Microprocessors and Microsystems xxx (2015) xxx–xxx

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

Hardware resource utilization optimization in FPGA-based Heterogeneous MPSoC architectures ☆

Bouthaina Dammak [a,*], Mouna Baklouti [b], Rachid Benmansour [a], Smail Niar [a], Mohamed Abid [b]

[a] University of Valenciennes and Hainaut Cambrésis, 59300 Valenciennes, France
[b] National School of Engineers of Sfax, 3042 Sfax, Tunisia

ARTICLE INFO

Article history:
Received 30 December 2014
Revised 7 April 2015
Accepted 5 May 2015
Available online xxxx

Keywords:
FPGA
MPSoC
Hardware accelerators
MIP model

ABSTRACT

Next generation FPGA circuits will allow the integration of dozens of hard and soft cores as well as dedicated accelerators in the same chip. These Heterogeneous Multiprocessor System-on-Chip (Ht-MPSoC) architectures will allow the design of very complex System-on-Chips (SoC) on a single FPGA chip and will fulfill modern application requirements, in terms of performance/energy consumption ratio. In this paper, we extend existing FPGA-based Ht-MPSoC architectures by considering sharing hardware accelerators among the cores. In these architectures, cores on the FPGA may have different resources that can be shared in different manners. To explore the large space of possible configurations of Ht-MPSoC on FPGA, designer needs a fast and accurate exploration tool. For this reason, a Mixed Integer Programming (MIP) model is also proposed to determine the Ht-MPSoC configuration that consumes the least HW resources while respecting the application execution time constraints. Using our MIP model, the design space of several hundreds of private and shared HW accelerators can be explored in a reasonable time with high accuracy.

1. Introduction

The increase in HW resources in the latest FPGA generation, makes it possible to implement extremely complex Heterogeneous Multi-Processor System-on-Chip (Ht-MPSoC) architectures. These architectures combine hardware and/or software cores, application-specific HW accelerators and communication units. The Xilinx Zynq 7000 Extensible Processing Platform (EPP) is an example of such architectures embedding a dual core ARM Cortex A9 processor and tens of thousands of programmable gate arrays [1]. Cyclone V from Altera [2] and SmartFusion2 from Micro-Semi [3] are other examples of Ht-MPSoC. These architectures include one or more hard-cores and up to 500 K of reconfigurable logic elements to build computational accelerators (Fig. 1).

Thanks to this reconfigurable area, it is possible to design either a *Symmetric Ht-MPSoC* (SHt-MPSoC), in which all the processors have the same number of private and shared HW accelerators, or an *Asymmetric architectures* (AHt-MPSoC) where HW accelerators attached to the different processors differ from one processor to the other. Figs. 2 and 3, show two examples of Ht-MPSoC with 4 processors (P1 to P4). Fig. 2 highlights a 4-core SHt-MPSoC architecture, in which the processors have the same type and number of HW accelerators. In Fig. 2, each processor has one and the same HW accelerator (named Pr) and share m accelerators with the other processors. Fig. 3 gives an example of an AHt-MPSoC. In this architecture P1, P3 and P4 have each one a private accelerator (named Pr i). These private accelerators can be similar or different. P2 has no private accelerator. P1 and P2 share the same accelerator, named "Sh 1" in Fig. 3, whereas P2, P3 and P4 share another HW accelerator, named "Sh 2". In an AHt-MPSoC, critical applications (or tasks for a multitasked system) are executed by cores with a large number of private accelerators. At the opposite, applications that are not critical are run by cores with small number (or not at all) private HW accelerators.

We think that AHt-MPSoC is a very promising class of architectures as it allows an efficient utilization of HW resources and provides high performances with less energy consumption [5]. However, their utilization increases even more the size of the design space of configurations to explore. Thus, it is necessary to provide to the designer a Design Space Exploration (DSE) tool to determine the best architectural configuration for a given set of concurrent applications. In addition, this tool plays an important role in the design flow of Ht-MPSoC architectures as it allows to

☆ The work was done as part of a Franco-Tunisian project.

* Corresponding author.
E-mail addresses: bouthaina.dammak@univ-valenciennes.fr (B. Dammak), mouna.baklouti@enis.rnu.tn (M. Baklouti), rachid.benmansour@univ-valenciennes.fr (R. Benmansour), smail.niar@univ-valenciennes.fr (S. Niar), mohamed.abid@enis.rnu.tn (M. Abid).

http://dx.doi.org/10.1016/j.micpro.2015.05.006
0141-9331/

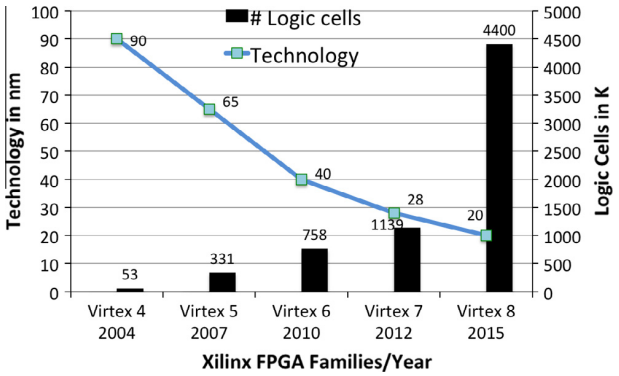

Fig. 1. Increase in the number of logic cells for Xilinx FPGAs (on the right axis) for different semiconductor device fabrication node (on the left axis) [4].

determine the most efficient Ht-MPSoC configuration in a reduced time. This configuration is the one that requires the least FPGA resources and gives a reduced execution time and energy budget.

