Reconfigurable MPSoCs for On-Demand Computing

Linfeng Ye Jean-Philippe Diguet, Guy Gogniat
Lab-STICC, CNRS - Université Européenne de Bretagne/UBS, Lorient, France
linfeng.ye@univ-ubs.fr

Abstract – This paper presents a new approach in the domain of reconfigurable architectures. The objective is to automatically adapt a multiprocessor architecture according to application requirements. It is based on a generic model of architecture and a configuration management flow. Our method is illustrated with an instance of bi-processor architecture model (XPSoC-V2) implementing a MP3 decoder.

1 Introduction

Reconfigurable computing is not a new idea if we refer to Esrin paper in 1963 [1], where the main argument is already given: "obtaining a more effective architecture by means of specialization". Many multiprocessor and configurable machines have already been developed in the 90's, like ARMEN[2] for instance that was based on transputers, DSP and FPGAs. Obviously, things have changed with the integration progress on a single chip providing fast communication capabilities that were the main limit of previous attempts.

If we look back again in the past Von Neumann and al. brought the concept of instruction stream in the 40's, then Harvard architecture has been proposed in order to parallelize both streams. Finally, in the 60's the concept of the cache memory has been introduced to speed up stream outputs. A configuration stream as specified in Fig.1 can now be considered as the third kind of stream in the context of reconfigurable architectures. Logically, this new stream that can go by data and instruction must be speed up with a cache hierarchy considering voluminous files but moderate update frequencies.

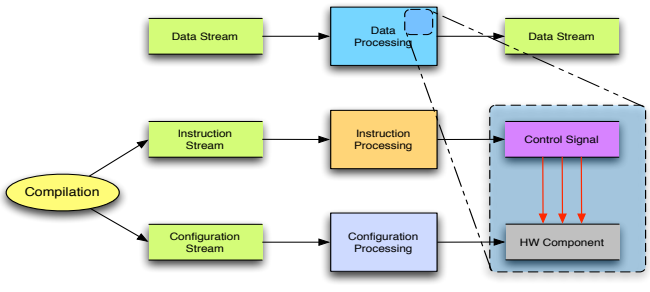

Fig. 1: Compute Processing in R-MPSoC

However, an efficient access to configuration files is not enough, some important issues remain, namely a programing model oriented towards reconfigurable architectures and the configuration decision. In this paper, we address the first point assuming we have an optimized configuration cache hierarchy.

Our methodology targets mass market applications in the domain of embedded systems (e.g. Mobile Internet Device) based on standard functions for which flexible hardware accelerators may be designed and dynamically uploaded according to application needs. In this paper, we present a methodology for the management of R-MPSoC, focusing especially on modeling aspects and mechanism of configuration processing. Our approach is a general Server-Manager-Worker model (or Configuration-Instruction-Data Processing) based on a XPSoC (X-Processor System-on-Chip) architecture, where:

- XServer: proceeds reconfiguration stream requests or initiates reconfiguration stream download
- XWorker: dedicated slave processor executing a single task for which it is specialized
- XManager: multi-tasks, observes the system, controls XWorkers and decides XWorker configurations at runtime

To achieve this goal, there are some problems to be solved:

1. Modeling of configurable architectures
2. Configuration mechanism and protocol
3. Decision policy based on constraints, user requests and availability of configurations
4. Cache policy for configuration processing

Today we have mainly addressed points 1 and 2 and simple solutions have been used for others points. Our models have been instantiated on a Xilinx FPGA providing dynamically reconfiguration capabilities.

The rest of the paper is organized into three sections. Section 2 compares this work to literature. In section 3, we provide a description of our R-MPSoC system model and the associated mechanism and protocol of the reconfiguration processing. In section 4 we present our configuration processing dataflow and in section 5 we describe our case study and implementation of XPSoC-V2. Finally, we draw some conclusions and perspectives.

2 Related Work

A lot of successful researches have been done in the domain of reconfigurable embedded systems, most of them are focused on architecture design. Two specific aspects are related to our concerns, the first one is the configuration management. In [3] is presented an embedded Linux as a platform for the management of dynamic reconfiguration of systems-on-chip, authors show how partial reconfiguration are achieved with a single line of Linux shell script. Angermeier et al. [4] use dynamic partial reconfiguration to implement Erlangen Slot Machine (ESM) architecture, which is to accelerate the application development as well as the research in the area of partially reconfigurable hardware. The advantage of the ESM platform is its unique slot based architecture, which allows the slots to be used and reconfigured independently of each other. Eustache et al. [5] present a safe and efficient solution to manage asynchronous configurations of dynamically reconfigurable systems-on-chip. Bomel et al. present a networked lightweight and partially reconfigurable platform assisted by a remote bitstreams server and in [6] is detailed a partial bitstream ultra-fast downloading process through a standard Ethernet network, with a sustained rate of 80 Mbits/s over Ethernet 100 Mbit/s.

