PR-HMPSoC: a Versatile Partially Reconfigurable Heterogeneous Multiprocessor System-on-Chip for Dynamic FPGA-based Embedded Systems

Tuan D. A. Nguyen *and* Akash Kumar
Department of Electrical & Computer Engineering
National University of Singapore
4 Engineering Drive 3, Singapore 117583
Email: {tuann, eleak}@nus.edu.sg

***Abstract*—FPGA-based heterogeneous Multiprocessor Systems-on-Chip (HMPSoCs) are becoming quite popular for high performance embedded systems because of their powerful computational ability and relatively flexible architecture to adapt to unexpected system requirement changes. However, with the insatiable demands of supporting an extensive range of applications beyond the limited resources of FPGA chip and shorter time-to-market, many research works on partially reconfigurable (PR) FPGA architectures have been conducted to fulfill the needs. Those have yet to fully provide a versatile framework to exploit the flexibility of PR such as hardware/software task migration and bitstream relocation; more importantly, the on-chip debug features to access all processors currently loaded in the system are compromised because of the lack of native-support from vendor tools. In this paper, a novel PR-HMPSoC architecture for dynamic FPGA-based embedded system is proposed to provide solutions for all of the above issues. The results from the experimental system consisting of one static Microblaze and three PR Microblaze/hardware accelerators connected by a Network-on-Chip show that the architecture is very promising with just *8%* reduction in operating frequency.**

***Index Terms*—FPGA, partial reconfiguration, multiprocessor, heterogeneous, debug, bitstream relocation, task migration**

I. Introduction

Nowadays, the demands for FPGA-based embedded systems with higher performance in terms of powerful computational ability and fast processing time are rising rapidly. Homogeneous multiprocessor systems may be able to cope with the requirements. However, according to Amdahl's law [1], increasing number of processors does not always translate to linear speedup because not all portions of the application can be parallelized. Employing heterogeneous platforms called Heterogeneous Multiprocessor Systems-on-Chip (HMPSoCs) with different dedicated hardware accelerators and specialized processors can improve the overall performance of system by effectively boosting up specific computational intensive and sequential tasks. In addition, these systems are developed on FPGA therefore they can be reprogrammed after manufacturing to apply the changes in requirements or bug fixes.

Nevertheless, the number of applications that need to be supported by one system is expanding. It is not practical to integrate dedicated hardware accelerators for each of the application onto a single FPGA chip because the FPGA resources are limited. More importantly, even if all applications can be incorporated in FPGA, they are not likely to operate at the same time, which makes the chip becomes underutilized most of the time. One solution is to develop multiple FPGA configurations, each of them supporting a group of applications called a use-case [2]. The particular configuration will be loaded to FPGA once needed. The disadvantages of this approach are the excessive storage requirement as the number of use-cases increases and the lengthy reconfiguration process for the whole FPGA.

Here is where the idea of dynamically loading and unloading modules at run time comes. This feature is called *partial reconfiguration* and is increasingly supported in a wide range of FPGA devices by vendors such as Xilinx and Altera. However, designing a HMPSoC that supports PR is not a trivial task; there are several design challenges that the architect must consider to build a flexible, high performance and reliable system as listed below.

1) **Task migration**. It is helpful not only in fault-tolerant system to move a running task and intermediate data from a faulty processing element (PE) to another to maintain the operation of the system, but also in preemptive environment, i.e, a low priority task can be suspended to relinquish the current PR region (PRR) to another high priority task and then resumed once the new task finishes. This feature is only applicable if the task manager has access to all internal memory instances of PE. The replication and shared memory strategy suggested by [3] can be used, but it is fairly restrictive and requires redundant memories.
2) **Runtime loading of processor executable code**. The purpose of PR system is to dynamically change the functionality of PEs by making use of PR feature. Therefore, it should be able to load the processor executable code at runtime when the PE is configured as processor. It can only be done if a micro-kernel is preloaded with the partial bitstream to load the task at run time by

TABLE I
PR-HMPSOC IN COMPARISON WITH PREVIOUS WORKS. IT OUTPERFORMS ALL PRIOR SYSTEMS IN MOST ASPECTS.