In the literature, very few works have been devoted to DSE tool for AHt-MPSoC. This is due to the fact that previous and current generation of FPGA circuits offer relatively few resources, in terms of logic elements, compared to ASICs. Thus, it was possible to explore the entire design space in relatively short time interval either by simulation or by simple analytical models. In other studies, the authors only consider SHt-MPSoC architectures in which all processors have the same number and type of accelerators. This approach limits the application of Ht-MPSoC and cannot effectively operate for high complex reconfigurable systems.

In the solution that we propose in this paper, we target next generation FPGA circuits with a high number of reconfigurable logic elements and their utilization in AHt-MPSoC architectures. In these architectures, the number of HW accelerators and their type may vary from one processor to another. Furthermore in our model, the applications executed by the processors may differ from one processor to another. Our solution is based on Mixed Integer Linear Programming (MIP) formulation to explore the very large space of possible configurations. Due to the complexity of finding an optimal enumerative solution, the proposed mathematical model allows the identification of a global minimum (i.e. area usage) in a reasonable time. This model was solved, after a process of constraints linearization, using CPLEX linear program solver.

The paper consists of 6 sections. In the next section, a survey on existing approaches in Ht-MPSoC is presented. In Section 3, we detail AHt-MPSoC architectures and discuss their benefits. In Section 4, we develop our MIP-based Design Space Exploration

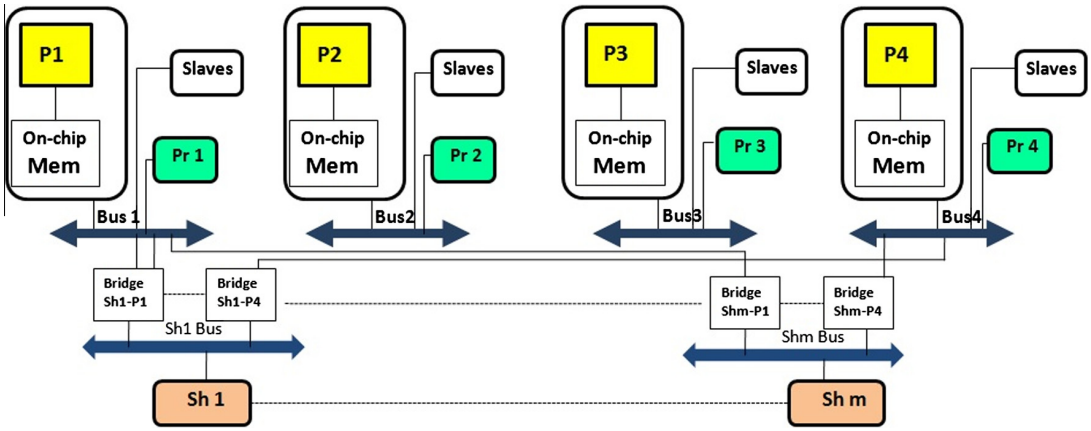

Fig. 2. Example of a SHt-MPSoC architecture with 4 processors. Each processor has one private HW accelerator (Pr *i*) and share m accelerators (Sh *i*) with the other processors.

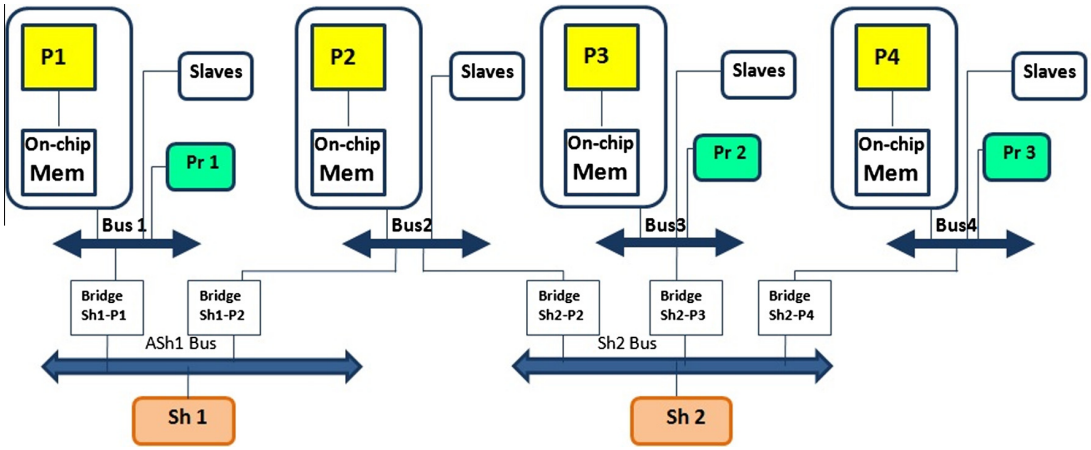

Fig. 3. Example of an AHt-MPSoC architecture with 4 processors. P1 and P2 share one accelerator (Sh 1) and P2, P3 and P4 share another accelerator (Sh 2). P2 has no private accelerator.

Please cite this article in press as: B. Dammak et al., Hardware resource utilization optimization in FPGA-based Heterogeneous MPSoC architectures, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.05.006

(DSE) for AHt-MPSoC. The next section presents the experimental results and the obtained performances of our AHt-MPSoC for real and synthetic benchmarks. Finally in Section 6, we give a conclusion and some possible extensions to make AHt-MPSoC more efficient.

2. Related works

The integration of custom instruction in FPGA-based MPSoC increases the performance gain by incorporating hardware components to handle computational tasks [6–9]. Modern platforms, including FPGAs and ASICs support different couplings of hardware components with the processor. In [10], couplings schemes are classified into two principal modes: Closely coupled mode and Loosely coupled mode (Fig. 4).

