Configuration server for self-adaptive architectures

Linfeng Ye, Jean-Philippe Diguet, Guy Gogniat
Lab-STICC, CNRS - European University of Brittany/UBS, Lorient, France
linfeng.ye@univ-ubs.fr

Abstract

This paper presents a network-based solution to efficiently manage configurations of reconfigurable Multiprocessor system on chip. We first define our R-MPSoC architecture and the concept of on-the-fly reconfiguration management. Then two issues are discussed and some solutions are presented: (1) formalization of configuration protocol and (2) synchronization of configuration. The second point mainly addresses the issue of on-the-fly reconfiguration flow and the way a process accepts a new configuration. Finally, we present the global method and give several implementation results from a MP3 player case study.

1. Introduction

The key argument for reconfigurable architectures is probably the capacity of specialization that enables to improve both energy efficiency and performances. An infinity of architectural solutions is then possible to dynamically adapt architectures to applications, the main issue is the storage of configurations that remains voluminous in the case of current FPGAs.

If we consider such architectures in the context of embedded systems, which are now usually connected somehow to a network, then a new kind of solution appears. Actually networks can be used in order to feed on-demand reconfigurable embedded systems with relevant configurations. It means that a configuration stream can now be considered as the third kind of stream, besides data and instructions, in the context of reconfigurable architectures. Consequently it may be also speed up with a cache hierarchy considering voluminous files but moderate update frequencies.

On the application side of embedded systems, we can observe the same evolution in terms of complexity. Multiple and heterogeneous applications can share resources and stress hardware in different ways such as typical networking, signal/image processing, cryptography functions. In case of personal devices, 3D graphics, display and GUI can be added. An OS can then become a necessity for hardware abstraction and resources allocation.

Both architecture and application evolutions lead to highly dynamic and data or context dependent behaviors of embedded systems. In these changing conditions, the problem of optimization, regarding for instance energy efficiency, can hardly be handled without any configuration considerations.

Finally, any design methodology targeting embedded systems is strongly constrained by development cost. The use of IPs and standard API are tracks to reduce the effort cost, this option must also be considered in the domain of reconfigurable architectures. This approach makes sense in the domain of mass market products and ambient intelligence, based on standard set of applications and basic functions.

In this context, self-adaptivity applied on well-defined architecture models, is a promising way to solve the question of optimization at run-time. The first condition is the availability of dynamically reconfigurable architectures. We can reasonably make the assumption that such architectures, which already exist, will be more efficient in a near future in terms of power and reconfiguration time. The second condition is on-line decision capability.

Given this situation and the assumption that a majority of embedded systems will be somehow connected to a network, we propose in this paper an overview of a global approach for the design of self-adaptive systems targeting effective architectures according to applications demands. The methodology is first locally based on a combination of static and dynamic configuration decisions for both hardware and software aspects, the objective is to get a trade-off between decision complexity and configuration storage cost. Secondly, the idea is to increase the local configuration space by means of a shared and distributed hierarchy of configuration caches. This approach is based on two assumptions. First some generic architecture models have been previously defined and secondly applications are based on an intensive use of an evolutionary library of API and standard coprocessors or IPs.

The remainder of the paper is organized as follows. Section 2 presents related work and background of adaptive embedded systems. In Section 3, we present the concept of on-the-fly reconfigurable MPSoC. Section 4 discusses about the network-based configuration protocol. We describe our MP3 case study and its implementation on XPSoC-V2 in section 5. Finally, we conclude.

2. Related Work

A lot of successful researches have been done in the domain of adaptive embedded systems, most of them are focused on architecture design. In [6], the association between algorithmic and architectural views is relevant for H-264 implementation, however the hardware controller remains specific and local. In [8] is presented an embedded Linux as a platform for the management of dynamic reconfiguration of system-on-chip, authors show how partial reconfiguration is achieved with a single line of Linux shell script. Angermeier et al. [2] use dynamic partial reconfiguration to implement the Erlangen Slot Machine (ESM) architecture, which accelerates the application development flow 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. Manet et al. [7] present a custom OPB-ICAP working at 2.8 Gbit/s (Maximum theoretical bandwidth: 2.98 Gbits/s) on the Virtex-4. Eustache et al. [4] present a safe and efficient solution to manage asynchronous configurations of dynamically reconfigurable system-on-chip. Bomel et al. present a networked lightweight and partially reconfigurable platform assisted by a remote bitstreams server and in [9] 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.

