n^n\}$ is an n-partition of set $\mathcal{P}$ defined in Section II-A, and $\mathcal{C}^n = \{CH_1^n, CH_2^n, ..., CH_k^n\}$ is a set of communication channels constructed from the set $\mathcal{C}$ defined in Section II-A. A subset P_i^n represents the set of processes mapped onto tile π_i. These processes produce data to the communication memory that is assigned to tile π_i. If the number of processes of a PPN is greater than the number of processing tiles in a platform, then some of the tiles execute more than one process. In this case, static schedule of processes is derived for every tile. This is done automatically by using the `pn` compiler [19]. Each channel $CH_l^n \in \mathcal{C}^n$ corresponds

to one channel $CH_l = (P_i, P_j) \in \mathcal{C}$ and is given by a tuple $(proc(P_i), proc(P_j))$, where $proc(P)$ represents the processor on which process P is mapped.

Recall that in our platforms FIFO channels reside in the communication memories and reading from channels is performed remotely through the communication component, while writing to channels is done locally by a processor through the local tile's bus on which the processor is the only master, see Section II-B. This means that writing is always contention-free, while reading is not because non-contention-free communication component may be used in the platform. In case of non-contention-free communication components, for each application-to-platform mapping, we define *Read contention* matrix $\mathbf{R} \in \mathbb{N}^{n \times n}$ as:

$$R_{ij} = \begin{cases} 1, & \exists CH_l^n = (\pi_i, \pi_j) \in \mathcal{C}^n \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

The matrix is used in Section III-B3 to analyze the influence of contention on the energy consumption of an application-to-platform mapping.

III. Energy model

The proposed energy model is used to estimate the energy consumption of a mapping of a streaming application modeled as described in Section II-A onto an MPSoC platform modeled as described in Section II-B. The energy model relies on the properties of the PPN application model and the platform model. The following subsections describe our energy model.

A. Model formulation

Without loss of generality and for the sake of clarity, we assume in the following that each process within an application is mapped to a different processing tile in a platform (i.e. the number of processes is equal to the number of processing tiles). In general, the proposed model is applicable to any application-to-platform mapping given that multiple processes of an application can be grouped and represented as a single process by finding a sequential schedule between the processes, as explained in Section II-C.

Since the PPN representation of an application is a set of concurrent processes, we can express the energy consumption of the application-to-platform mapping $E_{app \to pla}$ as the sum of energies consumed by processes E_{P_j}:

$$E_{app \to pla} = \sum_{j=1}^{n} E_{P_j} \quad (2)$$

A PPN process reads input data from (a part of) input channels, performs computation on input data and generates output data which is further written to (a part of) output channels (see Fig. 1(b)). Read and write accesses to channels are blocked if required data is not available or if there is no space for new data. Having this in mind, we can express the energy E_{P_j} as:

$$E_{P_j} = E_{RD_j} + E_{EXE_j} + E_{WR_j} + E_{BLK_j} + E_{CTRL_j} \quad (3)$$

where E_{RD_j} and E_{WR_j} are the energies consumed by reading from and writing to channels without blocking, respectively; E_{EXE_j} is the energy consumed by performing the computation in the process; E_{BLK_j} is the energy consumed while the process is blocked on read and write, and E_{CTRL_j} is the energy consumed by control structures in the process code. In the example given in Fig. 1(b), E_{CTRL_j} corresponds to the control structures in lines 1, 2, 4, 6, 9 and 11. Further, E_{RD_j} and E_{WR_j} can be expressed as:

$$E_{RD_j} = \sum_{r_j} N^{r_j} (E_{RD_j}^{r_j} + c \cdot E_c^{r_j}) \quad (4)$$

and

$$E_{WR_j} = \sum_{w_j} N^{w_j} \cdot E_{WR_j}^{w_j} \quad (5)$$

where r_j/w_j is the number of channels process P_j reads from/writes to; N^{r_j}/N^{w_j} is the number of times P_j accesses each read/write channel, and $E_{RD_j}^{r_j}/E_{WR_j}^{w_j}$ is the energy profile of one read/write from/to a channel. Recall that in our platforms writing to channels is local and reading from channels is remote (Section II-B). This means that reading from channels may go through a non-contention-free communication component and hence, energy consumed by reading from channels contains the contention dependent part $c \cdot E_c^{r_j}$. If the communication component is contention-free, c is 0, if it is non-contention-free, c is 1, while $E_c^{r_j}$ is the energy consumed while P_j is waiting for data from channel r_j when the communication component is non-contention-free. Similarly to $E_{RD_j}^{r_j}$ and $E_{WR_j}^{w_j}$, E_{EXE_j} becomes:

$$E_{EXE_j} = N^{F_j} \cdot E_{F_j} \quad (6)$$

where N^{F_j} is the number of times P_j executes its computation function F_j, and E_{F_j} is the energy profile of the function. The energy E_{BLK_j} consumed while the process is blocked can be divided to energy $E_{BLK_j}^{RD}$ consumed while the process is blocked on reading due to unavailable data and energy $E_{BLK_j}^{WR}$ consumed while the process is blocked on writing due to unavailable space. E_{BLK_j} can be expressed as:

$$E_{BLK_j} = E_{BLK_j}^{RD} + E_{BLK_j}^{WR} \quad (7)$$

The energies $E_{BLK_j}^{RD}$ and $E_{BLK_j}^{WR}$ can be further expressed as:

$$E_{BLK_j}^{RD} = \frac{T_{BLKRD_j}^{total}}{(T_{BLKRD_j}^{1} + c \cdot T_c^{1j})} \cdot (E_{BLK_j}^{rd} + c \cdot E_c^{1j}) \quad (8)$$

and

$$E_{BLK_j}^{WR} = \frac{T_{BLKWR_j}^{total}}{T_{BLKWR_j}^{1}} \cdot E_{BLK_j}^{wr} \quad (9)$$

where $T_{BLKRD_j}^{total}/T_{BLKWR_j}^{total}$ is the time spent in blocking on read/write by all the channels during the whole execution of the process P_j, $T_{BLKRD_j}^{1}/T_{BLKWR_j}^{1}$ is the time spent in one blocking on read/write by a channel, and $E_{BLK_j}^{rd}/E_{BLK_j}^{wr}$ is the energy profile of one blocking on read/write by a channel. During blocking on read, the process checks the write counter of the corresponding FIFO channel by reading its value through the communication component – see Fig. 2(a) line 3. If contention may occur (c is 1), checking the write counter on average will last longer with additional time T_c^{1j}, and the energy consumed by the checking will increase on average with E_c^{1j}.

The above mentioned energy profiles $E_{RD_j}^{r_j}$, $E_{WR_j}^{w_j}$, E_{F_j}, $E_{BLK_j}^{rd}$, $E_{BLK_j}^{wr}$ and E_{CTRL_j} associated with an application process are obtained by first converting the corresponding part of the process code to assembly equivalent, then counting the number of times N_i each assembly instruction i is executed in the corresponding assembly equivalent, and finally assigning the energy cost E_i to each instruction in the processor ISA. Therefore, each energy profile is the sum of the number of

times N_i each instruction i is executed in the corresponding assembly equivalent multiplied by the energy cost E_i of the given instruction i on a selected platform type. Hence, each of the above mentioned energy profiles can be represented as:

$$\sum_i N_i E_i \tag{10}$$

The contention dependent energy $E_c^{r_j}$ consumed by one read from a channel through a non-contention-free communication component can be expressed as:

$$E_c^{r_j} = \frac{T_{stall}^{r_j}}{T_{1stall}} \cdot E_{stall} \tag{11}$$

where $T_{stall}^{r_j}$ is the total estimated stall time during one read access on channel r_j through non-contention-free communication component, T_{1stall} is the latency of one stall, the ratio $T_{stall}^{r_j}/T_{1stall}$ is the estimated number of stalls on the communication component for one read access on channel r_j, and E_{stall} is the energy cost of one stall.

The contention dependent energy E_c^{1j} consumed by one checking of the write FIFO counter through a non-contention-free communication component can be expressed as:

$$E_c^{1j} = \frac{T_c^{1j}}{T_{1stall}} \cdot E_{stall} = \frac{T_{stall}^{1j}}{T_{1stall}} \cdot E_{stall} \tag{12}$$

where T_{stall}^{1j} is the total estimated stall time through non-contention-free communication component for one check for data availability and the ratio $T_{stall}^{1j}/T_{1stall}$ is the estimated number of stalls for one check for data availability.

