# Improved hard real-time scheduling and transformations for embedded Streaming Applications

Spasic, J.

**Author**: Spasic, J.
**Title**: Improved hard real-time scheduling and transformations for embedded Streaming Applications
**Issue Date**: 2017-11-14

# Chapter 6

# An Accurate Energy Modeling of Streaming Systems

**Jelena Spasic** and Todor Stefanov, "An Accurate Energy Model for Streaming Applications Mapped on MPSoC Platforms", *In Proceedings of the IEEE International Conference on Embedded Computer Systems: Architectures, MOdeling, and Simulation (IC-SAMOS'13)*, pp. 205–212, Samos, Greece, July 15-18, 2013.

THE solution to the problem of accurate energy modeling of an application-to-MPSoC mapping, that is, **Problem 4** introduced in Section 1.3, is presented in this chapter.

The investigated research problem is further described in Section 6.1. It is followed by a summary of our contributions in Section 6.2. The related work is addressed in Section 6.3. The considered system model is described in Section 6.4. The energy model formulation and the procedure to extract the parameters of the energy model are given in Section 6.5. The model is experimentally evaluated in Section 6.6. The concluding discussion is given in Section 6.7.

## 6.1 Problem Statement

As discussed in Sections 1.2.2 and 1.3, finding an efficient application-to-platform mapping is the key issue for optimizing the energy consumption and performance of streaming MPSoC 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) [Int11] 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 [QKUP00,LJSM04,SC01,VJD+07,BTM00,YVKI00,CAC, PPKD10, BZZ04, KLPS09, BVC04]. Only in a few cases, the power/energy consumption of the whole system has been modeled [LPB04, C+10, HLF+11, RAN+11,SRH+11,PP12]. 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 [QKUP00, BTM00, YVKI00, KLPS09, BVC04, LPB04, HLF+11, PP12], introducing additional inaccuracy in the model. 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, we address the problem of accurate and efficient energy modeling of an application-to-platform mapping when a streaming application is modeled using the Polyhedral Process Network (PPN) [VNS07] MoC and mapped onto a tile-based MPSoC platform with distributed memory.

## 6.2 Contributions

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. It models the total (static and dynamic) energy consumption and is applicable to different types of processors. The energy model is integrated in the existing Daedalus design flow [TNS$^+$07], 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.

## 6.3 Related Work

Research on power/energy modeling has been mainly done for individual system components in isolation – processors [QKUP00,LJSM04,SC01,VJD$^+$07, BTM00,YVKI00], memories [CAC], interconnections [PPKD10,BZZ04,KLPS09, BVC04]. 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.

Only a few works deal with power/energy modeling of the whole system. [LPB04] analyzes power distribution among components in a homogeneous shared bus based MPSoC platform. However, 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.

[C$^+$10] presents the performance and power modeling of multi-programmed multi-core systems. In this work, it is assumed that there is no data dependency between the processes running on a platform. The model is characterized and validated by real measurements. However, real applications usually consist of data dependent processes, and thus the energy consumption due to communication between the processes should be considered. In contrast to [C$^+$10], our model considers the data dependency between the processes, and hence the energy consumption due to interprocessor communication is modeled.

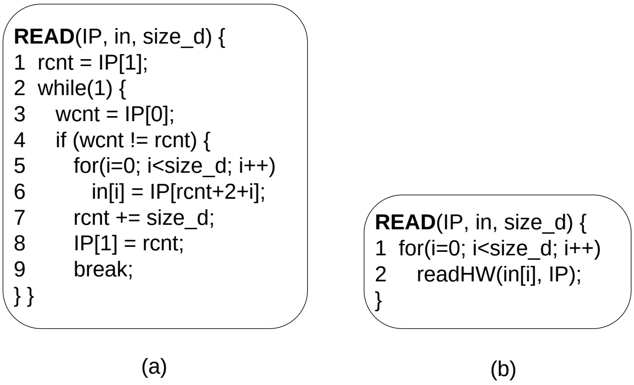
[HLF$^+$11] presents a multi-core power modeling and estimation tool flow which consists of two tools: *PowerMixer$^{IP}$*, an IP power model builder, and *PowerDepot*, a power estimation tool which generates and embeds power monitors into a SystemC simulation environment. Power model characterization and validation are done by using transistor-level and gate-level simulations.

