XPSoC: A Reconfigurable Solution for Multimedia Contents Protection

Linfeng Ye, Jean-Philippe Diguet, Guy Gogniat
Lab-STICC, CNRS - European University of Brittany/UBS,
F-56321 Lorient Cedex - FRANCE
{*linfeng.ye, jean-philippe.diguet, guy.gogniat*}@*univ-ubs.fr*

ABSTRACT

Network Multimedia data often need to be encrypted to protect private content and access control. Considering performance constraints and embedded system issues, many hardware solutions are proposed to provide high security level and fast encryption process. However these hardware solutions are often unchangeable at run-time and cannot adapt to data-dependent encryption schemes. Reconfigurable architectures are an opportunity to efficiently implement hardwired security functions, which are called according to data-dependent parameters. In this paper we propose a solution that relies on a reconfigurable MPSoC architecture model (XPSoC) and on HW/SW libraries of standard functions that can be easily used by means of HW independent API. We demonstrate our approach on a Virtex IV MPSoC architecture, with bit-processing functions representative of image and security domains.

KEYWORDS: MPSoC, Reconfiguration computing, Selective encryption, Digital Rights Management (DRM)

1. INTRODUCTION

With the fast development and fusion of the multimedia and Internet technologies, an extensive application field is growing while being supported by promising economic trends. However, the virtuality of the network also induces new challenges in terms of security and privacy rights. Encryption and watermarking are two fundamental groups of technologies that have been identified for protecting copyrighted digital multimedia content. Encryption appears to be the right approach that can be used to provide confidentiality in network multimedia applications, such as video conferencing, Pay TV and on-line video games. But the cost of multimedia data encryption/decryption can be strongly increased by the additional effort needed for protecting copyrighted digital content. Software implementations of ciphering are usually too slow to process image and video data in commercial systems. Considering performance constraints and embedded system issues, some hardware assisted solutions are proposed to provide a high security level and a fast encryption process. However these hardware solutions are often unchangeable at run-time and are usually not adaptable to data-dependent encryption schemes.

Dynamic reconfiguration allows to update or substitute blocks of logic at run-time in order to improve the area and performance efficiency of a FPGA design. Reconfigurable architectures provide flexible solutions for designing ad hoc embedded systems that can reach high performances and HW efficiency with low clock frequencies. They are also appropriate for long-life products where HW updates and evolutions, based on reconfiguration, make sense. Dynamic reconfiguration capabilities are not really widely used today out of research laboratories, but yet they are of real interest to cope with fluctuating application behaviors. This is more often the case in application domains such as multimedia and network processing where execution times can be strongly data and context-dependent.

However, such solutions also mean important HW design efforts and significant design cost overhead. Actually performances of FPGA-based embedded systems are usually related to exploitation of spatial parallelism and heterogeneous architectures based on coarse-grain soft-core instructions and HW accelerators. Contrary to other solutions such as GPP (General Purpose Processor) and DSP (Digital Signal Processor) widely used in embedded systems, FPGAs suffer from a lack of standard solutions that limits its design reuse. For instance the popularity of TI[1] in the

[1]Texas Instrument

domain of embedded systems, relies on the definition of API (application programming interface) for coprocessors access and on binaries for direct implementation of standard applications. We also notice that the development of complex multi-core architectures is supported by specific libraries for efficient implementation of standard applications and functions. We believe that such approaches are also relevant for reconfigurable devices, but architecture models must be defined first.

Our methodology targets applications in the domain of embedded systems based on standard functions for which flexible hardware accelerators may be designed and dynamically updated according to application needs. In this paper, we present a methodology for the management of Reconfigurable MPSoC, focusing especially on modeling aspects and mechanism of configuration processing. Our models have been instantiated on a Xilinx FPGA providing dynamically reconfiguration capabilities.

The rest of the paper is structured as follows. After an overview of related work in Section 2, our architecture model is described in Section 3. We then present the so called ”on-the-fly reconfiguration flow” in Section 4. Finally, we demonstrate the efficiency of our approach with results for representative Multimedia Contents Protection application in Section 5 and conclude the paper in Section 6.

