13th International conference on Sciences and Techniques
of Automatic control & computer engineering
December 17-19, 2012, Monastir, Tunisia

Exploration tool for analyzing the performance of MPSoC design

Sonda Chtourou [1], Zied Ben Salem [1,2], Mohamed-Wassim Youssef [3], Mohamed Abid[1]

[1] CES Research Laboratory, National School of Engineers of Sfax, University of Sfax, Tunisia.
{sonda.chtourou, zied.bensalem, mohamed.abid}@ceslab.org
[2] ALPHA ENGINEERING Industrial Society, Tunis, Tunisia.
engineering.director@alpha-engineering.net
[3] Higher Institute of Computer Science, Tunis, Tunisia.
wassim.yousef@yahoo.fr

Abstract. *The major problem faced when designing Multi-Processor Systems-on-Chip (MPSoCs) is the difficulty and the complexity of interfacing the hardware (HW) and software (SW) parts with high-performance requirements. Moreover, HW is always designed far too late in the product development cycle. Thus, SW can't be checked for early internships in the design cycle. Virtual prototyping represents a good alternative to respond this problem by allowing early design integration. However, virtual prototyping platforms use textual-language description to design and integrate HW and SW parts. Thus, designer needs flexible and interactive tool that runs on the top of virtual platform to rapidly and easily explore alternative solutions. In this paper, we propose an interactive tool with a graphical web-powered layer for HW and SW exploration at virtual prototyping level. We propose also an approach to estimate the global performance of an MPSoC during exploration process. Using this tool, designer would be able to make early HW and SW integration and validation in the design process of MPSoC. This tool provides precise and accurate performance information regarding the MPSoC to be designed. The presented approach was validated using H264 encoder design as a case study.*

Keywords. *Interactive exploration, MPSoC, virtual platform, virtual prototyping level and H264 encoder.*

1. Introduction

Current embedded software, such as multimedia and telecommunication applications, are becoming more and more complex. Consequently, the design complexity is exponentially increasing. Embedded applications, like H264 compression video and 3D image synthesis, are designed in multi-threading context and require computing and power guaranties to handle their complexity. Thus, embedded systems use an increasing hardware parallelism (MPSoC) to deliver high computation and communication performances while maintaining a reduced power budget. In addition, to manage the complexity of both the supported applications and the target architectures, the use of operating system (OS) becomes mandatory. Furthermore, the continuous increase of embedded application and HW design complexity has intensified the challenges. Indeed, today's marketing managers require high-performance MPSoC with minimal cost and short time-to-market.

Classical design flow recommends designing HW and SW in parallel way. However, HW is always designed far too late in the product development cycle. Thus, SW can't be checked for early internships in the design cycle. This implies a huge gap between HW and SW integration. Meanwhile, virtual prototyping seems to be a good candidate to respond to this need. Indeed, the use of virtual prototyping simulators allows early design validation. However, virtual prototyping platforms use textual-language description to integrate HW and SW parts. Therefore, it's not easy to extract the best applications parallelism with high-performance requirements to be mapped on the best MPSoC architecture which meets constraints. Thus, designer needs interactive tool that runs on the top of virtual platform with graphical interfaces to rapidly and easily explore others alternative solutions and then to learn how a particular performance metric depends on a particular parameter.

In this paper, we propose an interactive tool for HW and SW exploration at virtual prototyping level. This interactive exploration tool will help designer to take decision of the optimal design at early stage in the design cycle of MPSoC. To achieve our goal, we proposed an approach to estimate MPSoC performance and we described an interactive technique of exploration based on different HW and SW variations. The major contribution of this work is that we enabled exploration through a graphical web layer that runs on the top of virtual platform in order to make the technique of exploration interactive and to allow an easy exploration. As case study, we realized experimentation with H264 encoder benchmark to prove the concept of the proposed approach. The experiment shows that the proposed tool permits to handle the complexity and flexibility needed in the development of MPSoCs and the early selection of the best system which meets constraints.

This paper is organized into four sections. Section I presents the problem statement and related works. Section II details the development of an interactive tool for exploration of MPSoC at virtual prototyping level. Section III exposes how the proposed interactive exploration was applied to the H264 encoder case study. Different results

and interpretations of exploration of the parallelized H264 encoder mapped on MPSoC are discussed in Section IV.