System	Multiprocessor	Par. Reconf.	Runtime loading *.elf*	On-chip Debug	Bitstream Relocation	Task Migration
Göhringer *et al.* [6]	yes	yes	yes	no	no	no
Beretta *et al.* [7]	yes	yes	no	no	yes	no
Cazzaniga *et al.* [8]	yes	yes	no	no	no	no
Navas *et al.* [9]	no	yes	no	no	no	no
PR-HMPSoC (this work)	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**

itself. This approach requires extended memory to store the micro-kernel beside the task. Another method is preloading the entire task with partial bitstream at design time which cannot be changed at run-time.

3) **Bitstream relocation**. It is necessary to minimize the bitstream storage and increase the flexibility of the system in which one particular partial bitstream of one module can be dynamically allocated to different locations on the chip. One of the requirements for this feature is uniform PRRs [4], [5] in terms of size, resource footprint and input/output interface with static region. Therefore, the structure of PEs should be made identical.
4) **Processor debug**. The effort in developing applications on multiprocessor system would be greatly improved if the developer can control all processors within the system to efficiently debug and monitor running applications. It helps increasing the reliability of system. In addition, the development environment for new architecture should be similar to the conventional system without PR feature to eliminate the overhead of learning process.

While there are many on-going research works proposing architectures and design methodologies for PR systems such as [6], [7], [8], [9], they are not able to provide solutions to handle all the aforementioned issues.

Contributions: In this paper, a novel architecture is proposed which offers a sophisticated framework to tackle all those requirements. This new architecture is developed for Xilinx FPGA. The experimental system is implemented on Xilinx Virtex 6 consists of one static Microblaze and three partially reconfigurable Microblaze processors/hardware accelerators connected by a simple Network-on-Chip.

The remaining of this paper is organized as follows. The state of the art of PR systems is discussed in Section II. The proposed architecture is presented in Section III, followed by experimental results in Section IV. Finally, the conclusion and future works are presented in Section V.

II. RELATED WORKS

In [6], Göhringer *et al.* introduces the RAMPSoC architecture in which every node can be anything, from microprocessor, microprocessor + co-processor to hardware accelerator. The PRRs for nodes are determined at design time and they are different from each other, therefore the system does not have the generic architecture, which is essential for bitstream relocation as mentioned in Section I. Besides, the paper suggests methods to send software executable file to a processor in PRR by using ICAP module [10] or transferring it via communication infrastructure. For the former approach, the partial bitstream and corresponding instruction data cannot be programmed simultaneously, because there is only one ICAP instance that can be used at a time. For the latter case, the author does not specify how it is implemented.

The system suggested by Beretta *et al.* [7] is divided into fixed size PRRs called *slots* with the same resources for bitstream relocation. Each slot can be configured with more than one processor core or hardware accelerator. However, the procedure of loading instruction codes for the processor cores in those slots is not mentioned.

Cazzaniga *et al.* [8] proposes a PR-MPSOC with the introduction of MARC (Multi-Adaptive Reconfigurable Core), the customizable reconfigurable processor, which is indeed a small version of a single processor system with one Microblaze processor and its peripherals are connected via standard PLB bus. However, the instruction code for MARC cannot be loaded at runtime because it can only be merged with the partial bitstream at design time after extracting the BRAM location in the corresponding *.ncd* file. This method unnecessarily increases the reconfiguration time of the MARC in case only instruction code is required to be updated while keeping the current MARC architecture as is.

The RecoBlock SoC platform, proposed in [9], is quite versatile and lightweight with flexible, reconfigurable interconnection between blocks and built-in buffers for data-driven streams. Still, the architecture targets only single processor systems with multiple reconfigurable hardware accelerators.

All of the above works do not provide processor debug facility which hinder the development process in a complex multiprocessor system. The task migration possibility is also not supported. Table I summarizes the features of aforementioned systems compared with PR-HMPSoC. As shown, PR-HMPSoC is the only one that provides all the features stated in Section I. In the following section, the details of proposed architecture are presented.