2. RELATED WORK

There are various studies reported so far in the domains of multimedia contents protection and reconfigurable computing, however, few research projects are interdisciplinary and combine these two domains. In this section, we briefly introduce some relevant work.

Zhang [7] gives a state-of-the-art anatomy of the security and trust related to DRM (Digital Rights Management), the author indicates in this paper that recently DRM-related cipher researches have mainly been focusing on an improvement on performances of broadcast encryption schemas suitable for much wider applications. Kougianos [8] presents a hardware assisted solution to achieve low power usage, real-time performances, reliability, and ease of integration with existing consumer electronic devices. The author indicates that the specification and the design of digital watermarking systems is a huge task that still needs efforts to obtain a more general approach meeting expected requirements.

Numerous experiences have been carried out in the domain of reconfigurable architectures, a new taxonomy of reconfigurable System-on-Chip is available in [4]. In this paper, Ghringer present the complexity of the new degrees of freedom in terms of run-time adaptivity and classifies the different approaches of reconfigurable architecture provided by academics and industry. In the domain of Reconfigurable MPSoC, we can identify two kinds of approaches. The first one consists in new computing architectures that aim to provide designers with flexible architectures mostly on FPGA, the Erlangen slot machine [6] is a good example of such solutions. We didn’t follow this kind of approach since our aim is to be able to reuse standard SW solutions in the domain of embedded systems. The other solution, that we have chosen, targets the specialization of embedded processors by means of dynamic reconfiguration. In this category, RAMSoC [5] is an interesting project, where are addressed the two main drawbacks of traditional approaches. The first one is the necessity to find a trade-off between homogeneous and application-specific MPSoC. The second one is a meet-in-the-middle design methodology to offer runtime configuration capabilities. This work is related to a kind of architecture model with various processor types. The proposed solution is based on soft-processor (MicroBlaze) with configurable accelerators that can communicate through a configurable network on chip.

Our approach is quite different. We use Petalinux [3] as an available Linux distribution for MicroBlaze and we consider an architecture model where master processors running Linux communicate with specialized slaves, without OS, executing computation intensive applications. Our objective is to decide configuration online, to provide synchronization solutions between master and slaves and to execute programs where registered functions are called. So our contribution, which is independent from the OS choice, is mainly focalized on the way a master can fire and specialize slaves with specific computing intensive tasks. From an OS point of view it means that we mainly use synchronization mechanisms. From a SW perspective, applications are designed by means of standard libraries called througn API independent from implementation. Finally, the decision of specialization is implemented as new process running on the master.

Actually our solution is closer to TI approach, used for instance with the Da Vinci SoC to implement video codec on coprocessor through standard API (VISA, xDM [1]). However we adapt this approach to the domain of dynamically and partially reconfigurable architectures, where a master processor can dynamically assign tasks to slave processors which are configured at run-time.

3. XPSOC ARCHITECTURE MODEL

A general definition of the XPSoC is defined by a reference model $MxSy_iSy_{i+1}Sy_n$, where Mx is a XManager with x places for reconfigurable coprocessors, S means a XWorker with y_i coprocessor places. In case S is not a XWorker but an hardware accelerator (XModule), no reconfigurable places are mentioned. The master processor runs Petalinux [3], it is in charge of I/O communications and process management. This model has been designed for data-flow applications. Communications between processors, for data and control, are implemented with shared memories.

A slave executes one thread at a time, this thread is decided by XManager. When a XWorker is in a sleep mode, it pends on its shared memory waiting for some new tasks to execute. Figure 1 shows an example of a given architecture model (M0S2S3) for MicroBlaze-based Xilinx architectures. It is composed of one Xmanager and two Xworkers; the two Xworkers have two and three reconfigurable coprocessors (C11,C12) and (C21,C22,C23) respectively. These coprocessors, which are acceded through Fast Simplex Link (FSL) busses, are configured by XManager at run-time.