In [3] 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 have 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. The current state of the art shows that dynamically reconfigurable coarse-grain architectures have been already proposed, but there is a lack of contributions dealing with configuration server. However this points is essential to allow an efficient deployment of reconfigurable architectures.

3 On-the-fly reconfigurable MPSoC

3.1 R-MPSoC architecture

The R-MPSoC, defined in this paper, is a Multi-Processor System-on-Chip composed of a master processor and not less than one slave processor and one reconfigurable component. The R-MPSoC is composed of three types of binary data and processing: Data, Instruction and Configuration streams and associated processing. The configuration stream is a sequence of hardware configuration data (e.g. FPGA bitstream). We consider there are two types of reconfiguration stream in our R-MPSoC model: Static reconfiguration stream and Dynamic reconfiguration stream. As shown in Fig.1, we present a Master / Slave R-MPSoC architecture. In this system, the *dynamic* reconfiguration stream can be performed "on-the-fly", and the *static* stream can be performed only by a global reconfiguration.

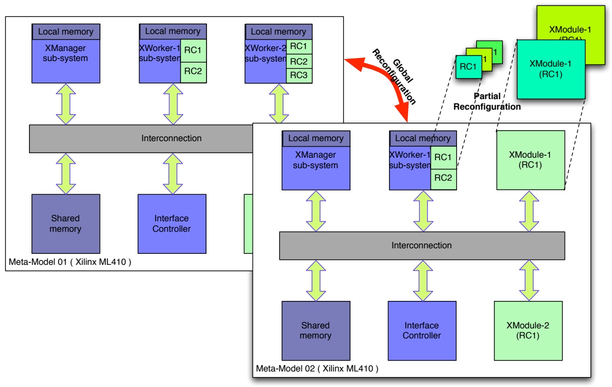

Figure 1. MPSoC Reconfiguration

We consider an architecture meta-model, which fits with a large set of embedded systems. The R-MPSoC model has two kinds of subsystems: XManager and XWorkers, the first one is a GPP configuring and controlling the second ones, which can be configurable soft cores (e.g. MicroBlaze) or dedicated accelerators. The life of an R-MPSoC begins with the choice of an architecture model composed of a XManager subsystem (XManager) controlling a given number of configurable XWorkers. For instance the Xilinx-ML410-MM02 (Fig.1) model includes three XWorkers subsystems: one XWorker processor with two reconfigurable co-processors (XWorker-1) and two independent reconfigurable components (XModule-1 and XModule-2), and various other components such as shared memories, IP and peripherals. At system boot, the boot loader loads the first hardware configuration (Xilinx-ML410-MM02 full configuration bitstream) and loads the software binary (e.g. Petalinux operating system) for XManager subsystem and for

XWorker subsystems (e.g. a lightweight operating system kernel). The XManager implements an operating system managing a set of processes. Processes communicate through message passing for synchronization and shared memories for data, and they can have various possible implementations in Hardware and / or in Software, which correspond to different cost / performance trade-offs.

3.2 On-the-fly reconfiguration and synchronization

On-the-fly reconfiguration is a style of reconfiguration in which the manager modifies the program and / or the hardware configuration while the application is running, without stopping or restarting. Our approach is based on the assumption that application bottlenecks have been first identified after a profiling step at design time. Thus, an application is specified with a list of functions that may be efficiently implemented with hardware coprocessors if speed up is required. The configuration of such an application running on a XWorker, is controlled by the XManager through a shared memory where global variables indicate to the XWorker the implementation of key functions. The XWorker regularly checks if a new configuration is available. If a hardware implementation is indicated then a memory or a FIFO address is given, in this case the XWorker won't use a software call but will use a standard HW/SW interface to send data to the coprocessor.

Thus two main steps may be considered for self-adaptive systems, the first one is the decision and the second one the reconfiguration process. Both are based on the availability of a given set of hardware configurations, this paper focuses on this specific point. An example of decision implementation can be found in [5] for an architecture model based on a GPP with specific hardware modules. The reconfiguration process is based on a set of APIs to communicate transparently with hardware or software tasks through an Unified Configuration and Communication Interface (UCCI).

Our proposition is based on the idea that configuration servers may deliver, through a hierarchy of caches, architectures configuration and application binaries for different kinds of architecture models. Thus, given architecture models and a set of standard applications, some configuration servers can deliver IPs and coprocessors to be implemented dynamically according to a list of standard functions.