III. PROPOSED ARCHITECTURE

A. Overview

The overview of the proposed architecture is depicted in Fig.1. There are three interconnect planes in this system. One component can belong to more than one plane.

1) The first one is the Primary PLB bus plane which connects *Master Processor* (MstProc) with other standard Xilinx IPs such as ICAP [11], SysACE [12], MDM

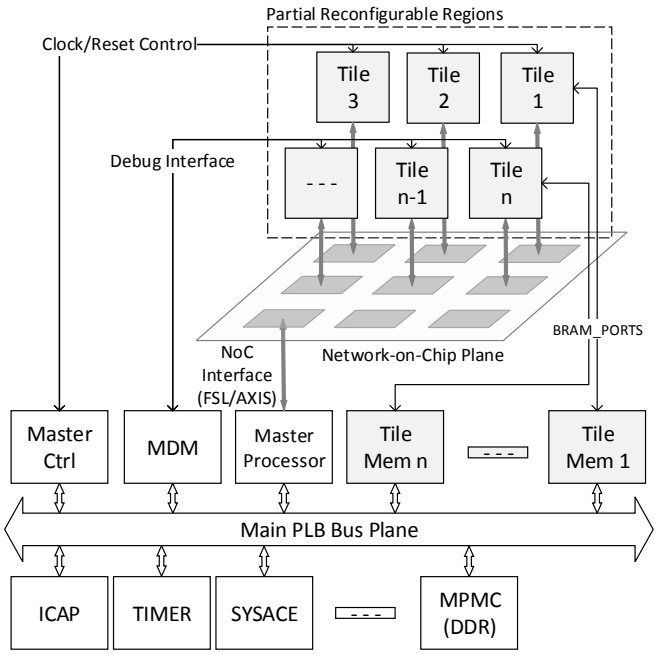

Fig. 1. Overview of the architecture. The connections from Master Controller and MDM to all Tiles are not shown for simplicity.

[13], MPMC [14], Timer [15], etc.; and compulsory PR-HMPSoC custom IPs: Master Controller (MstCtrl) and Tile Memories (TileMems). The MstProc can be either Xilinx Microblaze soft-core or ARM hard-core (on Zynq chip). This MstProc, as the name suggests, handles all the management operations of the system. Its role and PR-HMPSoC custom IPs are discussed in more detail in Section III-B. Besides, two IPs that must be instantiated to satisfy basic operations of the PR system are the *ICAP* module for partial bitstream reconfiguration and the external memory controller such as *SysACE* for accesses to compact flash card where partial bitstreams, processor executable files or any other input/output data reside. The MDM (for debugging), *MPMC* (for accesses to DDR-RAM) and Timer, etc. are optional and dependent on the initial requirements.

2) The second plane is the Network-on-Chip (NoC), which is the high-speed dedicated communication infrastructure between a set of *Tiles* and the MstProc. A Tile represents a PR PE which can be partially reconfigured at run-time to become either a processor or a hardware accelerator. Structure of Tile is considered in Section III-B. The NoC plane is used separately with the Primary PLB bus for two reasons:
   - First, most of applications mapped on HMSoCs are very computationally intensive and require significant amount of intermediate data transferred between PEs, or Tiles in this case. With dedicated communication medium, the data is not affected by the input/output traffic managed by the MstProc (to/from ICAP and SysACE) which are expected to occur frequently in practical systems with lots of different applications and configurations for Tiles. It increases the predictability and analyzability of the system performance, which are critical in designing real-time system.
   - Second, it provides freedoms in choosing any sophisticated architecture to suit bandwidth requirements, area constraints, power efficiency or dynamic reconfiguration without changing the rest of the system. The experimental system presented in Section IV utilizes the NoC proposed by [16].
3) The final plane is called Mixed plane. It consists of groups of directed *connections* from TileMems, MDM (optional) and MPMC (optional) to all predefined number of Tiles in NoC plane. The implementation of MDM module and how the processors configured in Tiles are accessed from the development environment are discussed in Section III-C. Beside NoC, this plane supplies alternative means of sharing and storing data for Tiles by the direct links to MPMC module which handles up to 8 ports. It is very useful in below scenarios.
   - **Shared memory model**. Two or more Tiles can share the common DDR locations and make use of NoC connections to synchronize with each other about the availability of data.
   - **Large temporary storage**. In addition to the dedicated TileMem for each Tile, the DDR in this case plays a role as a secondary memory which is bigger in size but slower in access time.