B. Derivation of model parameters

From the model formulation in Section III-A we can see that the energy model has three types of parameters – *N* parameters such as N^{r_j}, N^{w_j}, N^{F_j}, N_i; *T* parameters such as $T_{BLKRD_j}^{total}$, $T_{BLKWR_j}^{total}$, $T_{BLKRD_j}^{1}$, $T_{BLKWR_j}^{1}$, $T_{stall}^{r_j}$, $T_{stall}^{1j}(T_c^{1j})$, T_{1stall}; and *E* parameters such as E_i, E_{stall}. This section explains how the value of each of the parameters is obtained. Parameters N^{r_j}, N^{w_j} and N^{F_j} are obtained by counting integer points in input, output and process domain polytopes of P_j, see Section II-A, which can be done automatically by using the *Barvinok* library in the `pn` compiler [19]. It is done only once per application and the obtained parameters can be used for any mapping of that application to any MPSoC platform. Parameter N_i is obtained by counting how many times an instruction from the processor ISA is executed in the corresponding assembly equivalent of the process code. This is obtained by using ISS simulators or some hardware tracing circuits and our profiler tool, see Section III-B2. It is done only once per application for a selected processor type. Parameters $T_{BLKRD_j}^{total}$ and $T_{BLKWR_j}^{total}$ are obtained from a cycle-accurate SystemC timing simulation of PPNs [22]. This SystemC simulation should be performed for each application-to-platform mapping because the blocking time, i.e. waiting for data/space, depends on the specific mapping of the processes of an application to the platform. Parameters $T_{BLKRD_j}^{1}$ and $T_{BLKWR_j}^{1}$ are obtained by using ISS simulators or some hardware tracing circuits. It is done only once for a selected processor type and for a selected implementation of the read/write primitives. Parameters $T_{stall}^{r_j}$ and T_{stall}^{1j} are obtained for each mapping, by performing the analysis explained in Section III-B3. Parameter T_{1stall} is obtained from data-sheets or from measurements. The energy cost E_i for each instruction i and energy cost E_{stall} for a stall are obtained from measurements – see Section III-B1, and this is done only once per platform type (processor type, communication infrastructure type, selected technology).

1) Extraction of the energy costs: In this section we will describe how the energy costs E_i for each instruction i and the energy cost E_{stall} for a stall are derived.

Since our platforms consist of processing tiles and communication infrastructure, the energy costs E_i and E_{stall} can be expressed as:

$$E_i = E_{i_{tile}} + E_{comm} = (P_{i_{tile}} + \frac{P_{comm}}{n}) l_i \tag{13}$$

and

$$E_{stall} = E_{stall_{tile}} + E_{comm} = (P_{stall_{tile}} + \frac{P_{comm}}{n}) l_{stall} \tag{14}$$

where $E_{i_{tile}}$ and $E_{stall_{tile}}$ are tile-dependent energy costs and E_{comm} is a communication infrastructure-dependent energy cost. The energy costs are obtained by multiplying the corresponding power costs $P_{i_{tile}}$, $P_{stall_{tile}}$, P_{comm} with the instruction latency l_i, and the stall latency l_{stall}. The power consumption $P_{i_{tile}}$ is the power consumed by an instruction i during its execution on a processing tile. The power cost $P_{stall_{tile}}$ is the power consumption of a tile when a stall occurs. P_{comm} is the power consumed by the communication infrastructure when there is no communication over the infrastructure, while n is the maximum number of tiles that the interconnect allows. The power consumption of communication is captured within $P_{i_{tile}}$ power cost for *load* and *store* instructions. These power costs are extracted from measurements.

There may be instructions with different latencies depending on cases they are used. An example is a conditional branch instruction which can be taken or not taken with different latencies for both cases. In this case, we consider an instruction as a set of instructions with finite number of elements equal to number of possible cases. We consider every instruction from that set as an individual instruction and assign a power cost to each of them.

For each instruction i we determine its power cost $P_{i_{tile}}$ by measuring the power consumption with minimum activity and maximum activity of the instruction. The final power cost is an average of the measured maximum and minimum power consumption. In order to measure the maximum and minimum power consumption, we create simple test codes with the instruction under test in a loop and run them on the tile. In the "minimum activity" case an instruction performs its action each time on the same operands, so there is no switching activity on processor core buses. In the "maximum activity" case an instruction performs its action each time on different operands such that switching activity on the buses is maximized. The power cost of a stall $P_{stall_{tile}}$ is obtained by measuring the power consumption of a system when stall occurs. The power cost P_{comm} is measured on a platform with maximum number of tiles n, which the corresponding interconnect allows, while there is no communication between the tiles. These estimations of energy costs are performed only once for the selected processor type and only once for the selected communication infrastructure.