Finally, the engineer in charge of system integration, can generate the solution file as follows:

- full_*.bit : R-MPSoC global hardware configuration, including an architecture model and all reconfigurable components.
- pr_*.bit : partial hardware configuration
- XManager.bin : Software binary for XManager
- XWorker-N.bin : Software binary for XWorker-N
- application.bin : Application binary
- application.prl : Application hardware partial reconfiguration list

In our approach, the granularity of a XWorker job can be a XManager Linux process or thread pending for XWorker results when such a configuration has been decided. Typically XWorker jobs can be speech, audio, image processing, video coding, network processing or security applications such as encryption. The R-MPSoC system automatically loads first hardware configuration (full_*.bit) from non-volatile memory (e.g. CompactFlash, hard disk) after the power-up, and then loads XManager.bin for XManager and loads XWorker-*.bin for XWorker. XManager launches the main process, which is responsible for managing all the tasks and dynamic reconfiguration of modules. If no specific configuration is obtained or required, XManager performs a regular instruction processing. If XManager determines a new configuration: it computes the allocated resources and XWorker positions according to the status of the system and the configuration.

XManager downloads the next configuration stream to memory and updates the cache (global server, local server, hard disk, etc) according to the cache strategy. XManager can perform on-the-fly reconfiguration. In this case it first deactivates the target reconfigurable component (RC), then XManager updates the table of reconfigurable component and informs XWorker that a hardware implementation is available for a given standard function. Finally XWorker executes program and tests its configuration with a given rate $1/N_i$ before each critical section of the application (usually a loop nest), by accessing to a shared memory where the XManager updates variables indicating if the associated reconfigurable component is or not configured. The master can also fix the configuration test rate (N) according to the estimated downloading delay.

For each XWorker, a configuration table is defined in a memory space shared with the Xmanager as shown partially in Fig.2. The first record provides global parameters such as architecture model (XMID), Xworker status and input and output addresses and sizes. Then a new record is added for each reconfigurable place (for instance a co-processor). From a synchronization point of view, the main information are firstly the FUID (FUnction ID), which is a unique identifier of a standard function that also depends on I/O data formats. Then, there are two synchronization bits. The ENable bit is written by Xmanager, it indicates to the XWorker that a hardwired function is ready and can be used for next task execution. The Done bit is written by the Xworker, it indicates to the Xmanager that the coprocessor is no more used by the current task. In the case of a R-MPSoC composed of one Xmanager and a Xworker with configurable

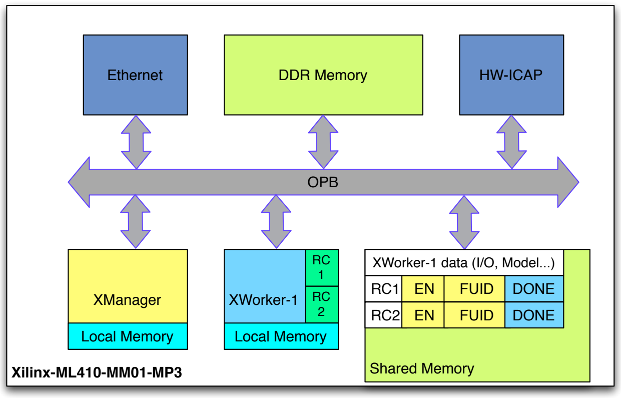

Figure 2. On-the-fly Reconfiguration

co-processors, we obtain a on-the-fly configuration flow described hereafter.

1. XManager determines, through a decision daemon process, if a new configuration is required for XWorker-N; for instance if a new function must or not be implemented in coprocessor-M. Such a decision depends also on the availability and the access delay to configuration files in local memories or on remote servers. If a reconfiguration is required, then the on-the-fly reconfiguration flow starts, it is described in steps 2 to 7.

2. First XManager indicates to XWorker-N, by setting “EN” bit to “0”, that current function implemented on coprocessor-M won’t be available as a hardware version when the coprocessor will be released by the current task. If the bit “Done” of the XWorker-N coprocessor-M is set to 1, then co-processor is unused and the flow jumps to step 5; Otherwise the Xworker switches to a software version as soon as it can, regarding constraints related to task granularity. If necessary, the Xmanager loads configuration files in a local memory (e.g. DDR) from flash memory or remote configuration servers.