2. Problem statement and related works

MPSoC allows integration of all needed electronic circuits in a single chip. Meanwhile, it opens up new challenges for HW and SW integration in order to satisfy maximum performance and tight time-to-market requirements. The SW running on the MPSoC architecture is called software stack (figure 1). The software stack is composed of two components: the application tasks code and the hardware-dependent software (HdS). The HdS layer is composed of three layers which are OS and communication middleware, the hardware abstraction layer (HAL) and the HAL/API. The HdS is responsible for providing application and architecture specific services such as scheduling the application tasks, communication between the different tasks, external communication with other sub-systems and HW resource management and control [1].

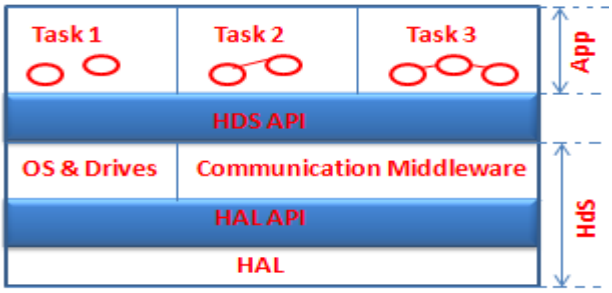

Fig. 1. A typical software stack.

The major problem faced when designing HdS is the difficulty and the complexity of interfacing the HW and the SW. Indeed, this process depends on several important details like drivers, type of memory, CPU and communication. Thus, SW can be validated only on ready-to-use model of the HW platform. In addition, HW is generally designed with register-transfer level (RTL) which is always far too late in the product development cycle. This leads to an outstanding cycle design. Besides, if the HW implementation is done at a low level, making modification of mapping is costly and difficult. Thus, the development of HW and SW components in parallel way risks to lead to incompatibilities and to make unfeasible the early check of SW.

Meanwhile, abstraction levels allow designing SW in incremental process by verifying the SW on an abstracted representation of the HW. In this manner, HW and SW development can progress together without conflict. Consequently, HW and SW integration in complex MPSoC should converge on using different HW and SW abstraction levels. We referenced in this research to the thesis work of Katalin Popovici from Tima laboratory [1] who performed the validation of SW in four abstraction levels

which are system architecture design, virtual architecture design, transaction accurate architecture design and virtual prototype design. Each level depicts a step in the SW refinement process. This helps to gradually transform the system with high level representation and abstract components into a concrete low level executable SW code. Particularly, in the virtual prototyping level, designer adapts the specific SW to map the target processors and peripherals. This integration takes care of the processor dependent SW code into the SW stack. The HW platform includes all the HW components, such as cache memories or scratch pads, to allow low level access to the HW resources and the final memory mapping. Therefore, compared to other abstraction levels, the virtual prototyping level provides precise and accurate performance information.

For this purpose, several frameworks and tools using in backend virtual prototyping platform were developed in order to speed up exploration of MPSoC design. [2], [3], [4] and [5] proposed frameworks based on SystemC allowing higher simulation speed with early performance estimation. Proposed frameworks are useful for exploration and permit to rapidly find the most adequate HW/SW configuration. However, SystemC is HW-oriented language and it is not the standard language to design complex applications at algorithm level [6]. Simulink and UML are also used inside frameworks for high-level modeling. Simulink is a mixed HW/SW architecture model allowing abstraction of HW/SW interfaces of multithreaded heterogeneous multiprocessor architecture with specific HW I/O [7]. [6], [8] and [9] propose frameworks enabling mixed HW/SW refinement and opening new facilities like communication mapping exploration and interconnection component refinement. However, embedded programmers are still reluctant to adopt Simulink models on MPSoC because it quickly becomes unsuitable with complex HW architectures. Some environments also use Qemu emulator which is equipped with dynamic program trace facilities. The Qemu emulator can run an operating system Linux for several architectures like x86, Sparc, MIPS, PowerPC and ARM ISAs. QEMU was used as the foundation for Yeh Tse-Chen projects [10]. These projects showed that the combination of QEMU and SystemC can make the co-simulation at the cycle-accurate level extremely fast, even with a full-fledged operating system up and running [10]. However, Qemu shows lack of flexibility to configure specific HW architectures and supports only homogeneous designs. Meanwhile, OVP (Open Virtual Platform) allows simulating complex heterogeneous and homogeneous MPSoC architectures at high simulation performance which can be measured by hundreds of million instructions per second (MIPS) [11]. Besides, OVP provides a large set of free open sources and standard models components providing more flexibility to configure complex HW system. OVP uses the same level of simulation but also tackles the simulation speed problem by using advanced techniques for the software simulation such as OS simulation for the former and code morphing for the latter. Several studies have been performed using the OVP simulator targeting MPSoC architectures. [12] investigated two different virtual platform emulators QEMU and OVP for integration into SCE (System-On-Chip Environment) and opted on integrating the OVP. In addition, [13] checked simulation performance of OVP and possibility for hybrid simulation with SCML (Open SystemC Modeling Library). Moreover, [14] used OVP to simulate both hardware architectures and run-