In the first mode, the hardware accelerator is part of the processor data path and has direct access to the processor memory. At the opposite, in the second mode, the accelerator is placed outside the processor on a dedicated bus [8,10,11]. A group of closely coupled hardware components operates at a single clock cycle fixed by the slower components. At the opposite, each loosely coupled hardware component runs at its fastest possible individual frequency. Loosely coupled mode is quite popular in multi-media applications like image encoding/decoding applications. Nomadik [12], Freescale i-Mx35 [13] and S3C6400 [14] are examples of multi-media architectures designed with loosely coupled accelerators. These platforms embed on the same die an ARM [15] processor and different multi-media accelerators for video, audio, imaging, and graphics processing.

For recent multi-media applications, a large number of custom instructions can be identified to be executed in hardware components. In order to avoid an excessive area usage of hardware components, previous works propose to identify and exploit commonality between identified custom instructions and to share hardware resources. In [16], the authors propose a polynomial-time heuristic that uses resource sharing to minimize the area required to synthesize a Set of custom Instruction Extension (ISEs). Their resource sharing approach transforms the set of ISEs into a single hardware data path. Nevertheless, their proposed heuristic minimizes the ISEs area usage without a control on latency constraint.

Zuluaga et al. [17] introduce latency constraints in the merging process of the ISEs to control the performance improvement. Their proposed parametric algorithm combines a path-based resource sharing algorithm, similar to the ones presented in [16], with a timing budget management scheme.

More recently, the work presented by Stojilovič et al. [18] aims at a pragmatic increase in flexibility to integrate different ISEs from different applications. This work is motivated by data path based algorithm. While [16] aims at minimizing the area cost, [18] increases HW accelerators flexibility for a moderate cost. Their approach ensures that all instruction set extensions (ISEs) from an application domain map on the same proposed domain-specific coarse-grained array. The architectures proposed in these papers belong to loosely coupled application-specific architectures. The work we present in this paper differs from [16–18] at various levels. First, they consider the sharing of fine grained custom instructions that are closely coupled to the processor pipeline. At the opposite, we consider loosely coupled class application-specific architectures. The sharing of coarse grained custom instructions provides a better performance improvement.

In most of the cited works, the proposed approach is to share HW logic between different custom instructions for several tasks mapped on the same and single processor. Their proposed heuristics select the custom instructions to be mapped on logic providing a more area saving with operation sharing. Instead, we propose the sharing of an entire custom instruction between different cores and our MIP model explores the custom instructions to be mapped on hardware and the sharing-degree of hardware to support the ISE.

In [16–18], the proposed algorithms only focus on area reduction without taking into account the impact on performance. Even if authors of [17] introduce the latency features, they did not consider application-performance constraints. At the opposite, our work explores all the possible sharing configurations that minimize the area usage and respect the desired application performance constraints in term of execution time.

In [19], the authors present another approach to minimize runtime reconfigurable area by resource sharing for closely coupled application-specific architectures. In this paper, the authors propose to share at runtime reconfigurable area for multi-core architecture. They develop an algorithm to select the ISEs to be mapped on the same fabric to optimize the fabric sharing between cores leading to the best execution time.

In [20], the authors propose a pseudo-polynomial time algorithm to explore the design space of Multi-Application Specific

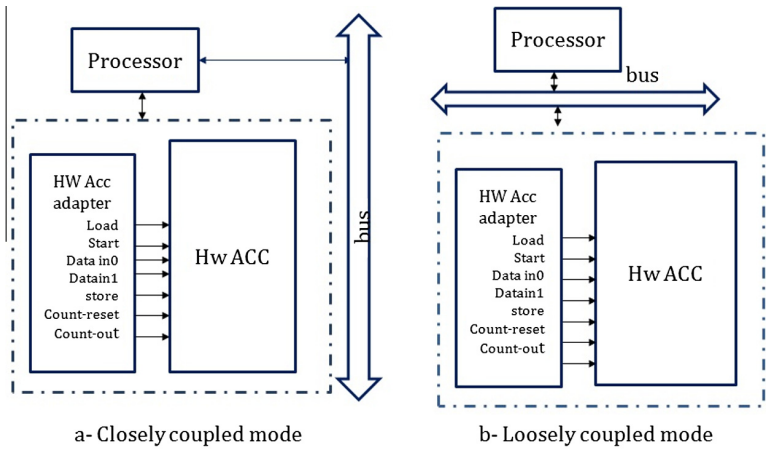

Fig. 4. Hardware accelerator architectures classification: Closely coupled and loosely coupled.

Please cite this article in press as: B. Dammak et al., Hardware resource utilization optimization in FPGA-based Heterogeneous MPSoC architectures, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.05.006

Instruction Processor (or M-ASIP). Their algorithm identifies the appropriate application-partitions and identifies custom instructions satisfying the area-performance trade-off.

At the opposite of our work, [20] did not consider custom instructions sharing. [19] consider the sharing of runtime reconfigurable fabric between processors for mapping different hardware components executing different custom instructions, However, we propose to share the hardware components between different processors to execute different custom instructions.

[21] also targets FPGA area minimization by the utilization of hardware sharing between different functional blocks. However, the level of sharing proposed by the authors depends on functional modules that constitute a particular design (fabric slices, distributed RAMs, DSP slices, block RAMs). As demonstrated by the authors, their approach is less efficient in terms of fabric usage and clock frequency than the designs they are derived from. In this paper, the design space exploration of hardware sharing is not automated. Moreover, the proposed sharing approach can be applied only to uni-processor architecture design. In our paper, we first consider MPSoC architectures and the sharing is done at higher level. Our approach does not produce clock frequency reduction or fabric usage overheads.