3. XWorker executes the program, and tests its configuration with a given update rate compliant with the task granularity by reading the bit ”EN” of coprocessor-M. If coprocessor-M is disabled (EN=”0”), the Xworker cannot used the hardware function anymore. It informs XManager by setting ”Done” bit to ”1” that the coprocessor is free for reconfiguration.

4. XManager tests the bit ”Done” of the XWorker-N coprocessor-M, if it is set to“1”, then the configuration starts in step 5.

5. Xmanagers copies the configuration from DDR Memory to HW-ICAP and finally performs dynamic partial reconfiguration. Depending on the size of the configuration file, different iterations might be necessary.

6. XManager updates the ”FUID” bits of coprocessor-M to indicate the new function available as a hardware version and then enables the XWorker-N coprocessor-M by setting ”enable” bit to 1.

7. XWorker tests its configuration by reading the bit ”EN” of the coprocessor-M, if it is set to ”1”, function calls corresponding to the FUID will run with the coprocessor-M and Done is set to ”0”.’

In this scenario, we assume that XManager manages a list of used standard functions, which is provided by the application designer (application.prl). In practice, this list of all available hardware accelerator modules or coprocessors for an application binary, is the result of the program profiling analysis based on the official standard function lists which is maintained by configuration suppliers.

4 Network-based configuration protocol

A lot of future applications, based on distributed embedded systems, are expected in the domain of intelligent environment or transportation and in the context of nomadic devices. It means that wired and wireless local area networks can be considered as an available solution for delivering configurations. Given this assumption, we have implemented different architectures providing a networked-reconfiguration service for Xilinx FPGA. In our previous work [9], the objective was to implement partial and dynamic reconfiguration directly from a remote and local server. In other words, we can bypass the DDR level and the hard disk level of the cache hierarchy when the requested configuration is not available on board. We propose now a general and formalized approach based on the idea that an embedded reconfigurable system can efficiently download a set of configurations from global servers according to application requirements (e.g. Video Encode IPs supplier, Multimedia IPs suppliers). In the domain of open-source projects, it means that current available programs could be extended to hardware components in order to provide designers with infinite possibilities of (re)configurations and upgrades, it also opens very exciting perspectives in terms of collaborative researches. However, hardware components must be classified and organized to make this ”on-demand computing” service reliable in the domain of reconfigurable computing. Meta-modeling can provide the adapted framework to such a standardization effort. Thus, depending on technologies, devices families and architecture Meta-Models, different solutions can be proposed. If we consider for instance R-MPSoC based on Xilinx FPGA and Microblaze cores, we can propose a meta-model from which can be derived various architecture models.

We propose in this paper an overview (Fig.3) of a global approach for the Network-based self-adaptive systems targeting effective architectures according to application demands. This approach is based on two assumptions. First some generic architecture models have been previously defined and secondly applications are based on an intensive use of an evolutionary library of API and standard coprocessors or IPs. The main idea is to reduce the design space to be explored at run-time in order to introduce short and low cost decision overheads. Thus, the approach consists first in loading a reduced pre-defined set of configurations used for fast local adaptations according to context and data variations. Then, this set can be updated at run-time through a network connection if better configurations are necessary or if new applications, requiring new configurations, are started.

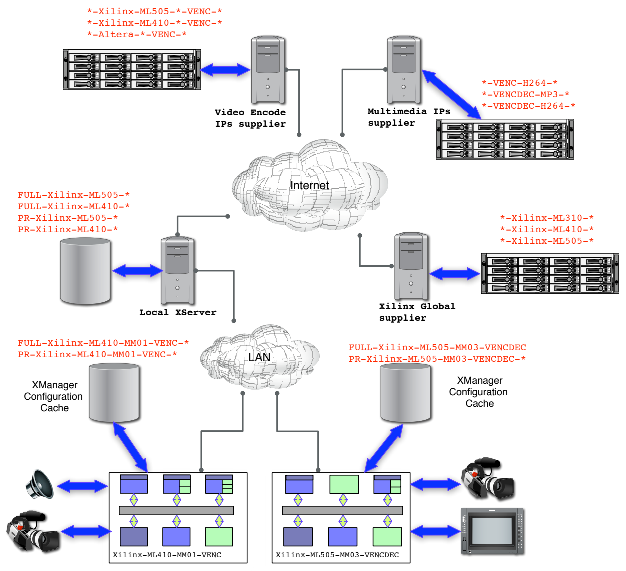

Figure 3. Network-based self-adaptive systems

As shown in Fig.3, we propose a global configuration framework based on local and global servers. The update of configurable systems is based on a client/server protocol over an existing network protocol such as TCP/IP, where a request is formalized with the following information:

- CSID: Configuration Supplier ID, (Xilinx, IBM, 3rd party, University, ..);
- HSID: Hardware Supplier ID, (Xilinx, Altera, ST, Atmel, ..), the reconfigurable device provided;
- HRID: Hardware Reference ID, (ML410, ML505, ..), the board identifier ;
- MMID: generic Architecture Meta-Model ID (MM01, MM05, ..), to be implemented on various targets;
- XMID: XModule or XWorker ID (XW01, XW02, ..), according to MMID parameters;
- IFIP: Reconfigurable IP Interface Protocol (FSL, OPB, PLB, AHB, ..), according to MMID parameters;
- IFID: Reconfigurable IP Place (FSL-i, OPB-j, ..), according to MMID parameters;
- FAMI: Algorithm family (Video, Audio, Security, Network, Digital communications, ..);
- FUID: reconfigurable IP Function ID (IMDCT, ME, Turbo-C, ..);
- VERS: Reconfigurable IP Function version;
- FREQ: Reconfigurable IP clock Frequency.

Note that these data should be first registered within a common database, our approach is based on this kind of distribution model. Thus a reconfigurable and networked embedded system can send a request for reconfiguration files according to application needs. When received, configuration files are stored locally within DDR or Flash memories and available for on-the-fly reconfiguration.

5 MP3 case study

5.1 Architecture model

XPSoC-V2 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 architecture 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 Hardware Mutex component and a HW-ICAP controller.

The XManager is implemented as a Microblaze softcore from Xilinx and acts as a GPP, the XWorker uses the Microblaze softcore with two reconfigurable coprocessors and acts as a Single-Task reconfigurable 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), 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.

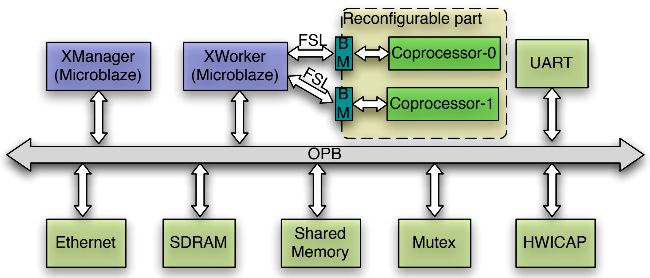

Figure 4. XPSoC-V2 hardware architecture

5.2 Implementation results

On a Xilinx ML410 (Virtex4 FX60) FPGA, the implementation of XPSoC-V2 leads to an area occupation of 32% for logic block slices, 23% for RAM blocks and 13% for DSP. The XManager runs a petalinux [1] 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. Three types of coprocessors have been generated for this test: a DCT32 coprocessor, a IMDCT coprocessor and a MUL16 coprocessor. The DCT coprocessor consumes 32 words at the input and provides 16 words of 64 bits at the output. 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 demonstrator is shown below:

	SW	HW-MUL16	HW-IMDCT	HW-DCT32
Slice utilization	0	161/ 25280	482/25280	569/25280
DSP utilization	0	4/128	4/128	8/128
LUT utilization	0	86/50560	521/50560	526/50560
Time execution (seconds)	105.0	91.1	71.1	61.6

Figure 5. Mp3 decoding on XPSoC-V2

In the first column the whole application is implemented in software so no additional hardware is required. In that case the execution time is 105 s. The second column details the architecture containing a MUL16 coprocessor which provides an improvement of 13% compared to the software only version. For this architecture the additional hardware cost for the coprocessor is only 161 (out of 25280) slices, 4 DSP blocks and 86 LUT. The last two columns describes two architectures containing a DCT as a coprocessor. It can be noticed that both solutions provide an improvement (respectively 32% and 41%) of the execution time.