   However, it is assumed that there is a memory manager running on MstProc to allocate DDR memory regions to Tiles or they can be preassigned based on the initial requirements to avoid access conflicts.

B. Tile, TileMem, MstProc and MstCtrl

As previously defined in Section III-A, a Tile is a PR PE belonging to NoC plane. Each Tile is complemented by one TileMem, a static module attached to the Main PLB bus. The structure of Tile and TileMem are identical across all of their instances in the system and are illustrated in Fig.2. Tile is composed of one clock/reset facilitator (not shown in the figure) to handle the input clock/reset requests, and one PR module, PR_Wrapper, which is a *blackbox* as a rule of designing PR module [17]. The input/output interface of Tile is standardized with four groups of signals: clock/reset, debug (to MDM), NoC (FSL/AXIS stream interface) and PLB. TileMem is basically a wrapper of BRAM with one multiplexer to arbitrate accesses between MstProc and Tile (connected to TileMem through two BRAM Ports). It is possible to connect the PLB interface of Tile to the Main PLB bus, but this standard BRAM port allows flexibility in designing hardware inside the PR module, *PR_Wrapper*, because it is simpler to implement a hardware accelerator with direct and predictable links to the memory rather than dealing with other buses. The

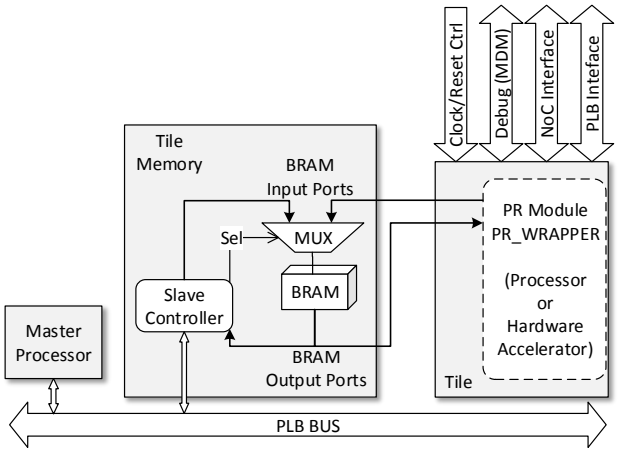

Fig. 2. Detailed implementation of Tile and Tile Memory. PR_Wrapper is PR module of Tile which is a *blackbox*.

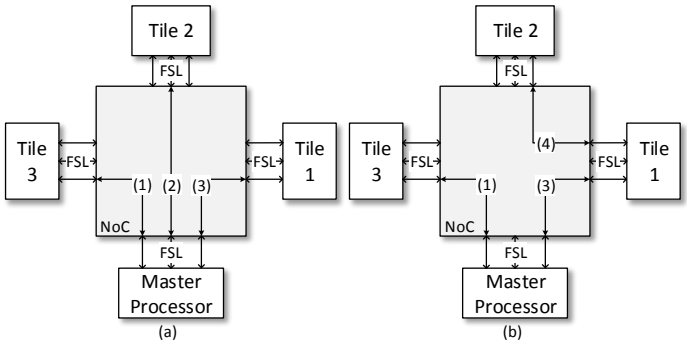

Fig. 3. Two communication scenarios for experiments. They are only applicable for NoC [16] used in the experimental system to determine the links at run-time. Any other NoCs can be used since PR-HMPSoC is not restricted to a specific interconnect.

Slave Controller in TileMem is implemented such that the multiplexer is transparent from the view of MstProc and Tile.