The authors report an 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 [HLF$^+$11]. In contrast, our energy model considers the contention on different kinds of non-contention-free communication structures with the energy estimates close to real energy measurements.

The FLPA power estimation methodology for MPSoCs is presented in [RAN$^+$11]. 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 structures we achieve energy estimates close to real energy measurements.

[SRH$^+$11] 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 with real measurements. In addition, our work considers various contention-free and non-contention-free communication infrastructures in the energy modeling.

In terms of application and platform models, the closest work to ours is [PP12]. An application is modeled as 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



```
READ(IP, in, size_d) {
  rcnt = IP[1];
  while(1) {
    wcnt = IP[0];
    if (wcnt != rcnt) {
      for(i=0; i<size_d; i++)
        in[i] = IP[rcnt+2+i];
      rcnt += size_d;
      IP[1] = rcnt;
      break;
} }
```

(a)

```
READ(IP, in, size_d) {
  for(i=0; i<size_d; i++)
    readHW(in[i], IP);
}
```

(b)

**Figure 6.1:** *The read primitive implemented in software (a) and hardware (b).*

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.

## 6.4 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 give more details on the PPN application model presented in Chapter 2, and then describe the MPSoC platform model we consider in this chapter.

### 6.4.1 Application Model

An example of a PPN and the structure of its process $P_3$ is given in Figure 2.2. 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). Figure 6.1 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 Figure 6.1(a) lines 1, 3 and 4, *read data*, see Figure 6.1(a) lines 5 and 6, and *release space*, see Figure 6.1(a) lines 7 and 8. In case of the HW read primitive, blocking FIFO access is encapsulated in the *readHW* function and realized in hardware, see Figure 6.1(b). As explained in Chapter 2, the execution of a PPN process represents a process domain, described by using the polytope model [Fea96b]. In addition, accessing input and output ports of the PPN process is represented by the corresponding input and output port domains which are subsets of the process domain. 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 [VNS07]. The counting ability of the PPN model is used in Section 6.5.2 for the computation of the so-called $N$ energy model parameters $N^{r_k}$, $N^{w_k}$ and $N^{F_k}$. In the example given in Figure 2.2(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.

### 6.4.2 Platform Model

In this work, we consider tile-based MPSoC platforms with distributed memory. The generic architecture template of our platforms is shown in Figure 6.2. 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, lo-

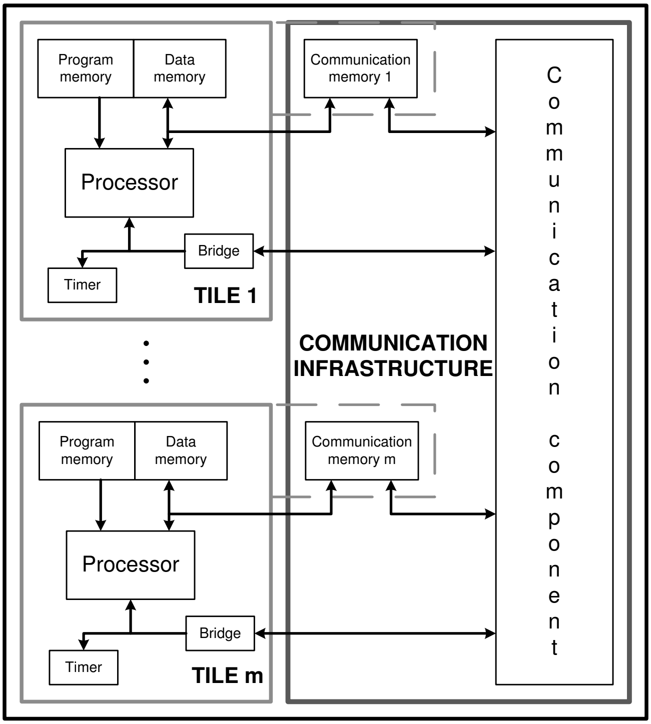


**Figure 6.2:** *The architecture template of MPSoC platforms.*

cally 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.