5.3 Applied Networked Reconfiguration

We have tested our networked reconfiguration scheme on XPSoC-V2 with a MP3 decoder application. In this example we set the granularity of reconfiguration to the level of the whole application, it means that a new hardware configuration is checked when the application is launched for a given mp3 file to be decoded. The demonstration scenario is described hereafter:

1. Startup of XPSoC-V2 with the first architecture (Xilinx-ML410-MM01-MP3-full.bit).
2. XManager boots on Petalinux.
3. XManager downloads the first section of MP3 data (a.mp3) from the network to the DDR memory.
4. XManager initializes XWorker without accelerator (coprocessor is empty) and downloads a standalone application binary for MP3 decoding.
5. XManager launches a mp3 decoder task by sending the size and the address of a.mp3 to XWorker.
6. XWorker runs mp3, decodes and saves output as format WAV (a.wav) to DDR memory.
7. XManager sends back a.wav and waits for a new task from XServer.
8. XManager downloads the second section of MP3 (b.mp3) and decides to reconfigure XWorker to achieve better performances.
9. XManager sends a request to XServer, by giving some information of the new configuration for the XWorker coprocessors (Xilinx-ML410-MM01-MP3DEC-IMDCT.bit), as shown in Fig.6.
10. XManager performs dynamic partial reconfiguration.
11. Xmanager updates the XWorker configuration data in the shared memory, coprocessor-0 “EN” bit is set to 1, and the ”FUNC” field is updated (e.g. IMDCT).
12. XManager launches a mp3 decoder task by sending the size and the address of b.mp3 to XWorker.

13. XWorker runs mp3 with the new configuration based on the initialization step, functions implemented in hardware (e.g. IMDCT) are executed through FSL communication instead of software calls, after decoding output data are saved in a WAV format (b.wav) in DDR memory.

14. XManager sends back b.wav and waits for a new task to execute.

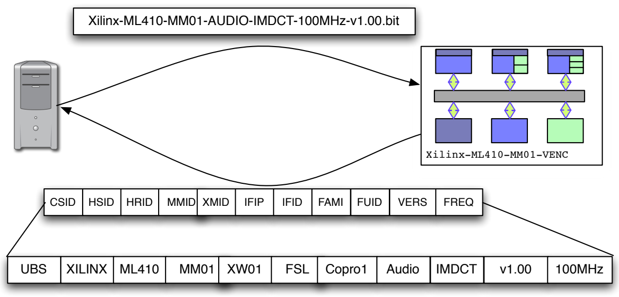

Figure 6. Networked Reconfiguration

This scenario is a demonstration of our approach for the control of hardware reconfiguration of networked embedded systems connected to a server delivering hardware partial configuration files of standard functions according to available architecture models. The different steps have been implemented to demonstrate the whole approach despite current FPGAs limitations. These limitations are related to partial reconfiguration issues and difficulties for implementing peripheral drivers for instance. But the evolution of FPGA tools show that these problems and instability issues will be solved in the future opening very interesting opportunities for our approach.

6 Conclusion

In this paper we have presented a network-based solution to manage configurations of reconfigurable Multiprocessors system-on-chip and we have introduced the concept of on-the-fly reconfiguration based on R-MPSoC model architectures. Then we have described the formalization of configuration protocol, the synchronization of configuration. Finally, we have presented a MP3 case study with a dual-processor reconfigurable architecture model running PetaLinux on Xilinx Virtex-4. This is the first step in demonstrating the efficiency of our global method for future distributed reconfigurable embedded systems. Our approach represents a solution for distributing reconfigurable architectures models to be used in industry domains or to be shared within an open-source framework. Current research tasks are focusing on the cache strategy and the configuration decision policy to be combined with this approach for networked-reconfiguration based on architecture models.

References

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

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

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

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

[5] Y. Eustache and J.-P. Diguet. Specification and os-based implementation of self-adaptive, hardware/software embedded systems. In *CODES/ISSS '08: Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*, pages 67–72, New York, NY, USA, 2008. ACM.

[6] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier. An architecture and compiler for scalable on-chip communication. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(7):711–726, 2004.

[7] P. Manet, D. Maufroid, L. Tosi, G. Gailliard, O. Mulertt, M. Di Ciano, J. Legat, D. Aulagnier, C. Gamrat, R. Liberati, et al. An evaluation of dynamic partial reconfiguration for signal and image processing in professional electronics applications. *EURASIP Journal on Embedded Systems*, 2008.

[8] 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.

[9] P.Bomel, J.Crenne, L.Ye, G.Gogniat, and J-Ph.Diguet. Ultra-fast downloading of partial bitstreams through ethernet. In *Proc of ARCS 2009*, Delft, The Netherlands, March 2009.