MstProc is the one that manipulates all the operations of the system. It controls the clock as well as reset signals of all Tiles by writing to registers in the MstCtrl. This is important because the PR modules inside Tiles have to be reset after being partially reconfigured. The clock enable signal is also useful when one Tile is not in use. If it is known in advance that the Tile will be reused in the near future, it is better to gate the clock than to reconfigure it with the *blank* partial bitstream. The MstProc also reads the corresponding partial bitstreams from external memory such as compact flash card via SysACE and sends them to ICAP to reconfigure the Tiles when new processors or hardware accelerators are required by a particular application. In case of loading processor executable files for Tiles, the MstProc only needs to transfer them to the corresponding TileMem via the Main PLB bus.

Despite the uniform look of the proposed architecture with identical instances of Tile, the heterogeneity of the system comes from the variants of PR_Wrapper. This generic architecture provides the possibility in configuring the partial bitstreams to any existing PRRs in the system that have the same resource footprints [4], [5]. However, this problem is left for future work.

C. Debugging

One advantage of PR-HMPSoC is that it offers debugging capability for Tiles in case they are configured as processor. If the processors are Microblaze and Xilinx MDM is preferred as debug module, no third-party tool is required. Besides, the number of debug-enabled Tiles depends solely on the Xilinx MDM, at the time of this paper, one MDM supports up to 32 Microblaze processors. Due to the informal characteristic of the architecture, Xilinx tools are not able to identify the existence of Microblaze instances. Therefore, the system description file exported by Xilinx XPS, *system.xml*, must be modified to assist Xilinx SDK in recognizing the correct system hierarchy. After that, the normal flow of developing embedded multiprocessor suggested by Xilinx can be followed to create application projects for each processor. The traditional debug practices can also be applied such as using the XMD console to communicate with processors via MDM.

D. Implementation Consideration

In the current implementation of loading the processor executable codes for Tiles, the MstProc accepts file format *.mem* converted by the *data2mem* tool [18] from the *.elf* files generated by Xilinx SDK. The MstProc parses the content of *.mem* file and transfers it to the desired TileMem via the Main PLB bus. Other approaches can also be considered because the MstProc is able to manipulate memory of Tile easily.

In addition, the MstCtrl only drives the clock/reset signals to all Tiles bases on the requests from MstProc. In future works, the MstCtrl may also act as a standalone hardware-based ICAP controller or utilize the enhanced ICAP modules proposed by [19], [20] to handle and speed up the partial reconfiguration tasks independently. Then, the MstProc will be able to copy processor executable files or initial data to the TileMems in parallel with the partial bitstreams which will be transferred by MstCtrl. In this way, the overhead of entire partial reconfiguration operation can be reduced drastically resulting in a shorter time needed to change applications. This is the purpose of keeping TileMems in the static region instead of putting them inside Tiles.

IV. Experiments

The effectiveness and flexibility of PR-HMPSoC is assessed by the experimental system with one Xilinx Microblaze soft-core as MstProc and three Tiles. The NoC proposed by [16] is employed in this experiment to connect MstProc and Tiles. This NoC can be reconfigured at run-time to change the links between communicating nodes. Therefore, two NoC configurations are created at design time for the experiments as shown in Fig.3. The partial bitstreams, processor execution codes are generated by Xilinx PlanAhead and Xilinx SDK version 14.4 respectively. In the experiments, all components operate at $75MHz$.