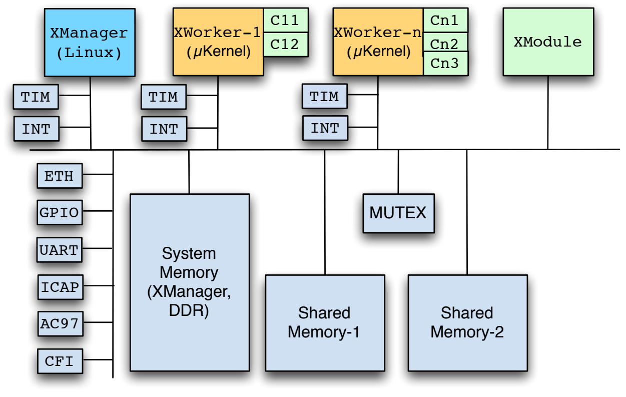

Figure 1. XPSoC: Reconfigurable Architecture Model and Synchronization Mechanism

Note that Xilinx partial reconfiguration flow [2] enables execution/reconfiguration overlapping, thus the XManager can reconfigure an unused co-processor without freezing the XWorker. The objective of the reconfiguration decision is to optimize execution time according to applications needs. The main concept of this architecture model relies on the following observation. A majority of embedded systems are designed with standard functions in various application domains (e.g. Image processing, Signal processing, Encryption etc.). Thus the idea is that, given architectural models, data-flow applications can be quickly specified by reusing standard functions previously registered in libraries with a unique identifier (FUID). Considering interface constraints, these functions can be massively and rapidly produced with high level synthesis tools. In our view they are classified according to application domains and called transparently through standard APIs. HW and SW versions are stored on configuration servers and classified according to various parameters such as the architecture model (e.g. M0S2S3) and the target platform (e.g. Board ML410, Virtex IV). In the case of dynamically reconfigurable devices such as Xilinx FPGA, requested bitstreams and binaries can be downloaded from servers to local memories (Flash, DDR) at boot-time or at run-time. More details about our XPSoC architecture model can be found in a previous paper [9].

4. ON-THE-FLY RECONFIGURATION

In the case of a XPSoC composed of a XManager and a XWorker with two configurable co-processors C_{1n} (n=1,2), we obtain the following on-the-fly configuration flow:

a) When a data-flow application is launched, XManager checks the associated profile. If a XWorker is available, then XManager can register tasks to be executed on XWorker. It indicates, in the shared memory, addresses for binaries and for data inputs / outputs. The first HW configuration is based on a profile information. If a reconfiguration is required, then the on-the-fly reconfiguration flow starts, it is described in steps b to g.

b) XManager indicates to the XWorker, by setting ”EN” bit to 0, that current function implemented on C_{1n} won’t be available as a HW version when the new coprocessor will be released by the current task. If the bit ”Done” of C_{1n} is set to 1, it is unused and the flow jumps to step e; Otherwise XWorker switches to the SW binary version as soon as the input buffer is empty. If necessary, XManager loads configuration files in a local memory (e.g. DDR) from flash memory or remote configuration servers.

c) XWorker executes the program, and tests its configuration when its input buffer is empty by reading the bit ”EN” of coprocessor C_{1n}. It means that the configuration rate is driven by task buffer size. If coprocessor C_{1n} is disabled (”EN”=0), the XWorker cannot used the HW function anymore. It informs XManager by setting ”Done” bit to 1 that the coprocessor is free for reconfiguration.

d) XManager tests the bit "Done" of C_{1n}, if it is set to 1, then the configuration starts in step e.

e) XManager copies the configuration from DDR Memory to HW-ICAP with Xilinx API [2] and finally performs dynamic partial reconfiguration. Depending on the size of the configuration file, different iterations might be necessary. It also copies the associated binary file in shared memory.

f) XManager updates the "FUID" bits of C_{1n} to indicate that the new function is available as a HW version and then enables C_{1n} by setting "EN" bit to 1.

g) XWorker tests its configuration by reading the bit "EN" of C_{1n}, if it is set to 1, function calls corresponding to the FUID will run with C_{1n} and Done is set to 0.