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

### 6.4.3 Application-to-Platform Mapping

The mapping of an application modeled as a PPN $G = (\mathcal{P}, \mathcal{C})$ onto a platform $\Pi = (\pi, CL)$ can be expressed as a tuple $M = (\mathcal{P}^m, \mathcal{C}^m)$, where $\mathcal{P}^m = \{P_1^m, P_2^m, ..., P_m^m\}$ is an $m$-partition of set $\mathcal{P}$ of the processes, and $\mathcal{C}^m = \{CH_1^m, CH_2^m, ..., CH_m^m\}$ is a set of communication channels constructed from the set $\mathcal{C}$ of all channels in a PPN. A subset $P_i^m$ represents the set of processes mapped onto tile $\pi_i$. These processes produce data to the communication memory that is assigned to tile $\pi_i$. 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 [VNS07]. Each channel $CH_l^m \in \mathcal{C}^m$ 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 $proc(P_i) = \pi_i$ represents the processor $\pi_i$ on which process $P_i$ 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 6.4.2. 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 $\mathbf{R} \in \mathbb{N}^{m \times m}$ as:

$$R_{ij} = \begin{cases} 1, & \exists CH_l^m = (\pi_i, \pi_j) \in \mathcal{C}^m \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

The matrix is used in Section 6.5.2 to analyze the influence of contention on the energy consumption of an application-to-platform mapping.

## 6.5 Energy Model

The proposed energy model is used to estimate the energy consumption of a mapping of a streaming application modeled as a PPN described in Sections 2.1.2 and 6.4.1 onto an MPSoC platform modeled as described in Section 6.4.2. The energy model relies on the properties of the PPN application model and the platform model. The following subsections describe our energy model.

### 6.5.1 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, that is, 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 6.4.3.

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 $E_{app \rightarrow pla}$ as the sum of energies consumed by processes $E_{P_k}$:

$$E_{app \rightarrow pla} = \sum_{k=1}^{n} E_{P_k}. \tag{6.2}$$

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 Figure 2.2(b)). Read and write accesses to channels are blocked if the required data is not available or if there is no space for new data. Having this in mind, we can express the energy consumed by a process $E_{P_k}$ as:

$$E_{P_k} = E_{RD_k} + E_{EXE_k} + E_{WR_k} + E_{BLK_k} + E_{CTRL_k}, \tag{6.3}$$

where $E_{RD_k}$ and $E_{WR_k}$ are the energies consumed by reading from and writing to channels without blocking, respectively; $E_{EXE_k}$ is the energy consumed by performing the computation in the process; $E_{BLK_k}$ is the energy consumed while the process is blocked on read and write, and $E_{CTRL_k}$ is the energy consumed by control structures in the process code. In the example given in Figure 2.2(b), $E_{CTRL_k}$ corresponds to the control structures in lines 1, 2, 4, 6, 9 and 11. Further, $E_{RD_k}$ and $E_{WR_k}$ can be expressed as:

$$E_{RD_k} = \sum_{r_k} N^{r_k} (E^{r_k}_{RD_k} + c \cdot E^{r_k}_{c}) \tag{6.4}$$

and

$$E_{WR_k} = \sum_{w_k} N^{w_k} \cdot E^{w_k}_{WR_k}, \tag{6.5}$$

where $r_k/w_k$ is a communication channel process $P_k$ reads from/writes to; $N^{r_k}/N^{w_k}$ is the number of times $P_k$ accesses each read/write channel, and $E^{r_k}_{RD_k}/E^{w_k}_{WR_k}$ 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 6.4.2). 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 $c \cdot E^{r_k}_{c}$. If the communication component is contention-free, $c$ is 0, if it is non-contention-free, $c$ is 1, while $E^{r_k}_{c}$ is the energy consumed while $P_k$ is waiting for data from channel $r_k$ when the communication component is non-contention-free. Similarly to $E^{r_k}_{RD_k}$ and $E^{w_k}_{WR_k}$, $E_{EXE_k}$ becomes:

$$E_{EXE_k} = N^{F_k} \cdot E_{F_k}, \tag{6.6}$$

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

$$E_{BLK_k} = E^{RD}_{BLK_k} + E^{WR}_{BLK_k}. \tag{6.7}$$

The energies $E^{RD}_{BLK_k}$ and $E^{WR}_{BLK_k}$ can be further expressed as:

$$E^{RD}_{BLK_k} = \frac{T^{total}_{BLKRD_k}}{(T^{1}_{BLKRD_k} + c \cdot T^{1k}_{c})} \cdot (E^{rd}_{BLK_k} + c \cdot E^{1k}_{c}) \tag{6.8}$$