The second point is the architecture modeling required by our approach. In [7] is described a model of reconfigurable architecture based on SystemC and Python script for the evaluation of different scheduling policies regarding reconfiguration. The main component is the configuration manager that can access to a list of configurable functions to be placed on eFPGA independent modules. This work is interesting but not directly connected to implementation, more over it doesn't fit with our architecture model based on reconfigurable processors and the necessity to have configuration ID.

3 R-MPSoC Modeling

To describe our approach, we need to define the elements that constitute our model of an embedded R-MPSoC. Fig.2 shows how this model is progressively specialized in order to provide a given reconfigurable architecture. We start from a meta-model (M01) of the architecture composed of a manager processor controlling some slave processors with a given number of co-processors, and various other components such as shared memories, IP and peripherals. Then an instance of the model (M11) is specified by the designer, in this case it is made of one manager processor and two slave processors, each SP can then be dynamically configured by means of two coprocessors.

A R-MPSoC is composed of three types of binary data and processing: Data, Instruction and Configuration streams and associated processing, with the following definitions:

- **Data stream:** data to be processed by Xmanager (mainly control), Xworker or IP (mainly dataflow)
- **Instruction stream:** sequence of processor (Xmanager, Xworker, IPs) instructions (software)
- **Configuration stream:** sequence of hardware configuration data (e.g. FPGA bitstream)
- **Data processing:** is any process used to compute a data stream.
- **Instruction processing:** is any process, that converts instruction into hardware control signals.
- **Configuration processing:** is any process, that configures data path, there are two types configuration processing: static configuration processing and dynamic configuration processing.
- **Function descriptor:** meta-data (unique function ID: FID) added to process code, by sofware designers, in order to provide Xmanager with information about process functions that can benefit from Hardware implementation to obtain significant speed-up, typically these functions are called in loop nests (e.g. DCT)
- **XManager** is a GPP interfacing the whole R-MPSoC system with remote server and devices, it is mainly responsible for the coordination of the system, software and input/output management, generally speaking it will be in charge of data processing, which doesn't require any specific hardware.
- **XWorker** is a single task processor specialized to execute a given process with coprocessors implementing function specified with FID.

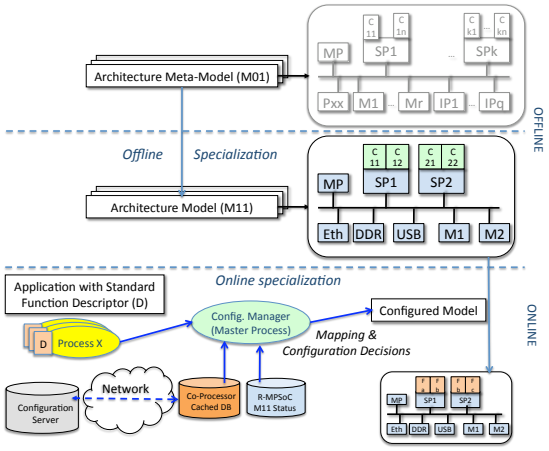

FIG. 2: Global model view where MP: Master Processor, SP: Slave processor, C: Reconfigurable co-processor and F: standard function implemented with C

4 Configuration processing

Our R-MPSoC has two kinds of subsystems: XManager and XWorkers, the first one is a GPP configuring and controlling the second ones. We've adopted an approach comparable to the *Legal Representative* detailled in [5] or *ghost processes* in BORPH [8]. Namely, the granularity of a XWorker job is a Xmanager Linux process that is pending for XWorker results when such a configuration has been decided. Typically XWorker jobs can be MP3 Encoding or Decodering, MPEG decoding. Linux is the manager OS and new services and configuration manager is progressively added to the global model in order to implement the reconfiguration flow, a simple R-MPSoC (bi-processor, single task) reconfiguration flow is described in Fig.3.