5. CASE STUDY AND RESULTS

We have designed a XPSoC instance on a Xilinx ML410 (Virtex-4 FX60) FPGA, the architecture model (M0S2) is based on a Master Processor with a two-coprocessor XWorkers. The implementation leads to an area occupation of 32% for logic block slices, 23% for RAM blocks and 13% for DSP. The XManager runs a petalinux OS. The area of coprocessors are not identical so reconfiguration times are different, it means that the choice of the coprocessor location has an impact on the reconfiguration initial penalty.

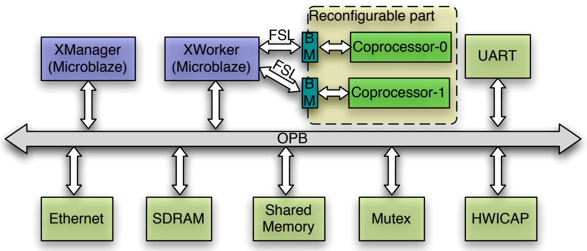

Figure 2. XPSoC-V2 Hardware Architecture

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. 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, 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.

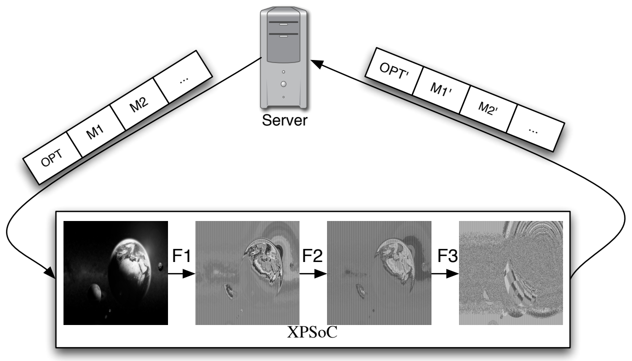

Figure 3. A Case Study Of Image Encryption/Decryption

The data-flow application deals with typical bit manipulations figuring a possible encryption algorithm. It is composed of three chained reconfigurable tasks using three standard functions which can be accelerated with HW coprocessors: F1 $\rightarrow$ F2 $\rightarrow$ F3. Figure 3 shows image encryptions associated with each function. Based on the option parameters in the OPT file, the application receives and encrypts / decrypts images (M1, M2, ...) and sends the outputs (OPT', M1', M2', ...) to server. Table 1 presents the main features including speedup per data for the three functions. T_i^s is average execution time per data, when function i is executed by XWorker without co-processor. T_i^h is average execution time per data, when function i is executed by XWorker with associated co-processor. Rt_i^k is reconfiguration time for function i implemented on a co-processor k. $g_i = \frac{T_i^s}{T_i^h}$ is speed-up per data, when function i is implemented with a co-processor.

Function (Name/fuid/I:0)	T_i^h (μs)	T_i^s (μs)	g_i	Rt_i^1 (μs)	Rt_i^2 (μs)	Area (slices)
Decryption F1 (2:1)	5.8	382.3	65.91	91367	383599	21
Encryption F2 (2:1)	6	647.3	107.88	92430	383197	21
BitInverse F3 (1:1)	9.7	20.5	2.11	95780	383142	55

Table 1. Description Of Encryption/Decryption Functions Chain

The encryption algorithm is data-dependent, thus the amount of data to be processed by each function depend on parameters transmitted as data. As a consequence the best architecture, in terms of performances, varies in time. The objective of these experiments is to demonstrate different aspects of our approach to dynamically select the right configuration according to application demands.

Figure 4 illustrates how the system decides and executes reconfigurations by itself according to the variation of input buffer sizes. N = 20248, 10124 and 512 Bytes for curves X,Y and Z respectively and input data file is the same for all three cases. The decision algorithm sorts critical functions according to their impact on the whole algorithm execution time by considering speed-up, providing by HW implementation, and function granularity, which is the amount of data to be processed. In order to avoid instability, the cost function implements a weighted summation with previous values according to a parameter a_i. When a_i is set to $1/2$, it means an average between current and previous iteration values, this is the value chosen for our experiments.