and

$$E^{WR}_{BLK_k} = \frac{T^{total}_{BLKWR_k}}{T^{1}_{BLKWR_k}} \cdot E^{wr}_{BLK_k}, \tag{6.9}$$

where $T^{total}_{BLKRD_k}/T^{total}_{BLKWR_k}$ is the time spent in blocking on read/write by all the channels during the whole execution of the process $P_k$, $T^{1}_{BLKRD_k}/T^{1}_{BLKWR_k}$ is the time spent in one blocking on read/write by a channel, and $E^{rd}_{BLK_k}/E^{wr}_{BLK_k}$ 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 Figure 6.1(a) line 3. If contention may occur ($c$ is 1), checking the write counter on average will last longer with additional time $T^{1k}_{c}$, and the energy consumed by the checking will increase on average with $E^{1k}_{c}$.

The above mentioned energy profiles $E^{r_k}_{RD_k}$, $E^{w_k}_{WR_k}$, $E_{F_k}$, $E^{rd}_{BLK_k}$, $E^{wr}_{BLK_k}$ and $E_{CTRL_k}$ associated with an application process are obtained by first converting the corresponding part of the process code to its assembly equivalent, then counting the number of times $N_{inst}$ each assembly instruction *inst* is executed in the corresponding assembly equivalent, and finally assigning the energy cost $E_{inst}$ to each instruction in the processor ISA. Therefore, each energy profile is the sum of the number of times $N_{inst}$ each instruction *inst* is executed in the corresponding assembly equivalent multiplied by the energy cost $E_{inst}$

of the given instruction *inst* on a selected platform type. Hence, each of the above mentioned energy profiles can be represented with:

$$(E^{r_k}_{RD_k}, E^{w_k}_{WR_k}, E_{F_k}, E^{rd}_{BLK_k}, E^{wr}_{BLK_k}, E_{CTRL_k}) = \sum_{inst} N_{inst} E_{inst}. \tag{6.10}$$

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

$$E^{r_k}_c = \frac{T^{r_k}_{stall}}{T_{1stall}} \cdot E_{stall}, \tag{6.11}$$

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

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

$$E^{1k}_c = \frac{T^{1k}_c}{T_{1stall}} \cdot E_{stall} = \frac{T^{1k}_{stall}}{T_{1stall}} \cdot E_{stall}, \tag{6.12}$$

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

### 6.5.2 Derivation of Model Parameters

From the energy model formulation in Section 6.5.1 we can see that the energy model has three types of parameters – $N$ parameters such as $N^{r_k}$, $N^{w_k}$, $N^{F_k}$, $N_{inst}$; $T$ parameters such as $T^{total}_{BLKRD_k}$, $T^{total}_{BLKWR_k}$, $T^{1}_{BLKRD_k}$, $T^{1}_{BLKWR_k}$, $T^{r_k}_{stall}$, $T^{1k}_{stall}(T^{1k}_c)$, $T_{1stall}$; and $E$ parameters such as $E_{inst}$, $E_{stall}$. This section explains how the value of each of the parameters is obtained. Parameters $N^{r_k}$, $N^{w_k}$ and $N^{F_k}$ are obtained by counting integer points in input, output and process domain polytopes of $P_k$, see Section 6.4.1, which can be done automatically by using the *Barvinok* library in the `pn` compiler [VNS07]. 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 $N_{inst}$ 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

Instruction Set Simulators (ISS) or some hardware tracing circuits and our profiler tool. It is done only once per application for a selected processor type. Parameters $T^{total}_{BLKRD_k}$ and $T^{total}_{BLKWR_k}$ are obtained from a cycle-accurate SystemC timing simulation of PPNs [vHHK10]. This SystemC simulation should be performed for each application-to-platform mapping, because the blocking time, that is, waiting for data/space, depends on the specific mapping of the processes of an application to the platform. Parameters $T^1_{BLKRD_k}$ and $T^1_{BLKWR_k}$ are obtained by using ISS 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 $T^{r_k}_{stall}$ and $T^{1k}_{stall}$ are obtained for each mapping, by performing the analysis explained later in Section 6.5.2. Parameter $T_{1stall}$ is obtained from data-sheets or from measurements. The energy cost $E_{inst}$ for each instruction *inst* and the energy cost $E_{stall}$ for a stall are obtained from measurements, and this is done only once per platform type (processor type, communication infrastructure type, selected technology).