The approaches for design space exploration of hardware resources sharing proposed in [16–18] focus on closely coupled single architectures. In [20,19], the authors consider multiprocessor architectures but they did not consider coarse grain instruction sharing. We think, in the context of FPGA based hybrid architectures, coarse grain hardware accelerators are much more efficient in terms of efficiency and power than fine grain hardware accelerators due to communications overheads. Moreover, in [20,19], the fine grained hardware accelerators are implemented on runtime reconfigurable hardware resources. However, the primary drawback of using runtime reconfiguration is the significant delay of reprogramming the hardware. Thus, we think that the runtime reconfiguration delay and the sharing delay will dominate the total execution time, especially applications with a small amount of computation between two consecutive hardware accelerators. For all these reasons, it is difficult to compare the different approaches.

3. AHt-MPSoC based on hardware sharing

Application-specific instructions are an effective way of improving the performance of processors. In these processors, the execution time of the critical computations is reduced by the utilization of new instructions executed on HW accelerators. These HW accelerators can be either loosely coupled to the processor via system bus, or memory controller, or closely coupled to the instruction pipeline. Within this work, the HW accelerators are implemented as hardware modules executing application-specific instructions, and are loosely coupled to the processor.

In this section, we first discuss the benefits of sharing HW accelerators in a multiprocessor architecture, then we propose our AHt-MPSoC architecture.

3.1. Proposed hardware sharing approach

A Ht-MPSoC, running N applications on the different processors, cannot implement all the HW accelerators. Due to hardware resources constraints, the HW accelerators integration for Ht-MPSoC architecture cannot be fully exploited. Most of existing embedded applications, such as multimedia, telecommunication or automotive applications, use the same set of kernel functions. Matrix operations, convolutions and filters are frequently used in such applications. For an Ht-MPSoC, that does not use hardware-sharing, separate custom private accelerators are needed for different custom instructions to provide the same computation power. The proposed sharing approach enables to share a range of specific instructions among different tasks and different computational tasks can share several HW accelerators. It is expected that this optimization will extenuate the area and power consumption and will preserve performance. We call a *shared pattern* the computational task existing on different applications. The shared pattern identification offers a range of possible optimization. In Fig. 5, different applications have the same computational patterns that can be executed on different and private HW accelerators. However based on our proposed technique of sharing accelerators, only one accelerator for each identified pattern can be implemented and shared among the processors. Currently, in the work we implemented so far, the identification of common patterns is done manually. This solution is feasible for small applications or when the number of parallel applications is reduced. For complex applications and/or a large number of concurrent applications, we need to do such identification automatically. We can use existing approaches such those proposed in [22].

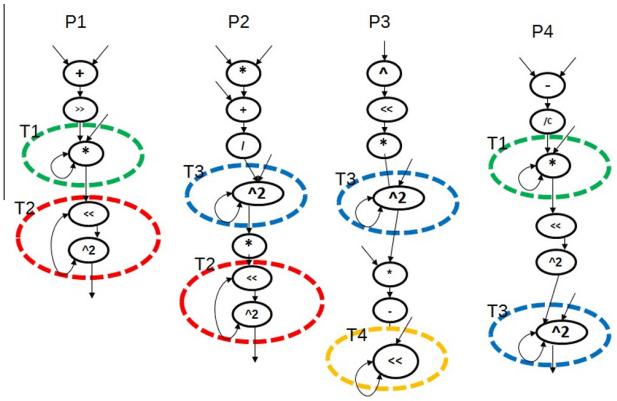

Fig. 5. An example of 4 parallel programs with different patterns that could be executed in HW accelerators. Patterns are represented by circles with different colors. Patterns with the same color are similar.

The *sharing degree* defines the number of processors sharing the same accelerator. In Fig. 5, a 3-shared degree HW accelerator for T3, can be shared between P2, P3 and P4 processors. For seek of clarity, in this figure we consider simple patterns. In our benchmarks, patterns contain more instructions. The accelerator sharing and the sharing degree between processors provide a large architectural space exploration. The next paragraph gives an overview of the proposed hardware AHt-MPSoC architecture.

3.2. AHt-MPSoC architecture

The AHt-MPSoC architecture class, proposed in this paper, uses HW accelerators that execute application-specific instructions and are loosely coupled to processor via shared bus. Bus-bridges are used to communicate between the processor and the shared accelerators. Fig. 3 illustrates 4-processor AHt-MPSoC where each processor has private accelerator as well as shared accelerators. The latter are activated by application-specific instructions.

The synchronization between processors, having shared HW accelerators, is established through a lock/barrier mechanism implemented in hardware. When a processor P_j tries to use a shared accelerator, which is already locked by another processor P_i, it is put in a waiting state until P_i unlocks the shared accelerator. The lock/barrier mechanism follows the "First In First Out" fashion. Fig. 6 details the synchronisation process between two processors. In the next Section, a Mixed Integer Linear Programming (MIP) model is presented for identifying the shared optimal resource allocation architecture respecting a required performance.

Please cite this article in press as: B. Dammak et al., Hardware resource utilization optimization in FPGA-based Heterogeneous MPSoC architectures, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.05.006

```
//control the syn register, we use test&set op

    while (Read HwA(baseaddr, reg _control offset) == 1)
    Do
//if shared HwAcc is in use, Pi makes no op and waits
    ;
    end while
//Pi uses HwAcc
    write HwA (baseaddr, reg_control offset, 1 );
// processing HwA and read results
    .....
//Pi relases the Hw Acc
    write HwA(baseaddr, reg _control offset; 0);
```

Fig. 6. Synchronised access to shared Hw Acc.

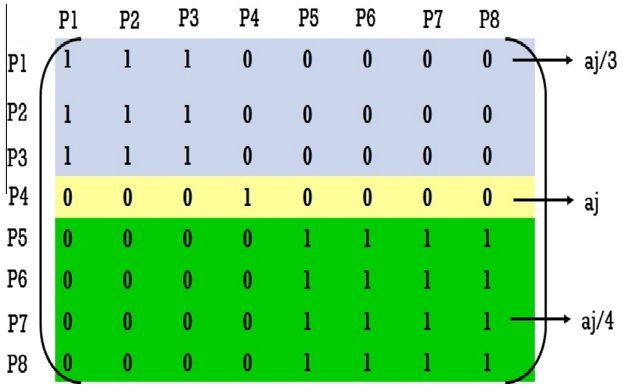

Fig. 7. y_{jik} variables for a T_j pattern.