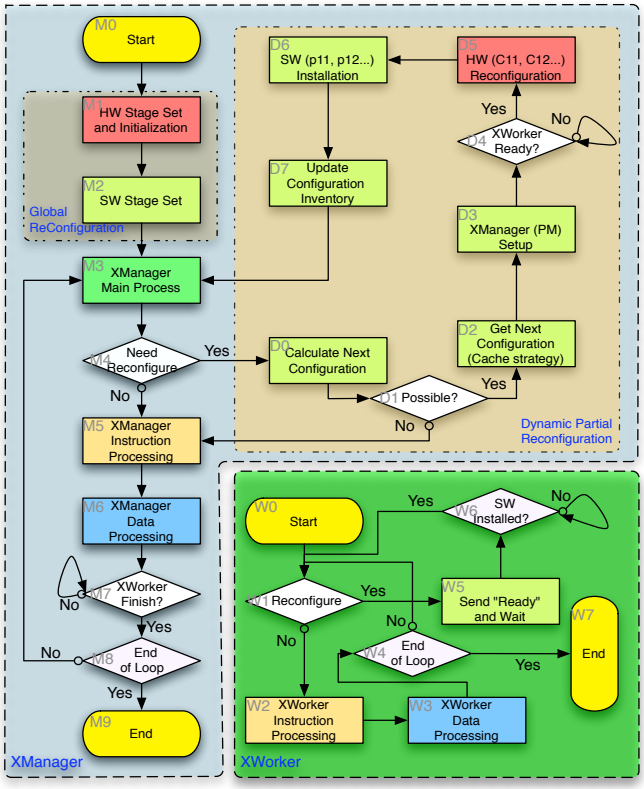

FIG. 3: R-MPSoC system flow chart

The R-MPSoC system automatically loads first hardware architecture (configuration) from non-volatile memory (e.g. Flash) after the power-on, and then loads boot binaries for XManager (e.g. Operating System) and for XWorker (e.g. boot application waiting for new configuration). XManager launches the main process, which is responsible for managing all the tasks and dynamic reconfiguration module. If no specific configuration is obtained or required, XManager performs a regular instruction processing. If XManager determines a new configuration: computes allocated resources and XWorker positions according to the status of the system and the configuration repository, in order to find out the most suitable configuration. XManager downloads the next configuration stream to memory and updates the cache (global server, local server, hard disk, memory, etc) according to the strategy (Temporal Locality, Spatial Locality, Least recently used, Most recently used, etc). XManager can perform hardware reconfiguration when it receives a "READY" signal from XWorker, then XManager performs software installation for new configuration by writing the new application in a local memory and sends a reset signal to XWorker. Finally XWorker boots on the previously implemented application. XManager updates the configuration repository and relaunches the task by using a new configuration.

If XManager cannot find a configuration more suitable than the current configuration, it controls one XServer to execute the process (it is goes to *M5* in Fig.3). Then XManager informs XWorker, which is concerned, about the upcoming reconfiguration, and then starts XWorker. After finishing all the missions, the system goes into hibernation or shutdowns.

5 Case study and Implementation

5.1 XPSoC-V2 hardware architecture

XPSoC-V2 (see Fig.4) is the name of the second instance of our R-MPSoC model developed within the framework discussed above. It targets audio processing applications such as MP3 encoder/decoder, OGG encoder/decoder and VoIP softphone. It is based on a bi-processor model including one XManager and one XWorker with two reconfigurable coprocessors. It also implements an Ethernet controller, an UART, a DDR SDRAM controller, a Compact Flash controller, a HW Mutex component, a shared memory and a HWICAP controller.

The XManager uses the Microblaze softcore from Xilinx and acts as a GPP, the XWorker uses the Microblaze softcore with two coprocessors reconfigurables and acts as a configurable DSP (Digital Signal Processor). The design of coprocessors is strongly dependent on the application and the available resources on the reconfigurable part. The Mutex component is used as a peripheral to coordinate CPUs (XManager, XWorkers) safe accesses to shared peripherals or memories. Coprocessors are connected to the XWorker by FSL (Fast Simplex Link) and accelerator critical functions, the Xilinx BusMacros are used to realize a direct communication between XWorker and coprocessor modules, providing fixed communication channels that help to keep the signal integrity during hardware reconfiguration.