### Extraction of the Energy Costs

In this subsection we will describe how the energy costs $E_{inst}$ for each instruction *inst* and the energy cost $E_{stall}$ for a stall are derived.

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

$$E_{inst} = E_{inst_{tile}} + E_{comm} = (p_{inst_{tile}} + \frac{p_{comm}}{m})l_{inst} \tag{6.13}$$

and

$$E_{stall} = E_{stall_{tile}} + E_{comm} = (p_{stall_{tile}} + \frac{p_{comm}}{m})l_{stall}, \tag{6.14}$$

where $E_{inst_{tile}}$ 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 $p_{inst_{tile}}$, $p_{stall_{tile}}$, $p_{comm}$ with the instruction latency $l_{inst}$, and the stall latency $l_{stall}$. The power consumption $p_{inst_{tile}}$ is the power consumed by an instruction *inst* 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 $m$ is the maximum number of tiles that the interconnect allows. The power consumption of communication is captured within the $p_{inst_{tile}}$ 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 the 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 *inst* we determine its power cost $p_{inst_{tile}}$ 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 $m$, 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.

### Extraction of the Energy Profiles

In order to create the energy profiles $E^{r_k}_{RD_k}$, $E^{w_k}_{WR_k}$, $E_{F_k}$, $E^{rd}_{BLK_k}$, $E^{wr}_{BLK_k}$ and $E_{CTRL_k}$ associated with an application process $P_k$, 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 process, the read and write primitives for each channel and the control structures. Each process of an application is executed as many times as many different execution traces can occur for that process. The execution traces can be obtained by ISS or by some hardware tracing circuits. The execution traces usually contain program counter values, instructions and can also contain some additional information (such as branch is taken or not, and others). 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 process 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).

### Analysis of Communication Contention

The derivation of the energy model parameters $T^{r_k}_{stall}$ and $T^{1k}_{stall}$ related to non-contention-free communication components is explained in this subsection. Since our procedure analyzes the contention on a remote tile-to-communication memory link, that is, $\pi_j \leftarrow \pi_i$ link, we will use in the following the notation $r_{ij}$ for a read channel of a process mapped onto tile $\pi_j$ that reads from communication memory of a tile $\pi_i$. Thus, $T^{r_k}_{stall}$ and $T^{1k}_{stall}$ become $T^{r_{ij}}_{stall}$ and $T^{1j}_{stall}$ from a tile point of view.

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 each read channel $r_ij$ in $G$ for CB communication component is given in Algorithm 10. Inputs of the algorithm are the contention matrix $\mathbf{R}$ defined in Section 6.4.3, the number $m$ 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 $a_R$, $h_R$, $ra_R$, $a_W$, $h_W$, $ra_W$. 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 $ra_R$ and $ra_W$ may occur on a master-slave link if there is a re-arbitration, which happens 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 $a_R$, $h_R$, $ra_R$, $a_W$, $h_R$, $ra_W$ 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 10 gives the procedure to derive $T^{r_{ij}}_{stall}$ and $T^{1j}_{stall}$ for the CB and