2) Extraction of the energy profiles: In order to create the energy profiles $E_{RD_j}^{r_j}$, $E_{WR_j}^{w_j}$, E_{F_j}, $E_{BLK_j}^{rd}$, $E_{BLK_j}^{wr}$ and E_{CTRL_j} associated with an application process P_j, we should

first obtain the assembly instruction profiles of the corresponding parts of the process code. The instruction profile of a code consists of instruction counters which show how many times each instruction from a processor ISA is executed in the corresponding code. In case of branch instructions we also need the number of taken and the number of not-taken branches for each branch instruction. We need the execution trace of an application in order to obtain the needed instruction profiles. Since the PPN application consists of processes repeated a number of times, we do not need the instruction trace of the whole execution of an application and we only need the traces of each computation function, the read and write primitives for each channel and the control structures. Each computation function of an application is executed as many times as many different execution traces can occur for that function. The execution traces can be obtained by ISS simulators or by some hardware tracing circuits. The execution traces usually contain program counter values, instructions and can also contain some additional information (e.g. branch is taken or not, etc.). By analyzing program counter values we can determine if a branch is taken or not. Our profiler tool reads the execution traces of an application and creates the instruction profiles of the application. The final instruction profile of the computation function with many possible execution traces is the average profile, where counters of each instructions are averaged. Profiling of an application is done only once for the selected processor type and selected implementation of read/write primitives (HW or SW).

3) Analysis of communication contention: The derivation of the energy model parameters $T^{r_j}_{stall}$ and T^{1j}_{stall} related to non-contention-free communication components is explained in this section. Since our procedure analyzes the contention on remote tile-to-communication memory link, i.e. $\pi_j \leftarrow \pi_i$ link, we will use in the following the notation r_{ij} for a read channel of a tile π_j that reads from communication memory of a tile π_i.

The communication contention may occur if the communication component within the MPSoC platform is an arbitrated structure. In this work, we consider two types of non-contention-free communication components – a crossbar switch (CB) and a shared bus (ShB), where the CB and ShB interconnections have a round-robin arbitration policy.

The procedure to derive $T^{r_{ij}}_{stall}$ and T^{1j}_{stall} for CB communication component is given in Algorithm 1. Inputs of the algorithm are the contention matrix **R** defined in Section II-C, the number n of processing tiles in the platform, the size $s_{r_{ij}}$ of a data token transmitted through a channel r_{ij} and latencies of the interconnect read and write arbiters a_R, h_R, r_R, a_W, h_W, r_W. During one read and write access through the CB before the transferring of data the corresponding arbiters first arbitrate the requests for access, with associated arbitration latency a_R and a_W, and then ensure the communication link, with associated handshaking latency h_R and h_W. Additional latency r_R and r_W may occur on a master-slave link if there is a re-arbitration, i.e. when the requested slave unit is different from the last granted slave unit. The parameter $s_{r_{ij}}$ is obtained from the PPN model of an application, while arbiters' latencies a_R, h_R, r_R, a_W, h_R, r_W are obtained from measurements or data-sheets. The contention on a CB component may occur when at least two processes from at least two different tiles perform read operation to the same communication memory at the same time.

Algorithm 1 Procedure to derive $T^{r_{ij}}_{stall}$ and T^{1j}_{stall} for the CB

Require: **R**, n, $s_{r_{ij}}$, a_R, h_R, r_R, a_W, h_R, r_W

1: **for** $1 \le i \le n$ **do**
2: $c_i = 0$
3: **for** $1 \le j \le n$ **do**
4: $c_i = c_i + R_{ij}$
5: **end for**
6: **if** $c_i > 1$ **then**
7: $c_i = 1$
8: **else**
9: $c_i = 0$
10: **end if**
11: **end for**
12: **for** $1 \le j \le n$ **do**
13: $q_j = 0$
14: **for** $1 \le i \le n$ **do**
15: $q_j = q_j + R_{ij}$
16: **end for**
17: **if** $q_j > 1$ **then**
18: $q_j = 1$
19: **else**
20: $q_j = 0$
21: **end if**
22: **end for**
23: **for** $1 \le j \le n$ **do**
24: $l = 0$
25: **for** $1 \le i \le n$ **do**
26: **if** $R_{ij} = 1$ **then**
27: $L^{1bc}_{r_{ij}} = h_R$
28: **for** r_{ij} such that $\pi_i \rightarrow \pi_j$ **do**
29: $L^{bc}_{r_{ij}} = (2 + s_{r_{ij}})h_R + q_j * 0.5r_R + h_W + q_j * 0.5r_W$
30: $L^{wc}_{r_{ij}} = L^{bc}_{r_{ij}} + c_i * \sum_{k=1,k\neq j}^{n} R_{ik}((2+s_{r_{ij}})(h_R + a_R)) + q_k * 0.5r_R + h_W + a_W + q_k * 0.5r_W)$
31: **return** $T^{r_{ij}}_{stall} = (L^{bc}_{r_{ij}} + L^{wc}_{r_{ij}})/2$
32: $L^{1wc}_{r_{ij}} = L^{1bc}_{r_{ij}} + c_i * \sum_{k=1,k\neq j}^{n} R_{ik}(h_R + a_R)$
33: $L^{1}_{r_{ij}} = (L^{1bc}_{r_{ij}} + L^{1wc}_{r_{ij}})/2$
34: $l = l + 1$
35: **end for**
36: **end if**
37: **end for**
38: **return** $T^{1j}_{stall} = (\sum_{l} L^{1}_{r_{ij}})/l$
39: **end for**