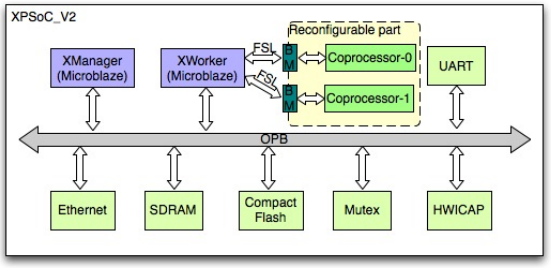

FIG. 4: XPSoC-V2 hardware architecture

5.2 XPSoC-V2 software architecture

XPSoC-V2 software splits into three parts (see Fig.5): Applications with meta-data, Xmanager middleware implementing the configuration manager process and services and XWorker middleware in charge of XManager / XWorkers communications and synchronization.

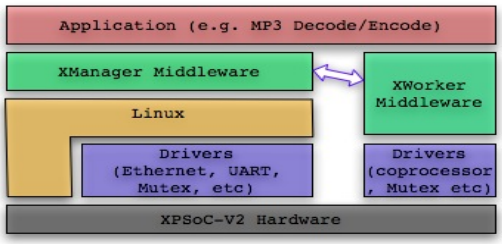

FIG. 5: XPSoC-V2 software architecture

5.3 Implementation results

The XPSoC-V2 architecture implements one XManager and one XWorker on a Xilinx ML410 FPGA. Place and route results leads to an area occupation of 32% for logic block slices, 23% for RAM blocks and 13% for DSP. The XManager runs a petalinux [9] operating system, and the granularity of task is the process level. In other words, the XWorker can run an application (e.g. MP3 decode) as a standalone program. Two types of coprocessors have been generated for the first test : an IMDCT coprocessor and a MUL16 coprocessor. The IMDCT coprocessor implements a 18x36 IMDCT (Inverse Modified Discrete Cosine Transform), and the MUL16 coprocessor implements a long long multiply and shift by 16.

The result of this simple demonstration is shown in 1:

TAB. 1: Mp3 decoding on XPSoC-V2

	SW	**HW-MUL16**	**HW-IMDCT**
Slice utilization	0	161/ 25280	482/25280
DSP utilization	0	4/128	4/128
LUT utilization	0	86/50560	521/50560
Time execution(s)	307.2	232.9	167.7

6 Conclusion

We defend the idea that future embedded systems can be dynamically built with standard configuration to be downloaded through a cache-based network connection. Consequently we now have to developed architecture models and methodologies adapted to this new design perspective. This paper describes a general approach of reconfigurable multi-processors embedded systems based on a generic modeling and an associated configuration processing flow. We prove our solution with a bi-processor reconfigurable system running PetaLinux on XILINX ML410 board. This is the first step and some work remains to be done. Current research task are focusing on the cache strategy, the configuration decision policy and the formalization of the configuration library.

References

[1] G.Estrin, B.Bussell, R.Turn, and J.Bibb, "Parallel processing in a restructurable computer system," *IEEE Transactions on Electronic Computers*, no. 6, pp. 747–755, Dec. 1963.

[2] J. Champeau and et al., "Flexible parallel fpga-based architectures with armen," in *Proc. Twenty-Seventh Hawaii Internation Conference on System Sciences Vol. I: Architecture*, vol. 1, 4–7 Jan. 1994, pp. 105–113.

[3] J. A. N. W. Williams, "Embedded linux as a platform for dynamically self-reconfiguring systems-on-chip," in *The Int. Conf. on Engineering of Reconfigurable Systems and Algorithms (ERSA)*, Las Vegas, USA, 2004.

[4] J. Angermeier and et al., "Spp1148 booth: Fine grain reconfigurable architectures," in *Proc. International Conference on Field Programmable Logic and Applications FPL 2008*, 8–10 Sept. 2008, pp. 348–348.

[5] Y. Eustache and J.-P. Diguet, "Reconguration management in the context of rtos-based hw/sw embedded systems," *EURASIP Journal on Embedded Systems*, vol. 2008, 2008.

[6] P. Bomel, J. Crenne, L. Ye, J.-P. Diguet, and G. Gogniat, "Ultra-fast downloading of partial bitstreams through ethernet," 2009.

[7] G. Beltrame, L. Fossati, and D. Sciuto, "High-level modeling and exploration of reconfigurable mpsocs," in *Proc. NASA/ESA Conf. on Adaptive Hardware and Systems AHS '08*, 22–25 June 2008, pp. 330–337.

[8] H. K.-H. So and R. W. Brodersen, "Improving usability of fpga-based reconfigurable computers through operating system support," in *Proc. International Conference on Field Programmable Logic and Applications FPL '06*, 28–30 Aug. 2006, pp. 1–6.

[9] Petalinux http://developer.petalogix.com/.