**Algorithm 10:** Procedure to derive $T_{stall}^{r_{ij}}$ and $T_{stall}^{1j}$ for CB.

```
Input: Contention matrix $\mathbf{R}$, number of tiles $m$, read channels $r_{ij}$ of processes in $G$, size
       of data tokens $s_{r_{ij}}$ for each read channel $r_{ij}$, latencies of the communication
       infrastructure $a_R, h_R, ra_R, a_W, h_R, ra_W$.
Output: Arrays $\mathcal{T}_{stall}$ and $\mathcal{T}^1_{stall}$ of time parameters $T_{stall}^{r_{ij}}$ and $T_{stall}^{1j}$.
for $1 \leq i \leq m$ do
    $c_j = 0$;
    for $1 \leq j \leq m$ do
        $c_j = c_j + R_{ij}$;
    if $c_j > 1$ then
        $c_j = 1$;
    else
        $c_j = 1$;
for $1 \leq j \leq m$ do
    $q_j = 0$;
    for $1 \leq i \leq m$ do
        $q_j = q_j + R_{ij}$;
    if $q_j > 1$ then
        $q_j = 1$;
    else
        $q_j = 0$;
for $1 \leq j \leq m$ do
    $l = 0$;
    $T_{stall}^{1j} = 0$;
    for $1 \leq i \leq m$ do
        if $R_{ij} = 1$ then
            $L_{r_{ij}}^{1bc} = h_R$;
            for $r_{ij}$ such that $\pi_i \to \pi_j$ do
                $L_{r_{ij}}^{bc} = (2 + s_{r_{ij}}) \cdot h_R + q_j \cdot 0.5 \cdot ra_R + h_W + q_j \cdot 0.5 \cdot r_W$;
                $L_{r_{ij}}^{wc} = L_{r_{ij}}^{bc} + c_i \sum_{o=1, o \neq j}^{n} R_{io}((2 + s_{r_{ij}})(h_R + a_R) + q_o \cdot 0.5 \cdot ra_R + h_W +$
                $a_W + q_o \cdot 0.5 \cdot r_W)$;
                $T_{stall}^{r_{ij}} = (L_{r_{ij}}^{bc} + L_{r_{ij}}^{wc})/2$;
                $L_{r_{ij}}^{1wc} = L_{r_{ij}}^{1bc} + c_i \cdot \sum_{k=1, k \neq j}^{n} R_{ik}(h_R + a_R)$;
                $T_{stall}^{1j} = T_{stall}^{1j} + (L_{r_{ij}}^{1bc} + L_{r_{ij}}^{1wc})/2$;
                $l = l + 1$;
    $T_{stall}^{1j} = T_{stall}^{1j}/l$;
return $\mathcal{T}_{stall}, \mathcal{T}^1_{stall}$;
```

it consists of three parts: 1) for each communication memory it is determined whether contention may occur – lines 1 to 8; 2) for each processing tile it is determined whether re-arbitration may occur – lines 9 to 16 in the algorithm; and 3) the estimation of $T_{stall}^{r_{ij}}$ and $T_{stall}^{1j}$ is performed at lines 17 to 31. Since the circular round-robin arbitration pointer is statistically located in the middle of the search space, we estimate $T_{stall}^{r_{ij}}$ at line 26 in Algorithm 10 as the average value of the best case stall time $L_{r_{ij}}^{bc}$, line 24, and the worst case stall time $L_{r_{ij}}^{wc}$, line 25. 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 $h_R$, $h_W$, $a_R$, $a_W$, $ra_R$, $ra_W$ 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 6.4.2. 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 Figure 6.1(a), where $s_{r_{ij}}$ corresponds to *size_d* in Figure 6.1(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 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, that is, we multiply $r_R$ and $r_W$ by 0.5 in lines 24 and 25 in Algorithm 10. 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 30, 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 11. The input parameters are similar to the CB case with the 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 7 in Algorithm 11. Then in the following lines, we estimate $T_{stall}^{r_{ij}}$ and $T_{stall}^{1j}$, line 14, 17, as the average of the best case stall time $L_{r_{ij}}^{bc}$, $L^{1bc}$, line 12, 15, and the worst case stall time $L_{r_{ij}}^{wc}$, $L^{1wc}$,

**Algorithm 11:** Procedure to derive $T_{stall}^{r_{ij}}$ and $T_{stall}^{1j}$ for ShB.

```
   Input: Contention matrix $\mathbf{R}$, number of tiles $m$, read channels $r_{ij}$ of processes in $G$, size
          of data tokens $s_{r_{ij}}$ for each read channel $r_{ij}$, latencies of the communication
          infrastructure $a$, $h$.
   Output: Arrays $\mathcal{T}_{stall}$ and $\mathcal{T}_{stall}^{1}$ of time parameters $T_{stall}^{r_{ij}}$ and $T_{stall}^{1j}$.
1  $n\_r = 0$;
2  for $1 \leq j \leq m$ do
3      $y\_r_j = 0$;
4      for $1 \leq i \leq m$ do
5          $y\_r_j = y\_r_j + R_{ij}$;
6      if $y\_r_j = 0$ then
7          $n\_r = n\_r + 1$;
8  for $1 \leq j \leq m$ do
9      for $1 \leq i \leq m$ do
10         if $R_{ij} = 1$ then
11             for $r_{ij}$ such that $\pi_i \rightarrow \pi_j$ do
12                 $L_{r_{ij}}^{bc} = (3 + s_{r_{ij}})h$;
13                 $L_{r_{ij}}^{wc} = L_{r_{ij}}^{bc} + (n - n\_r - 1)(3 + s_{r_{ij}})(h + a)$;
14                 $T_{stall}^{r_{ij}} = (L_{r_{ij}}^{bc} + L_{r_{ij}}^{wc})/2$;
15     $L^{1bc} = h$;
16     $L^{1wc} = L^{1bc} + (n - n\_r - 1)(h + a)$;
17     $T_{stall}^{1j} = (L^{1bc} + L^{1wc})/2$;
18 return $\mathcal{T}_{stall}$, $\mathcal{T}_{stall}^{1}$;
```