Algorithm 1 gives the procedure to derive $T^{r_{ij}}_{stall}$ and T^{1j}_{stall} for the CB and it consists of three parts: 1) for each communication memory it is determined whether contention may occur – lines 1 to 11; 2) for each processing tile it is determined whether the re-arbitration may occur – lines 12 to 22 in the algorithm; and 3) the estimation of $T^{r_{ij}}_{stall}$ and T^{1j}_{stall} is performed at lines 23 to 39. Since the circular round-robin arbitration pointer is statistically located in the middle of the search space, we estimate $T^{r_{ij}}_{stall}$ at line 31 in Algorithm 1 as the average value of the best case stall time $L^{bc}_{r_{ij}}$, line 29, and the worst case stall time $L^{wc}_{r_{ij}}$, line 30. The best case stall time is when only one tile wants to read from a communication memory (so there is no arbitration latency a_R, a_W). The worst case stall times are calculated by analyzing if contention may happen, and if it may happen, then latencies h_R, h_W, a_R, a_W, r_R, r_W for all the tiles that compete for the same communication memory are summed up and added to the best case stall time. Recall that our platforms with shared communication infrastructures use SW read/write primitives – see Section II-B. During one read SW primitive on a channel r_{ij}, $s_{r_{ij}} + 2$ reads and one write are performed – see lines 1, 3, 6 and 8 in Fig. 2(a), where $s_{r_{ij}}$ corresponds to $size\_d$ in Fig. 2(a). Here, re-arbitration may happen only on the first read (out of $s_{r_{ij}} + 2$ reads) and on a write. The frequency of the re-arbitration depends on both the application structure and mapping. Here, if the re-arbitration may happen we assume

Algorithm 2 Procedure to derive $T_{stall}^{r_{ij}}$ and T_{stall}^{1j} for the ShB

Require: **R**, n, $s_{r_{ij}}$, a, h

1: $n\_r = 0$
2: **for** $1 \leq j \leq n$ **do**
3: $\quad y\_r_j = 0$
4: $\quad$ **for** $1 \leq i \leq n$ **do**
5: $\quad\quad y\_r_j = y\_r_j + R_{ij}$
6: $\quad$ **end for**
7: $\quad$ **if** $y\_r_j = 0$ **then**
8: $\quad\quad n\_r = n\_r + 1$
9: $\quad$ **end if**
10: **end for**
11: **for** $1 \leq j \leq n$ **do**
12: $\quad$ **for** $1 \leq i \leq n$ **do**
13: $\quad\quad$ **if** $R_{ij} = 1$ **then**
14: $\quad\quad\quad$ **for** r_{ij} such that $\pi_i \rightarrow \pi_j$ **do**
15: $\quad\quad\quad\quad L_{r_{ij}}^{bc} = (3 + s_{r_{ij}})h$
16: $\quad\quad\quad\quad L_{r_{ij}}^{wc} = L_{r_{ij}}^{bc} + (n - n\_r - 1)(3 + s_{r_{ij}})(h + a)$
17: $\quad\quad\quad\quad$ **return** $T_{stall}^{r_{ij}} = (L_{r_{ij}}^{bc} + L_{r_{ij}}^{wc})/2$
18: $\quad\quad\quad$ **end for**
19: $\quad\quad$ **end if**
20: $\quad$ **end for**
21: **end for**
22: $L^{1bc} = h$
23: $L^{1wc} = L^{1bc} + (n - n\_r - 1)(h + a)$
24: **return** $T_{stall}^{1j} = (L_{ij}^{1bc} + L_{ij}^{1wc})/2$

that the re-arbitration on a read access to the channel happens every second time on the first read and every second time on a write, i.e. we multiply r_R and r_W by 0.5 in lines 29 and 30 in Algorithm 1. In the case of data checking, there is no possibility for re-arbitration because waiting for data represents the reading of the write counter (second read within the SW read primitive). Since from the SystemC timing simulation we obtain information about the blocking time on a tile basis, for estimation of T_{stall}^{1j}, at line 38, we sum up the average values $L_{r_{ij}}^{1}$ of each channel that a tile accesses for reading and divide the result by the number of the accessed channels for reading by that tile.