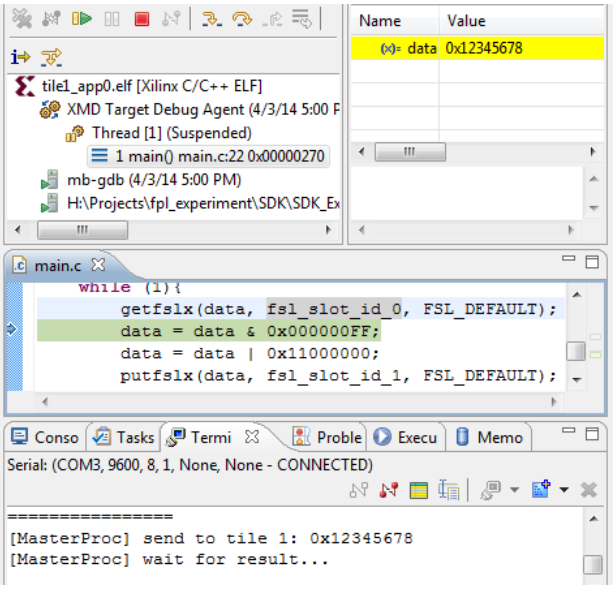

Fig. 4. Tile 1 is being debugged in Xilinx SDK. Any Tile configured with Microblaze can be recognized by Xilinx SDK debugger.

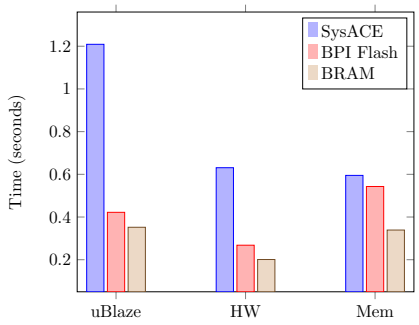

Fig. 5. The reconfiguration time of one Tile with different configurations: Microblaze (uBlaze), hardware accelerator (HW) and processor executable code (Mem). Their sizes are 410672 bytes, 206436 bytes and 11656 bytes respectively. The *SysACE*, *BPI Flash* and *BRAM* modules attached to Main PLB bus are used alternately to measure the times.

A. Scenario 1 — Debug Feature and Reconfiguration Time

In the first scenario, Fig.3a, MstProc communicates directly with all Tiles via the FSL interfaces supported by NoC. Each Tile is either Microblaze or hardware accelerator with the same functionality. Tile 1 is also debugged in the Xilinx SDK to show how easy and familiar it is for developers to work with PR-HMPSoC. The actual debug screen is captured in Fig.4. As seen, all debug functionalities provided by Xilinx SDK can be used with PR-HMPSoC: manipulating breakpoints, stepping through instructions, monitoring variables, etc.

The time spent by MstProc to reconfigure and load processor executable code for one Tile is also presented in Fig.5. The BRAM is actually used as a buffer to store configuration files read from SysACE or BPI Flash before being fed to the reconfiguration tasks. This method is 3 times faster than SysACE but can only achieve a 33% speedup compared to BPI Flash. However, the reconfiguration speed is rather slow

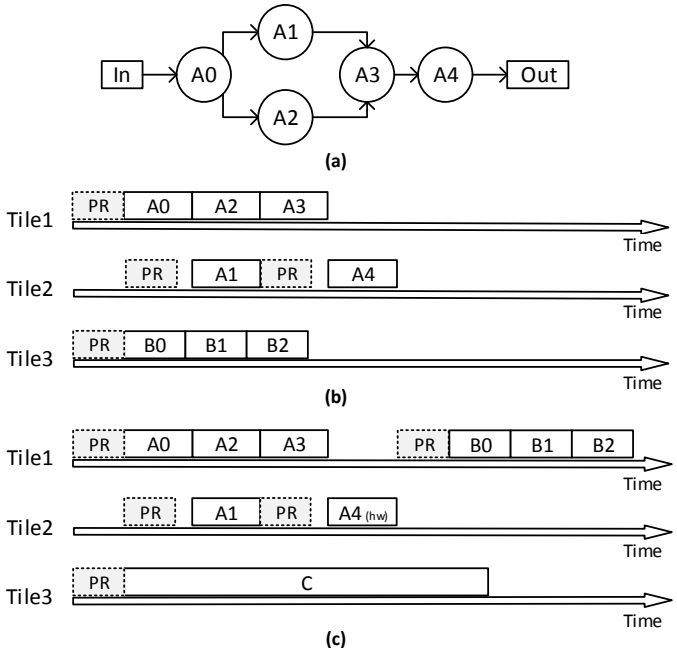

Fig. 6. There are three applications: *A*, *B* and *C*. The synthetic task graph of application *A* is given in (a). Two use-cases (b,c) are specified with different mappings for three Tiles. Task *A4* of application *A* is designed with two implementations: a Microblaze task and a pure hardware accelerator (denoted with *hw*). The length of tasks on time axis are for illustration purpose only and does not reflect the actual run-time.

due to the limitation of Microblaze which has been reported in [19]. A dedicated reconfiguration module will be considered in future works as previously discussed in Section III-D.