line 13, 16. In the best case, only one tile wants to read from a communication memory. By computing how many tiles $(n - n\_r - 1)$ read from any of the communication memories, we determine how long the tile may wait in the worst case.

## 6.6 Evaluation of the Energy Model

We evaluate our energy model, proposed in Section 6.5, by showing its accuracy considering various application-to-platform mappings. The obtained energy estimates by using our energy model are compared to real energy measurements obtained from real implementations of the considered systems, that is, applications, platforms and mappings. These real measurements are 100% accurate, thereby can be used as credible reference points. 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 [Mic] 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 [AXI] 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 a Sobel edge-detection filter and the second application is a MJPEG 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 MJPEG PPN model consists of 6 processes and 5 channels with much higher computation/communication ratio, and hence the MJPEG is a computation-dominant application. Since the maximum number of processes among these two applications is 6, we performed 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 6.5, we estimate the energy consumption for each application-to-platform mapping, specified in the first column of Table 6.1, 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 6.1 that our energy model is highly accurate for all three types of interconnects, with an

**Table 6.1:** *Accuracy of the energy model for CB, ShB and P2P MPSoC platforms*

| *app → 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 |
| MJPEG_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 |
| MJPEG_3_m1 | 44.1 | 42.73 | -3.1 | 40.5 | 42.19 | +4.17 | 43.64 | 44.39 | +1.72 |
| MJPEG_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 |
| MJPEG_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 |
| MJPEG_5_m1 | 125.63 | 121.65 | -3.17 | 127.75 | 119.31 | -6.61 | 137.94 | 126.84 | -8.05 |
| MJPEG_6_m1 | 77.4 | 77.15 | -0.32 | 74.7 | 75.27 | +0.76 | 84.42 | 79.32 | -6.04 |

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 the 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 6.2. By comparing Table 6.1 and Table 6.2, 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 when considering the contention in the energy model. Therefore, in our proposed energy model special attention is paid to modeling the contention on communication infrastructures.

From the results shown in Table 6.1 it is clear that our energy model is very accurate. Now, we would like to discuss 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 6.1, we measured the time needed for the energy estimation. 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 6.5.1 and derivation of $T^{r_{ij}}_{stall}$ and $T^{1j}_{stall}$ parameters, and the rest of the time is spent on getting the $T^{total}_{BLKRD_j}$ and $T^{total}_{BLKWR_j}$ parameters. The derivation of the other model parameters is not considered in this model evaluation time because they are derived only once at the beginning when the model is calibrated and they are independent of the mapping. The time efficiency of the proposed energy model is very good given its high accuracy. Note that the majority of the evaluation time (99%) is spent

**Table 6.2:** *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 |
| MJPEG_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 |
| MJPEG_3_m1 | 44.1 | 38.65 | -12.36 | 40.5 | 38.23 | -5.6 |
| MJPEG_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 |
| MJPEG_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 |
| MJPEG_5_m1 | 125.63 | 109.39 | -12.93 | 127.75 | 106.79 | -16.41 |
| MJPEG_6_m1 | 77.4 | 71.51 | -7.62 | 74.7 | 68.24 | -8.65 |

in the SystemC cycle accurate simulation. We run cycle accurate SystemC simulation for each mapping in order to obtain very accurate $T^{total}_{BLKRD_j}$ and $T^{total}_{BLKWR_j}$. 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.

## 6.7 Discussion

We have proposed, in this chapter, 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 show 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 4.34% with a standard deviation of 3.35% in comparison to real energy measurements for all considered communication infrastructures. The average model evaluation time of 2.5 minutes per single design point is very good given the high accuracy of the proposed energy model.