Let us now analyze the ShB case. The contention may happen when at least two processes from at least two different tiles perform read operation at the same time to any of the communication memories in the platform. The procedure to derive $T_{stall}^{r_{ij}}$ and T_{stall}^{1j} for ShB is given in Algorithm 2. The input parameters are similar to the CB case with a difference that here we have only one arbiter. First we determine the number of tiles n_r that do not read from any communication memory, lines 1 to 10 in Algorithm 2. Then in the following lines, we estimate $T_{stall}^{r_{ij}}$ and T_{stall}^{1j}, line 17, 24, as an average of the best case stall time $L_{r_{ij}}^{bc}$, L^{1bc}, line 15, 22, and the worst case stall time $L_{r_{ij}}^{wc}$, L^{1wc}, line 16, 23. In the best case, only one tile wants to read from a communication memory. By computing how many tiles $(n - n\_r)$ read from any of the communication memories, we determine how long the tile may wait in the worst case.

IV. Evaluation of the model

We evaluate our energy model, proposed in Section III, by showing its accuracy considering various application-to-platform mappings. The obtained energy estimates are compared to real energy measurements obtained from real implementations of the considered systems, i.e., applications, platforms and mappings. We show that the proposed energy model is highly accurate for contention-free and different kinds of non-contention-free communication components, different applications and mappings, and different number of processing tiles in a platform.

The proposed energy model is evaluated on MPSoC systems prototyped on the Virtex-6 FPGA board ML605. Since the MicroBlaze [23] processor is the only available processor type on Virtex-6, we use MPSoC platforms with different number of MicroBlaze based tiles and with the AXI-4 [24] interconnect as a non-contention-free communication component, and a P2P interconnect as a contention-free communication component. We use the AXI interconnect configured in CB and ShB modes with a round-robin arbitration policy. The energy model is evaluated for two applications with SW read/write primitives. The first application is the Sobel edge-detection filter and the second application is the M-JPEG video encoder. The PPN model for the Sobel consists of 5 lightweight processes in terms of computation and 15 channels, thus the Sobel application is data communication-dominant which introduces a lot of contention on the CB and ShB. The M-JPEG PPN model consists of 6 processes and 5 channels with much higher computation/communication ratio, and hence the M-JPEG is a computation-dominant application. Since the maximum number of processes among these two applications is 6, we perform energy estimates for platforms with 2 to 6 processing tiles in a platform. The corresponding power consumptions of application-to-platform mappings are measured by using the ML605 on-board power monitoring device and an additional MicroBlaze processor which reads the corresponding power measurements from the monitoring device. Instruction traces for the applications are obtained by monitoring the Trace interface of a MicroBlaze processor. All the platforms run at a frequency of 100 MHz.

Applying our model, described in Section III, we estimate the energy consumption for each application-to-platform mapping, specified in the first column of Table I, for three types of communication infrastructures – the CB, ShB and P2P. In the first column, each mapping is denoted as $app\_n_{tiles}\_m_{map}$, where app is the application, n_{tiles} is the number of tiles in the platform, and m_{map} is the index of a mapping (as an application can be mapped onto a platform in many possible ways). The E_m columns contain the reference values of energy consumption of application-to-platform mappings, obtained by real measurements. The E_e columns contain the energy estimates of the same application-to-platform mappings obtained by using our energy model. The e_{rr} column for each type of interconnect gives the energy estimation error calculated as $e_{rr} = (E_e - E_m)/E_m \cdot 100\%$. It can be seen from Table I that our energy model is highly accurate for all three types of interconnects, with an average energy estimation error of 4.34% and a standard deviation of 3.35% among all the interconnection types.

In order to analyze the influence of the communication contention on energy consumption of an application-to-platform mapping, we perform the energy estimation for each application-to-platform mapping with CB and ShB interconnects without considering the contention in the energy model. The results are given in Table II. By comparing Table I and Table II we can see first that if the contention is not considered, the energy of a mapping is always underestimated, and second that the energy estimates are less accurate than the estimates with considering the contention in the energy model. Therefore, in the proposed energy model special attention is paid to modeling the contention on communication infrastructures.