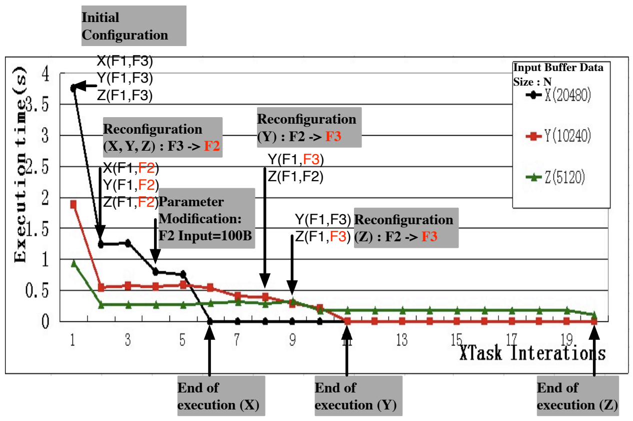

Figure 4. On-the-fly Reconfiguration

The initial configuration is (F1,F3) and the optimal solution is found after the first iteration for all three cases. We can observe the effect of specialization since the execution time is then drastically reduced for all curves. In iteration 4, we change the granularity of F2 from N to a fixed value equal to 100 Bytes. Consequently the optimal solution becomes (F1, F3). We observe that reaction delays depend on granularity since the impact on performances increases in relation to N. That's why the decision occurs during iteration 8 for Y and during iteration 9 for Z. The reduction of the execution time is observed in the subsequent iteration when reconfiguration has been performed. This effect is not visible on X since it ends after iteration 5, before the decision can occur.

6. CONCLUSION

Reconfigurable architecture provides an interesting hardware assisted solution for network multimedia applications since hardware specialization can offset low clock frequencies and finally provide high computing performances. This kind of architecture fits with encryption and watermarking algorithms dealing with bit-level operations. However the main drawback is the complexity of the programing. In our work we propose to overcome this difficulty by means of a predefined reconfigurable architecture model (XPSoC) and a programing model, based on API library and registered standard functions, for which hardware solutions are available. We also introduce an online decision solution, which sorts and implements critical functions according to their performance gains. In this paper, our approach is successfully applied to an example corresponding to run-time image encryption / decryption schemes.

REFERENCES

[1] "Accelerating innovation with the davinci SW code and programming model", http://focus.ti.com/lit/ml/sprp478/sprp478.pdf.

[2] "Early access partial reconfiguration user guide", http://www.xilinx.com.

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

[4] Ghringer, D. and Perschke, T. and Hübner, M. and Becker, J. "A Taxonomy of Reconfigurable Single-/Multiprocessor Systems-on-Chip". In *International Journal of Reconfigurable Computing*, 2009.

[5] Göhringer, D. and Hübner, M. and Schatz, V. and Becker, J. "Runtime adaptive multi-processor system-on-chip: RAMPSoC". In *IPDPS*, pages 1–7, April 2008.

[6] Majer, M. and Teich, J. and Ahmadinia, A. and Bobda, C. "The erlangen slot machine: A dynamically reconfigurable fpga-based computer". *J. VLSI Signal Process. Syst.*, 47(1):15–31, 2007.

[7] Zhang, Z. and Pei, Q. and Ma, J. and Yang, L. "Security and Trust in Digital Rights Management: A Survey". *International Journal of Network Security*, pages 247–263, 2009.

[8] Kougianos, E. and Mohanty, S.P. and Mahapatra, R.N. "Hardware assisted watermarking for multimedia". In *Computers & Electrical Engineering*, 339–358, 2009

[9] Ye, L. and Diguet, J.-P. and Gogniat, G. "Modeling of Reconfigurable MPSoCs for On-Demand Computing". In *GRETSI (traitement du signal et des images), FRANCE* , 2009