4. Mixed integer linear programming model

Our space exploration addresses the way to merge the computational patterns, existing on the different applications, to reduce the overall area usage while respecting applications-performance constraints. Increasing the sharing degree reduces the area usage, but may increase the delay of each processor to access shared accelerators and therefore the required performance will not be met. This situation cannot be accepted for hard and soft real time applications. Thus, the goal of our MIP model is to have minimum area usage, while keeping the execution time of each application under a required limit.

4.1. Problem formulation

The architecture is a MPSoC with n processors running n applications. These applications can be similar, i.e. Single Program Multiple Data model, or different, i.e. Multiple Program Single/Multiple Data. We define pattern as a time consuming task-kernel existing in one or different tasks. Based on the expected acceleration, we select the sequence of the most computational patterns executed on the n processors. These patterns are candidates to hardware implementations that may fulfill the desired performance. Let $\{T_1, T_2, \ldots, T_m\}$ denotes this sequence and $\{P_1, P_2, \ldots, P_n\}$ the sequence of n homogeneous processors. Finally we define N and M as, respectively, the set of processor $N = \{1, 2, \ldots, n\}$ and the set of patterns $M = \{1, 2, \ldots, m\}$.

Each consuming pattern T_j, is assigned a predefined area constant a_j. The value a_j represents the number of FPGA area units required by T_j to be implemented as HW accelerator. In addition, we define two continuous constants ts_{ji} and te_{ji} respectively for the start-time and the end-time of execution time of the pattern T_j on processor P_i.

The implementation of T_j in private way provides a defined acceleration $tacc_j$. For each processor P_i, an acceleration constant $limit_i$ is set as a constraint for the total processor acceleration acc_i.

Our problem definition can be formally declared as follows: Given a number of m patterns executed on n homogeneous processors, we look for the number of processors sharing each pattern (sharing degree) so that the maximum execution time of each processor is under the time-constraint and the total used area is minimized.

4.2. Objective function

In this section, we present a MIP formulation of the problem so that we can obtain an optimal solution with the help of a commercial optimization solver for mixed integer linear programming.

Let $x_j, j \in M$, be a binary variable that denotes whether the pattern T_j is implemented on Hardware (HW) for processor P_i or not.

$$x_j = \begin{cases} 1 & \text{if } T_j \text{ is implemented on } HW, \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Let $y_{jik}, j \in M, i \in N, k \in N$, be a binary variable that denotes whether the HW accelerator of task T_j is shared between processors P_i and P_k or not.

$$y_{jik} = \begin{cases} 1 & \text{if Acc of } T_j \text{ is shared between } P_i \text{ and } P_k, \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

We assume that $y_{jii} = 1$, this leads to the following equation:

$$\sum_{i=1}^{n}\sum_{k=1}^{n} y_{jik} \geqslant 1 \quad (3)$$

Our objective function aims at minimizing the total area required to implement the m patterns.

$$Total\_Area = \sum_{j=1}^{m}\sum_{i=1}^{n} x_j \frac{a_j}{\sum_{k=1}^{n} y_{jik}} \quad (4)$$

In the Total_Area equation, we have implicitly defined the sharing degree of a processor P_i for the task T_j. This sharing degree will be denoted by sh_{ij} and is equal to $\sum_{k \in N} y_{jik}$.

Fig. 7 shows an example of a matrix $Y_j = y_{jik}$ for the T_j pattern implemented on a 8-processor architecture. Each row i (respectively column k) in the matrix corresponds to processor i (respectively processor k) in the MPSoC. The variable on row i and column k corresponds to y_{jik} variable and determines if processors P_i and P_k share the same accelerator for T_j. For example, in Fig. 7, from the first three rows (blue[1] region), we deduce that the same HW Accelerator for T_j is shared between P_1, P_2 and P_3. For this region, for each row, the area consumed has been reduced by a factor of 3 and is equal to aj/shij = aj/3. Likewise, for the green region, a 4-shared HW accelerator is shared between P_5, P_6, P_7 and P_8. Each row j in this region needs aj/4 area units. The 4th row of the matrix shows that P_4 has a private HW Accelerator for T_j and consumes a_j area units.

In our problem, the objective function is the ratio of two linear terms. In order to linearize this function, (i) we define new continuous variables z_{ij}, w_{ij}, and θ_{ijk}:

$$z_{ij} = \frac{1}{sh_{ij}} = \frac{1}{\sum_{k=1}^{n} y_{jik}} \quad (5)$$

$$w_{ij} = z_{ij} x_j \quad (6)$$

$$\theta_{ijk} = z_{ij} y_{jik} \quad (7)$$

[1] For interpretation of color in Fig. 7, the reader is referred to the web version of this article.

and (ii) we add the following constraints to our model:

$$z_{ij}\sum_{k=1}^{n} y_{jik} = 1 \tag{8}$$

$$\sum_{k=1}^{n} \theta_{ijk} = 1 \tag{9}$$

The objective function (Eq. (4)) can be re-written as:

$$Total\_Area = \sum_{j=1}^{m}\sum_{i=1}^{n} a_j w_{ij} \tag{10}$$

4.3. Performance constraint

The performance constraint is based on the following principle: the total acceleration for each processor i provided by the mapping of the different tasks on HW accelerators needs to be upper a performance limit ($limit_i$). Regarding the sharing degree of Hw accelerator, each processor has a delay R_{ji} to access this shared HW accelerator.