Regarding the processor executable code, the loading time is quite high despite the small file size. The reason is that the MstProc has to parse the *.mem* file before writing to TileMem which incurs nontrivial text processing to extract the data. However, as already stated in Section III-D, other methods can be used to shorten the process.

B. Scenario 2 — Multiple Applications and Use-cases

The NoC configuration for second scenario is shown in Fig.3b, MstProc communicates directly with Tile 1 and 3, while Tile 2 only works with Tile 1 as a co-processor. Fig.6 illustrates two use-cases for three applications in PR-HMPSoC. The processor executable code of application B can be used for both Tile 1 and 3 without having to compile it separately for each Tile at design time. The purpose of re-using Tile 1 is to demonstrate that a particular processor execution code can be freely moved around Tiles, thanks to their uniformity and the special architecture of PR-HMPSoC. This is also the basic idea of task migration. Yet, a more advanced technique in software layer is required to store and load the status registers of Microblaze to resume the operation. The partial reconfiguration tasks, *PR*, are defined to load corresponding partial bitstreams and processor executable codes. These tasks can be included in the original task graphs before using a proper task scheduler to determine the best mapping for

TABLE II
RESOURCE USAGE OF ONE TILE (CONFIGURED AS MICROBLAZE WITH LMB CONTROLLERS), TILEMEM (16KB OF MEMORY) AND MSTCTRL USED IN THE EXPERIMENTAL SYSTEM. THE LAST ROW REPRESENTS RESOURCE CONSUMPTION OF STATIC MICROBLAZE INCLUDING LMB CONTROLLERS AND 16KB OF MEMORY.

Module	Slice LUTs	Slice Register	LUTRam	BRAM
Tile	930	935	151	0
TileMem	71	93	0	4
MstCtrl	56	152	0	0
Microblaze	856	937	150	4

applications. Nonetheless, this issue is not discussed here and is left as future works.

C. Resource Usage and Operating Speed

Table II shows the resource usage of PR-HMPSoC custom IPs: Tile, TileMem and MstCtrl. As seen, the resource overhead of Tile plus TileMem compared with Microblaze is about 17%. This overhead comes from the extended features of TileMem (can be accessed from MstProc) and Tile (processes requests from MstCtrl and supports PR feature). The resources of hardware accelerators used in the experiments are not shown because they are significantly smaller than Microblaze. Another overhead is the size of the PRR in which the Tile resides. In this experiment, one PRR consists of 1110 slices which is about 20% more than the actual implementation requirement. This size of the PRR is chosen to not only reduce the time of mapping and place-and-route processes but also increase the operating speed of the system. The maximum frequency supported by the experimental system is $75.279MHz$. This is 8% lower than the maximum frequency of conventional system with four Microblaze processors, which is $81.960MHz$, because of the restricted placement constraints.

As presented above, the architecture is very promising with small overheads of extended components, however, these components contribute significantly to the increased versatility of the new multi-processor system over the conventional one even when the PR feature is not used. The actual unavoidable overhead is the size PRRs. In a practical system with many types of hardware accelerators and Tiles, setting all PRRs to the same size is not an efficient way of utilizing FPGA resources. In this case, the architect may consider varying sizes of PRRs to restrict the flexibility of bitstream relocation to smaller subsets of modules.

V. CONCLUSIONS AND FUTURE WORKS

This paper presents the architecture of PR-HMPSoC, a partially reconfigurable heterogeneous multiprocessor system-on-chip. It discusses many advantages of PR-HMPSoC such as the heterogeneous Tiles with homogeneous structures. This characteristic allows the possible implementation of task migration as well as bitstream relocation methods. Besides, unlike previous architectures, the debug features for all Microblaze processors in Tiles are retained in the familiar Xilinx SDK environment. It is the desirable feature in the development process of multiprocessor systems which is rather complicated.