From the results shown in Table I it is clear that our energy model is very accurate. Now we would like to discuss

TABLE I. ACCURACY OF THE ENERGY MODEL FOR CB, SHB AND P2P MPSOC PLATFORMS

$app \rightarrow pla$	CB			ShB			$P2P$		
	E_m $[mWs]$	E_e $[mWs]$	e_{rr} [%]	E_m $[mWs]$	E_e $[mWs]$	e_{rr} [%]	E_m $[mWs]$	E_e $[mWs]$	e_{rr} [%]
Sobel_2_m1	59.9	61.66	+2.94	54.71	58.18	+6.34	53.95	52.34	-2.98
M-JPEG_2_m1	48.82	51.96	+6.43	49.56	51.99	+4.9	58.82	56.98	-3.13
Sobel_3_m1	82.12	73.32	-10.72	68.75	73.67	+7.16	74.43	73.51	-1.24
Sobel_3_m2	69.74	66.63	-4.46	60.98	62.13	+1.89	49.62	50.42	+1.61
M-JPEG_3_m1	44.1	42.73	-3.1	40.5	42.19	+4.17	43.64	44.39	+1.72
M-JPEG_3_m2	76.74	69.95	-8.85	68.84	67.58	-1.83	86.56	79.67	-7.96
Sobel_4_m1	58.32	58.7	+0.66	52.18	56.86	+8.97	68.07	68.06	-0.01
M-JPEG_4_m1	96.72	95.05	-1.73	93.8	93.69	-0.12	107.97	103.68	-3.97
Sobel_5_m1	71.5	71.03	-0.65	68.78	77.46	+12.62	79.52	85.87	+7.99
M-JPEG_5_m1	125.63	121.65	-3.17	127.75	119.31	-6.61	137.94	126.84	-8.05
M-JPEG_6_m1	77.4	77.15	-0.32	74.7	75.27	+0.76	84.42	79.32	-6.04

TABLE II. ACCURACY OF THE ENERGY ESTIMATION WHEN CONTENTION IS NOT CONSIDERED IN THE MODEL

$app \rightarrow pla$	CB			ShB		
	E_m $[mWs]$	E_e $[mWs]$	e_{rr} [%]	E_m $[mWs]$	E_e $[mWs]$	e_{rr} [%]
Sobel_2_m1	59.9	47.2	-21.2	54.71	46.76	-14.53
M-JPEG_2_m1	48.82	44.54	-8.77	49.56	45.5	-8.19
Sobel_3_m1	82.12	53.64	-34.68	68.75	55.05	-19.93
Sobel_3_m2	69.74	54.78	-21.45	60.98	53.04	-13.02
M-JPEG_3_m1	44.1	38.65	-12.36	40.5	38.23	-5.6
M-JPEG_3_m2	76.74	57.92	-24.53	68.84	54.82	-20.37
Sobel_4_m1	58.32	46.89	-19.6	52.18	45.56	-12.69
M-JPEG_4_m1	96.72	82.27	-14.94	93.8	81.17	-13.46
Sobel_5_m1	71.5	54.86	-23.27	68.78	58.15	-15.46
M-JPEG_5_m1	125.63	109.39	-12.93	127.75	106.79	-16.41
M-JPEG_6_m1	77.4	71.51	-7.62	74.7	68.24	-8.65

the efficiency of our model in terms of the time required to estimate the energy consumption of a single application-to-platform mapping. For every mapping, listed in the first column of Table I, we measure the time needed for the energy estimation. The energy estimation is conducted on a Lenovo T520 laptop with Intel i7-2620M processor running at 2.7 GHz and 4 GB of RAM. The average model evaluation time for a mapping is 2.5 minutes, where a few milliseconds are needed for evaluation of the formulas in Section III-A and derivation of $T_{stall}^{r_{ij}}$ and T_{stall}^{1j} parameters, and the rest of the time is spent on getting $T_{BLKRD_j}^{total}$ and $T_{BLKWR_j}^{total}$ parameters. Deriving of the other model parameters is not considered in this time because they are derived only once at the beginning when the model is calibrated and they are independent of mapping. The efficiency of the proposed energy model is very good given its high accuracy. Note that the majority of the evaluation time (99%) is spent in SystemC cycle accurate simulation which simulates 12000 cycles per second on average. We run cycle accurate SystemC simulation for each mapping in order to obtain very accurate $T_{BLKRD_j}^{total}$ and $T_{BLKWR_j}^{total}$. We need as accurate estimation of these blocking times as possible in order to have accurate energy estimates because these blocking times are significant part of the total execution time of an application, and hence the energy consumed in blocking could be also significant part of the total energy consumed by a mapping.