The performance constraint can be imposed as follows:

$$acc_i = \sum_{j=1}^{m} x_j tacc_j - x_j R_{ji} \geqslant limit_i, \tag{11}$$

R_{ji} variable is defined as the time interval between the end-time of executing task T_j on processor P_k, where P_k $(0 \leqslant k \leqslant i-1)$ is the last processor sharing Hw accelerator of T_j with P_i, and the start-time of executing task T_j on processor P_i.

$$R_{ji} = te^h_{jk} - ts^h_{ji} \text{ where } k = max\{0,1,\ldots,i-1\} \text{ and } y_{jik} = 1 \tag{12}$$

$$R_{ji} = \sum_{k=1}^{i-1} u_{jik}(te^h_{jk} - ts^h_{ji}) \tag{13}$$

where te^h_{jk} and $ts^h_{ji}, j \in M, i \in N, k \in \{1,2,\ldots,i\}$, are continuous variables that define respectively the start-time and the end-time of executing T_j on HW, respectively on processors P_k and P_i and are calculated as follow:

$$ts^h_{ji} = ts_{ji} - \sum_{l=1}^{j-1}(acc_l - R_{li}) \tag{14}$$

$$te^h_{jk} = te_{jk} - \sum_{l=1}^{j}(acc_l - R_{lk}) \tag{15}$$

and $u_{jik}, j \in M, (i,k) \in N^2$, is a binary variable defined as follow:

$$u_{jik} = \begin{cases} 1 & \text{if } P_k \text{ is the last processor sharing } T_j \text{ with } P_i, \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

Fig. 8 shows an example to calculate a delay of processors P_2 and P_3 to access a shared HW accelerator of task T_j.

Now the performance constraint can be re-written as:

$$acc_i = \sum_{j=1}^{m} x_j tacc_j - x_j \sum_{k=1}^{i-1} u_{jik}(te^h_{jk} - ts^h_{ji}) \geqslant limit_i \tag{17}$$

Our objective function is to minimize the total area:

$$Total\_Area = \sum_{i=1}^{m}\sum_{j=1}^{n} a_i w_{ij}$$

5. Experimental results

To evaluate the performance of the proposed AHt-MPSoC system as well as to study the effectiveness of our MIP model, we use synthetic and real applications. Our target platform is a Xilinx virtex 5 FPGA. On this platform, several Microblaze softcores running at 125 MHz can be mapped.

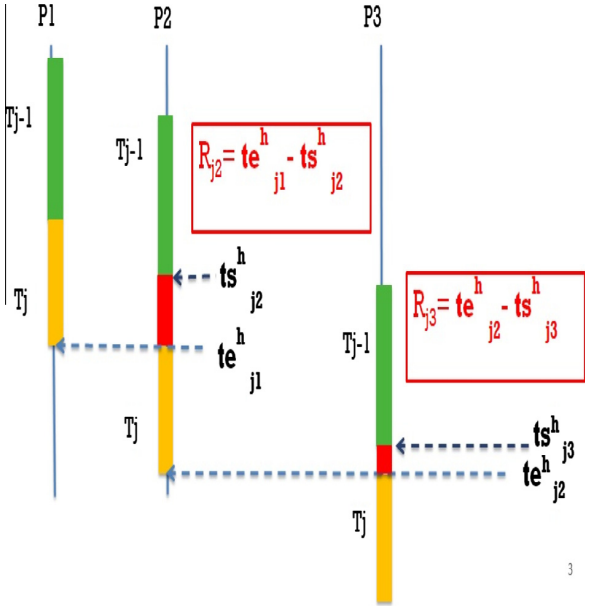

Fig. 8. Example to compute delays for a 3-shared Hw accelerator for T_j pattern.

Performance measurement and area usage are presented respectively in terms of clock cycles and area units. Power consumption has been measured using the Xilinx Xpower tool.

5.1. Synthetic applications

In this section, we use synthetic applications that are produced based on three computational patterns and non-computational tasks implemented as loop iterations, as illustrated in Fig. 9. For the different processors, we vary the number of iterations (i, j and k) of the non-computational loops to obtain different applications and different delays between the computation tasks T1 to T3. The three computational patterns consist on:

- T1 implements an inversion of a vector of 16 bits
- T2 implements 8 * 8 integer matrices multiplication and
- T3 implements search of a maximum value in a vector of 64 integers in memory

Table 1 summarizes the execution time and the area requirement for T1, T2 and T3 tasks. Note that the area requirement is presented in term of area unit. Here, an area unit corresponds to 150 slices. In these experiments only the additional area needed for HW accelerator is given, as the number of soft-cores is constant and has been fixed to 8.

Fig. 10 presents the logic area usage calculated based on our proposed MIP model while varying the required speedup. For each resolution, cplex requires 6 s to generate the optimal configuration. In Fig. 10, for each required speedup, we calculate the different $limit_i$ constraints for the different processors (Eq. (11)). The required $speedup_i$ of each processor i is defined as follows:

$$speedup_i = \frac{Tsw_i}{(Tsw_i - limit_i)} \tag{18}$$

where Tsw_i is the execution time of processor i without HW Accelerator. In Fig. 10, the configurations given by the MIP model

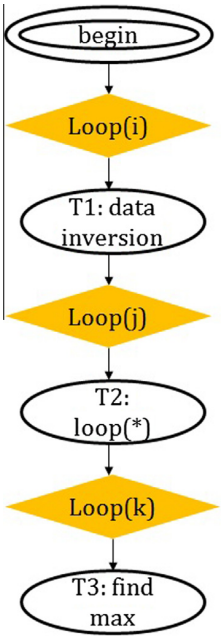

Fig. 9. Generation of different synthetic applications.

Table 1
T1, T2 and T3 Area and Execution time information.