The forthcoming works are designing a complete solution to generate the system configurations automatically, including finding optimal placements for PRRs and mappings for predefined sets of applications at design time. The bitstream relocation method will also be developed to increase the system flexibility and reduce the partial bitstream storage.

VI. ACKNOWLEDGMENTS

The authors would like to thank Sriram Vasudevan, Nanyang Technological University, Singapore and Rourab Paul, University of Calcutta, India for their contributions to this work.

REFERENCES

[1] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, spring joint computer conference*, pp. 483–485, ACM, 1967.

[2] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal, "Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 13, no. 3, p. 40, 2008.

[3] L. Gantel, S. Layouni, M. Benkhelifa, F. Verdier, and S. Chauvet, "Multiprocessor task migration implementation in a reconfigurable platform," in *Reconfigurable Computing and FPGAs, 2009. ReConFig'09. International Conference on*, pp. 362–367, IEEE, 2009.

[4] D. Koch, *Partial Reconfiguration on FPGAs: Architectures, Tools and Applications*, vol. 153. Springer, 2012.

[5] T. Drahonovsky, M. Rozkovec, and O. Novák, "Relocation of reconfigurable modules on Xilinx FPGA," in *Design and Diagnostics of Electronic Circuits & Systems (DDECS), 2013 IEEE 16th International Symposium on*, pp. 175–180, IEEE, 2013.

[6] D. Göhringer, M. Hübner, E. N. Zeutebouo, and J. Becker, "Operating system for runtime reconfigurable multiprocessor systems," *International Journal of Reconfigurable Computing*, vol. 2011, p. 3, 2011.

[7] I. Beretta, V. Rana, D. Atienza, and D. Sciuto, "A mapping flow for dynamically reconfigurable multi-core system-on-chip design," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 30, no. 8, pp. 1211–1224, 2011.

[8] A. Cazzaniga, G. Durelli, C. Pilato, D. Sciuto, and M. D. Santambrogio, "On the Development of a Runtime Reconfigurable Multicore System-on-Chip," in *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pp. 132–135, IEEE, 2012.

[9] B. Navas, I. Sander, and J. Öberg, "The RecoBlock SoC platform: a flexible array of reusable run-time-reconfigurable IP-blocks," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 833–838, EDA Consortium, 2013.

[10] O. Sander, L. Braun, M. Hübner, and J. Becker, "Data reallocation by exploiting FPGA configuration mechanisms," in *Reconfigurable Computing: Architectures, Tools and Applications*, pp. 312–317, Springer, 2008.

[11] Xilinx, *LogiCORE IP XPS HWICAP(v5.00a)*, 2010.

[12] Xilinx, *XPS SYSACE (System ACE) Interface Controller*, 2009.

[13] Xilinx, *Xilinx MicroBlaze Debug Module (MDM)*, 2012.

[14] Xilinx, *LogiCORE IP Multi-Port Memory Controller (MPMC)*, 2011.

[15] Xilinx, *LogiCORE IP XPS Timer/Counter (v1.02a)*, 2010.

[16] Z. J. Yang, A. Kumar, and Y. Ha, "An area-efficient dynamically reconfigurable spatial division multiplexing network-on-chip with static throughput guarantee," in *Field-Programmable Technology (FPT), 2010 International Conference on*, pp. 389–392, IEEE, 2010.

[17] Xilinx, *Xilinx Partial Reconfiguration User Guide*, 2013.

[18] Xilinx, *Xilinx Data2MEM User Guide*, 2009.

[19] M. Liu, W. Kuehn, Z. Lu, and A. Jantsch, "Run-time partial reconfiguration speed investigation and architectural design space exploration," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pp. 498–502, IEEE, 2009.

[20] S. G. Hansen, D. Koch, and J. Torresen, "High speed partial run-time reconfiguration using enhanced ICAP hard macro," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011*