ning software applications. From the above mentioned raisons, OVP has been embraced as a primary solution to make easier system description (SW and HW integration) and faster simulation [15]. Thus, OVP represents a good virtual platform solution to speed up exploration of MPSoC design. However, even if OVP performs early and fast system description, it uses scripts and textual-language description to integrate HW and SW parts. Therefore, its specification complexity rises with the virtual platform complexity rise (MPSoC, several OSs and applications). This becomes a new bottleneck in the HW/SW integration and in the exploration of the best applications parallelism to be mapped on the best MPSoC architecture that meets constraints. In fact, designer needs interactive graphical interfaces running on the top of OVP to easily explore others alternative solutions and then to learn how a particular performance metric depends on a particular parameter. In this context, we didn't find any work or environment with graphical layer allowing interactive exploration that runs on the top of OVP. Our contribution is to design a higher graphical layer on the top of OVP environment to allow an easy graphical interactive exploration and profit of OVP features.

3. Development of an interactive tool for MPSoC exploration at virtual prototyping level

3.1. General proposed workflow

To design a higher graphical layer on the top of OVP allowing an easy graphical interactive exploration, we started by defining a workflow handling different needed phases to make exploration and then design system satisfying performance constraints at early stage (figure 2):

- Phase 1: Partitioning and mapping: select a solution of partitioning and mapping tasks of chosen complex application into processors. Designer chooses the number of processors and how to map tasks into each processor in order to define the relationship between SW and HW;

- Phase 2: Configuration of the design: this phase contains two sub-actions achieved in parallel way:
  - Configuration of SW prototype: select different options and features (OS, application parameters) of the SW prototype. Then, generate SW prototype representing this configuration;
  - Configuration of the virtual MPSoC HW prototype: instantiate different needed compounds and interconnections with OVP. Then generate virtual HW prototype.

- Phase 3: Integration: integrate and simulate SW and virtual HW prototypes with OVP to make early integration;

• Phase 4: Performance estimation: use the defined approach of MPSoC performance estimation to estimate performance of the configured system;
• Phase 5: Interactive technique of exploration: allow designer to explore various integration alternatives to best satisfy defined performance requirements and objectives.

When designer fixes the best configuration which meets defined performance objectives, he can proceed to the real fabrication of the HW and then make final integration.

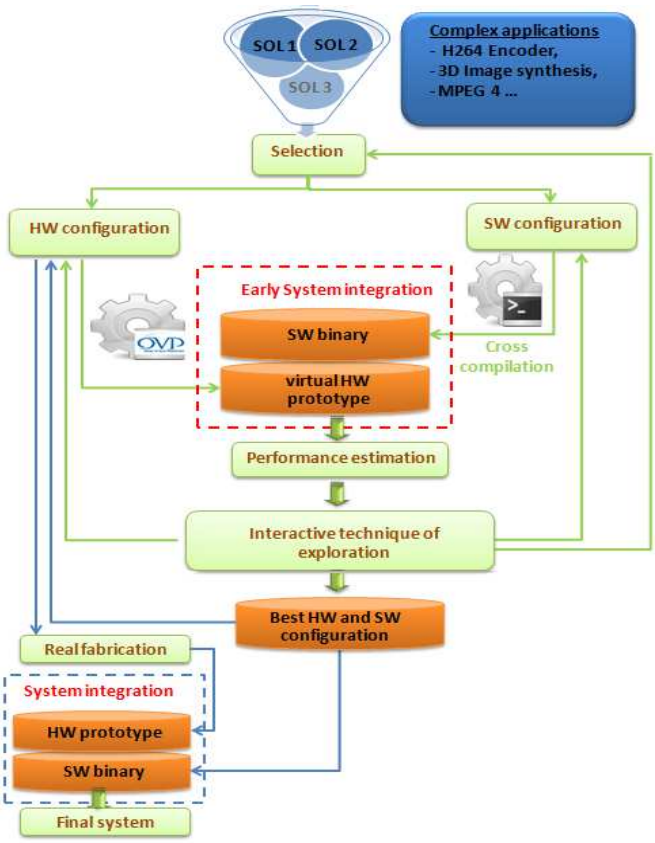

Fig. 2. Proposed workflow of interactive exploration.