	Area usage (area unit)	SW time (cycles)	HW time (cycles)
T1: Data inversion loop	2	440	222
T2: Loop multiplication	15	3815	213
T3: Search maximum	4	2000	1200

to satisfy 1.007 and 1.014 speedups only consume 2 area units. These solutions have only one shared HW accelerator for T1 ($x_1 = 1$ and $y_{1ik} = 1$), whereas T2 and T3 are executed in software ($x_2 = 0, x_3 = 0$). In contrast, when the speedup is increased, the generated solutions integrate T2 and T3 on HW accelerators. In Fig. 10, the solution given for a speedup of 2.15 consumes 25 additional area units to implement HW accelerators. This solution represents a configuration with HW accelerators for T1, T2 and T3 ($x_1 = 1, x_2 = 1, x_3 = 1$).

To illustrate the impact on performance and area consumption when HW accelerators are shared, we compare configurations of different points in Fig. 10 consuming the same area. The points 1.6 and 1.75 need 30 additional area units but they correspond to different configurations. In fact, for 1.6, the MIP model generates an AHt-MPSOC architecture with two HW accelerators of T2, one private for P1 and the second is shared between (P2, P3, P4, P5, P6, P7, P8). Whereas the AHt-MPSOC architecture, which provides a speedup equal to 1.75, has two HW accelerators of T2, the fist one is shared between (P1, P2, P4) and the second one is shared between (P3, P5, P6, P7, P8). We deduce that, different combinations of processors sharing the same HW accelerator could impact the performance of AHt-MPSoC architecture. Thus, for a fixed area on the FPGA, the designer has several possible configurations and he/she will choose the optimal hardware architecture configuration that provides higher performances.

Fig. 10 also demonstrates that the maximum speedup is provided with a reduced area-usage configuration compared to the configuration with only private HW accelerators. The 8-processor architecture with private HW accelerators for T1, T2 and T3 patterns provides a speedup equal to 2.6 and consumes 136 area units. To guarantee the same speedup, our MIP model generates a configuration that consumes only 96 area units. The generated AHt-MPSoC architecture integrates T1, T2 and T3 on HW accelerators ($x_1 = x_2 = x_3 = 1$ as shown in Table 2). This architecture has 4 HW accelerators for T1, 4 HW accelerators for T2 and 7 HW accelerators for T3. In Table 2, y_{1ik} vector indicates that for T1, processors (P1, P2, P3, P4) and (P5, P7) have two shared HW accelerators and (P6) and (P8) have their private ones.

In Fig. 10, we also compare the area usage of the MIP-based generated results and the real results obtained with the implementation on the FPGA. From this figure, due to the extra consumed logic-area needed by the Bus-bridges, we note a slight overhead difference between real measurement and MIP estimations. The area overhead depends on the number of implemented patterns as shared HW accelerators and is comprised between 3% and 6%. For a speedup equal to 1.6 only T2 is implemented as shared HW accelerator, thus the area overhead is about 3%. While for a speed up equal to 2.15, all patterns are mapped on shared HW accelerator

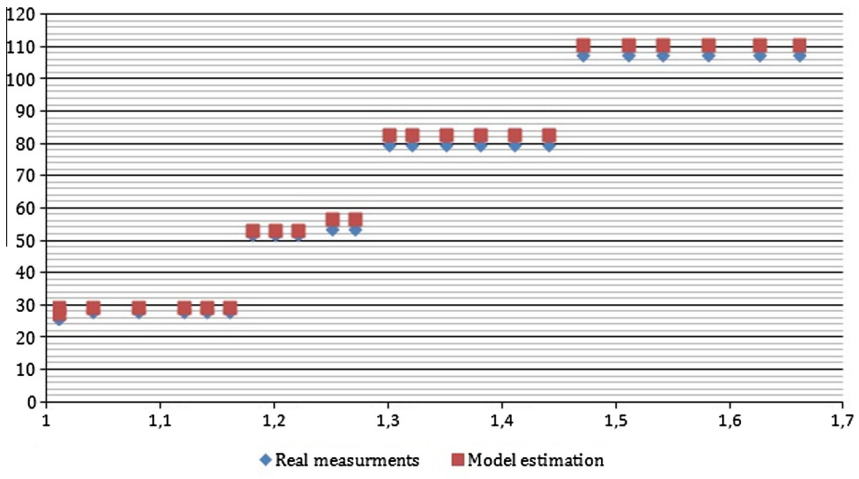

Fig. 10. Additional area usage (y-axis) of the HW accelerators for the generated configurations with different speedup (x-axis). In these experiments we use 8 processors and we compare real measurement on the FPGA with results obtained with our MIP model.

Please cite this article in press as: B. Dammak et al., Hardware resource utilization optimization in FPGA-based Heterogeneous MPSoC architectures, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.05.006

Table 2
MILP model outputs for a speedup constraint equal to 2.6.

Model variables	Variables value
Model inputs	
N	8
M	3
acc[M]	200 3375 1000
a[M]	2 15 4
te[M][N]	$\begin{Bmatrix} 440 & 670 & 910 & 1150 & 1240 & 1360 & 1470 & 1680 \\ 4255 & 4485 & 4725 & 4965 & 5055 & 5175 & 5245 & 5495 \\ 6255 & 6485 & 6725 & 6965 & 7055 & 7175 & 7245 & 7495 \end{Bmatrix}$
limit[N]	4150 4100 4200 4280 4300 4350 4500 4575
Model outputs	
x_j	1 1 1
y_{1ik}	$\begin{Bmatrix} 1&1&1&1&0&0&0&0 \\ 1&1&1&1&0&0&0&0 \\ 1&1&1&1&0&0&0&0 \\ 1&1&1&1&0&0&0&0 \\ 0&0&0&0&1&0&1&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&1&0 \\ 0&0&0&0&0&0&0&1 \end{Bmatrix}$
R_{ji}	$\begin{Bmatrix} 0 & 10 & 10 & 0 & 0 & 0 & 10 & 0 \\ 0 & 200 & 0 & 170 & 0 & 0 & 0 & 0 \\ 0 & 0 & 120 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix}$

and the area overhead reaches 6%. These results demonstrate that the proposed MIP model produces results close to those obtained with real implementation. For the performance delay, we note that the delay caused by bridges, used for shared HW accelerators configuration varies according to the complexity of the pattern. For T2 pattern, the delay is about 4% of the total execution time while for T3 pattern the delay reaches 15%.