V. CONCLUSION

We proposed an accurate energy model for streaming applications modeled using the PPN model and mapped onto MPSoC platforms. Special attention in our model is paid to the contention on non-contention-free communication infrastructures which is important to estimate accurately the energy consumption of a mapping. Experimental results on two applications with very different computation and communication characteristics mapped onto MPSoC platforms with different communication infrastructures showed that the proposed modeling and estimation methodology is highly accurate for different kinds of applications, different kinds of communication infrastructures within MPSoC platforms and various application-to-platform mappings. On average, the energy estimation error is 4% with a standard deviation of 3% in comparison to real energy measurements for all considered communication infrastructures. The average model evaluation time of 2.5 min per single design point for considered cases is very good given the high accuracy of the proposed energy model.

REFERENCES

[1] "The International Technology Roadmap for Semiconductors (ITRS), System Drivers," 2011, available on: http://www.itrs.net.

[2] G. Qu *et al.*, "Function-level power estimation methodology for microprocessors," in *Proc. of DAC*, 2000, pp. 810–813.

[3] E. Senn *et al.*, "Functional level power analysis: An efficient approach for modeling the power consumption of complex processors," in *Proc. of DATE - Volume 1*, 2004.

[4] A. Sinha and A. P. Chandrakasan, "Jouletrack - a web based tool for software energy profiling," in *Proc. of DAC*, 2001, pp. 220–225.

[5] A. Varma, *et al.*, "Accurate and fast system-level power modeling: An xscale-based case study," *ACM TECS*, vol. 7, no. 25, April 2008.

[6] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proc. of ISCA*, 2000, pp. 83–94.

[7] W. Ye *et al.*, "The design and use of simplepower: A cycle-accurate energy estimation tool," in *Proc. of DAC*, 2000, pp. 340–345.

[8] "HP Labs, CACTI," http://www.hpl.hp.com/research/cacti.

[9] S. Pasricha *et al.*, "Capps: A framework for power-performance tradeoffs in bus-matrix-based on-chip communication architecture synthesis," *IEEE Trans. on VLSI Systems*, vol. 18, pp. 209–221, February 2010.

[10] A. Bona, V. Zaccaria, and R. Zafalon, "System level power modeling and simulation of high-end industrial network-on-chip," in *Proc. of DATE - Volume 3*, 2004.

[11] A. B. Kahng *et al.*, "Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *Proc. of DATE*, 2009, pp. 423–428.

[12] N. Banerjee, P. Vellanki, and K. Chatha, "A power and performance model for network-on-chip architectures," in *Proc. of DATE - Volume 2*, 2004.

[13] M. Loghi, M. Poncino, and L. Benini, "Cycle-accurate power analysis for multiprocessor systems-on-a-chip," in *Proc. of ACM Great Lakes symposium on VLSI*, 2004, pp. 406–410.

[14] X. Chen *et al.*, "Performance and power modeling in a multi-programmed multi-core environment," in *Proc. of DAC*, 2010, pp. 813–818.

[15] C. W. Hsu *et al.*, "Powerdepot: Integrating ip-based power modeling with esl power analysis for multi-core soc designs," in *Proc. of DAC*, 2011, pp. 47–52.

[16] S. Rethinagiri *et al.*, "Hybrid system level power consumption estimation for fpga-based mpsoc," in *Proc. of IEEE ICCD*, 2011, pp. 239–246.

[17] M. Streubuhr, *et al.*, "Esl power and performance estimation for heterogeneous mpsocs using systemc," in *Proc. of the Forum on Specification, Verification and Design Languages*, 2011, pp. 1–8.

[18] R. Piscitelli and A. D. Pimentel, "A signature-based power model for mpsoc on fpga," *VLSI Design J.*, vol. 2012, no. 6, January 2012.

[19] S. Verdoolaege, H. Nikolov, and T. Stefanov, "pn: a tool for improved derivation of process networks," *EURASIP J.*, vol. 2007, pp. 19–19, January 2007.

[20] M. Thompson, *et al.*, "A framework for rapid system-level exploration, synthesis, and programming of multimedia mp-socs," in *Proc. of CODES+ISSS*, 2007, pp. 9–14.

[21] P. Feautrier, "Automatic parallelization in the polytope model," in *The Data Parallel Programming Model*. Springer-Verlag, 1996.

[22] S. van Haastregt, E. Halm, and B. Kienhuis, "Cost modeling and cycle-accurate co-simulation of heterogeneous multiprocessor systems," in *Proc.of DATE*, 2010, pp. 1297–1300.

[23] "Xilinx Inc., MicroBlaze Soft Processor Core," http://www.xilinx.com.

[24] "ARM Ltd., AMBA AXI Protocol - Version: 2.0 : Specification, 2010," http://www.arm.com.