3.2. Define an approach to estimate performance of MPSoC

To define an approach of performance estimation (phase 4 of the workflow), we have to fix performance metrics. Several works focused on performance estimation. In our project, we proposed HW and SW performance metrics to measure efficiencies in terms of used HW resources, average turnaround time and quality of the service (QoS). The designer can select or combine proposed metrics, evaluate each configured system and make decision.

- *Used hardware resources*

Through estimated percentage of used HW resources of the configured system, designer can reduce the cost on the final SoC by designing minimal configuration that performs the needed constraints. CPU and Memory are the most important HW compounds in term of necessity and cost. Thus, we chose to estimate used percentage of memory (%MEM) and of processor (%CPU):

- %MEM: Percentage of memory used by the process from the available physical memory;
- %CPU: Percentage of total CPU time that the process consumes from the elapsed CPU time.

- *Average turnaround time (execution time)*

Time elapsed by the whole application with the configured HW system depicts a widely used constraint to evaluate performance of the system. Time statistics consist of the actual elapsed time between invocation and termination (real), the CPU time running the program code (user) and the CPU time running the system calls (sys): real = user + sys.

- *Quality of Service*

Aside from the three performance metrics (%CPU, %MEM, time), the QoS provided by the system depicts an important aspect in SoC design. This metric mainly depends of the target application.

3.3. Define an interactive technique of exploration

We proposed a technique of exploration (phase 5 of the workflow) to explore other solution alternatives. This technique is based on several HW and SW variations which mainly affect the performance of the system. Designer can also modify the number of target CPUs and the SW parallelization levels. He can also adjust the number of MIPS and the architecture of target CPUs depending on the need of the mapped task. Distribution, type and size of used memory or SW parameter setting can be also revised. Thus, designer can use this technique to identify the effect of each modified HW or SW parameter on performance. Based on results, he can improve some parts of the design or revise design options to satisfy performance requirement.

3.4. Design of higher graphical layer on the top of OVP

We developed graphical user interfaces on the top of OVP handling different phases of proposed workflow (figure 2). These interfaces allow designer to perform an easy and fast interactive and graphical exploration and then design system satisfying performance constraints at early stage. Then, we implemented different scripts automating several back-end functions of these interfaces (configuration, compilation and generation of the SW, integration of configured HW and SW with OVP, storage of generated system in database, start up of configured system). Storing configured system in database allows designer to save time and effort to modify some parts of a configured design without repeating all steps. We used the following HW and SW tools (Linux: Host OS, APACHE: web server, PHP: Script language and MySQL: Manage database).

4. Validation of the proposed tool

After implementing this approach, it is imperative to check whether the solution achieves the expected behavior. After several researches, we chose H264 encoder benchmark because it is designed in multi-threading context and requires considerable HW resources to encode a video. In experimentation, we used the x264 version of H264 encoder [16].

4.1. Mapping H264 on MPSoC

The used H264 version encodes the video frame by frame. Thus, we found that it is interesting to split encoding the video between two processors (processor 1 and processor 2). Each processor encodes each following buffered frames as if we have a video coming from a camera. After encoding each buffered frames, each processor puts the encoded frames in a shared memory. Then, a third processor (processor 3) reads data (encoded frames) from the shared memory and generates the encoded video.

H264 runs several threads simultaneously. Thus, we need to use an OS to manage the different threads. We chose Embedded Linux as target OS because it addresses the basic needs of our project: flexibility to add, facility to modify or integrate new drivers in a configured Linux kernel, multi-processor support, stability and portability. All these advantages guarantee the extensibility of the system in future.

4.2. Implementation of parallelized H264

1) Custom virtual hardware design

In this part, we focus on the implementation of a custom virtual HW design to achieve the previous described objective. We need a design which contains 3 proces-

sors and a shared memory to assure communication between the different instances. We need also to use Embedded Linux for each processor in order to run the threads of the application.

The first step was to find a sub-system (minimal design which boots Linux) based on ARM processor. This sub-system contains several components: ARM926EJ microprocessor core (ARM), RAM memory (RAM), Interrupt Controller (PIC), Counter Timers (PIT), UART and Smart Loader Arm Linux (SmartLoader). The second step was to implement an MPSoC design based on arm sub-system for the parallelized benchmark. As we need 3 processors, we configured with OVP an MPSoC which contains 3 arm processors with minimal needed peripherals (3 arm sub-systems) (figure 3). We also configured a shared memory to communicate between different subsystems of the design. In each sub-system, we intended to run Embedded Linux to assure exchange of data (pictures) between processors.