5.2. Jpeg codec application

In this section, we give experimental results for a real Jpeg codec application executed on a 8-processor MPSoC. Fig. 11 presents a general overview of the Jpeg codec. The image is decomposed into 8 * 8 blocks of pixels. Each block is compressed through the encoder process. The array of compressed blocks is stored or forwarded to transmission channels. The image is reconstituted through the decoder process. The c++ code of this codec is provided in [23]. The encoder takes bmp images to encode them into jpeg format. The decoder process decodes jpeg images into bmp images. This codec considers luminance and chrominance matrix, so it supports RGB images. The scale factor of this codec is equal to 50.

In our experiments, we use a 10 kbyte image and we profile the jpeg-encoder and the jpeg-decoder applications on a microblaze processor. Profiling results show that DCT (respectively IDCT) is the most time-consuming function for jpeg-encoder (respectively jpeg-decoder) application. The DCT function is mainly composed of two functions: horizontal DCT (noted HDCT) and vertical DCT (noted VDCT). Each function consumes almost 20% of the whole execution time. The IDCT function is composed of horizontal IDCT (IHDCT) and vertical IDCT (IVDCT). Each function consumes almost 15% of the jpeg-decoder execution time. The 2D-DCT and 2D-IDCT computations are detailed in Eqs. (19) and (21).

$$F(u,v) = \frac{1}{\sqrt{2N}} * C(u) * C(v) \sum_{y=0}^{N-1} \sum_{x=0}^{N-1} f(x,y) * \tag{19}$$

$$\cos\frac{(2x+1)*u*\pi}{2N} * \cos\frac{(2y+1)*v*\pi}{2N} \tag{20}$$

$$f(x,y) = \frac{1}{\sqrt{2N}} \sum_{v=0}^{N-1} \sum_{u=0}^{N-1} C(u) * C(v) F(u,v) * \tag{21}$$

$$\cos\frac{(2x+1)*u*\pi}{2N} * \cos\frac{(2y+1)*v*\pi}{2N} \tag{22}$$

where $C(u) = \begin{cases} \frac{1}{\sqrt{2N}}, \text{if } u = 0 \\ 0, \text{if } u \geqslant 0 \end{cases}$

The 2D-DCT and 2D-IDCT are computed using the separability property of this transform. This means that $F(u,v)$ and $f(x,y)$ can be computed in two separate steps. Each 2-D transform (forward or inverse) is divided in two 1-D transform. The separated transformations can also be expressed in matrix operations (Eqs. (23) and (24)). Fig. 12 details DCT and IDCT computations.

$$F = T * f * T^t \tag{23}$$

$$f = T^t * F * T \tag{24}$$

Where

$$T_{i,j} = \begin{cases} \frac{1}{\sqrt{N}}, & \text{if } i = 0 \\ \frac{2}{\sqrt{N}} * \cos\frac{(2j+1)i\pi}{2N}, & \text{if } i > 0 \end{cases} \tag{25}$$

For each function of Fig. 12, the computational parts are highlighted to be implemented as custom instructions. For each computational part, the matrix M is the matrix of DCT coefficient and it is represented in Q12 format of the T matrix (Eq. (25)). The HDCT and IHDCT parts consist on a multiplication of 1 * 8 integer matrix with an 8 * 8 integer matrix while the VDCT and IVDCT parts consist on a multiplication of 8 * 8 integer matrix with an 8 * 1 integer matrix. Thus, we observe that we can only associate one pattern for HDCT and IHDCT tasks and another pattern for VDCT and IVDCT tasks.

The HDCT/IHDCT and VDCT/IVDCT patterns consume respectively 28 and 26 area units when implemented on a dedicated Hw Accelerator. In these experiments an area unit corresponds to 70 slices on the FPGA.

The jpeg encoder/decoder applications will be explored on a 8-processor MPSoC architecture, in which four processors compute the encoder application while the four others execute the decoder application. The optimal AHt-MPSoC configuration will be selected through the MIP-based exploration process. The exploration finds

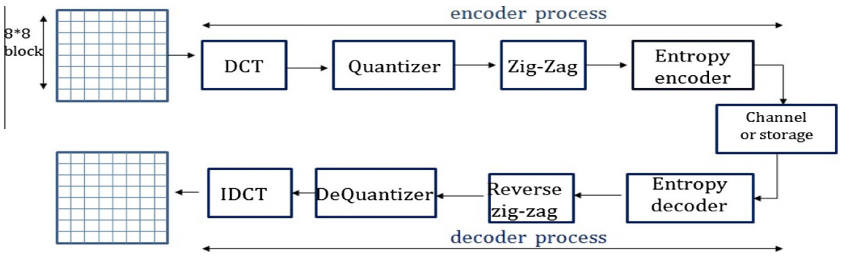

Fig. 11. Jpeg encoder and decoder tasks.

Please cite this article in press as: B. Dammak et al., Hardware resource utilization optimization in FPGA-based Heterogeneous MPSoC architectures, Microprocess. Microsyst. (2015), http://dx.doi.org/10.1016/j.micpro.2015.05.006