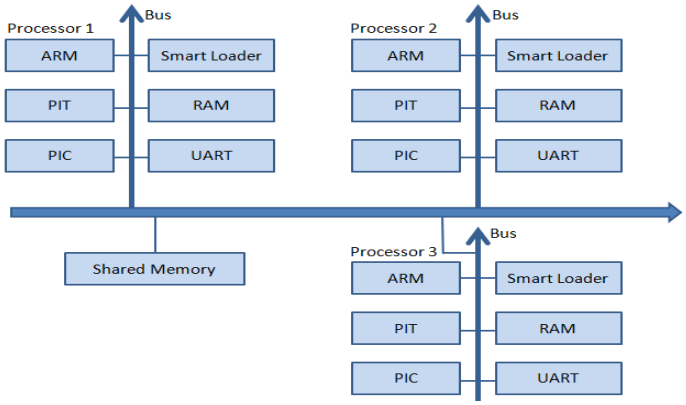

Fig. 3. Custom MPSoC design.

2) Configuring software stack

Embedded Linux is based on 2 principal compounds which are the kernel and the file system (rootfs). We started by configuring a specific kernel for each sub-system with all needed peripherals. To generate the compressed zImage of the kernel, we used an EABI cross-compiler for arm processor. The file system (rootfs) represents a principal part of the operating process of the whole system. It must be mounted at the start up because it contains all needed scripts to boot the OS. We configured all needed repository to boot Linux (/bin, /etc, /proc, /sys…). We used Busybox tools to add needed Linux commands. Finally, we modified booting scripts in order to start running target application automatically after the boot.

3) H264 parallelization

H264 encodes and writes the encoded frames, into the generated video, in a same function. Thus, we started by modifying H264 code in order to separate encoding the video and generating the encoded video. After that, we modified H264 code for processors 1 and 2 in order to encode then write each following buffered frames in the

shared memory. Then, we modified H264 code for processor 3 in order to read each following encoded frames written by each encoder processor and generate the encoded video.

4.3. Methodology of exploration

The performance objective of our case study is to design a system which runs parallelized H264 on 3 processors with minimal time and HW resources while keeping an acceptable QoS. We performed interactive exploration with the assumption that we keep the chosen partitioning and mapping solution (IV. A: H264 parallelized 3 main tasks on 3 processors). However, during the interactive technique of exploration, we intent to explore other variations that affect performance like: (i) variation of the number of buffered frames (HW and SW modification), (ii) variation of the encoder parameters (SW modification).

As we mentioned in section 3.2, QoS performance metrics depend mainly on target application. As we chose H264 codec, it is interesting to measure the quality of the encoded video because it is the most important characteristics of a codec. PSNR is the most widely used video quality metric [17]. Consequently, we used PSNR metric to measure the quality of the encoded video.

5. Tests and results

5.1. Configuration of the first prototype system

We used configured graphical web interfaces to configure the first prototype system (see figure 4). We have just to click checkbox and bottom of proposed interfaces to configure and generate the first prototype system (see 3.4). Then, we selected needed performance metrics to estimate and we pressed “Emulate” button to start booting the whole system with OVP (see figure 4). Each console started up automatically the associated target task of the parallelized H264. When the whole application terminated, each processor wrote the performance estimated metrics. The generated system is automatically stored in database and then designer can apply interactive technique of exploration by modifying some part of the design without repeating all steps.

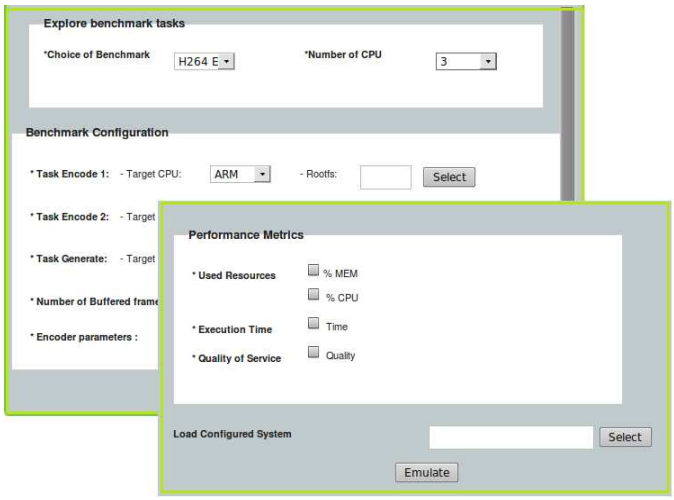

Fig. 4. Proposed graphical interfaces.

5.2. Exploration of parallelized H264

1) Evaluation of references metrics

First of all, we executed the H264 on processor 1 (200 MIPS) without any modification (default version of H264). Then, we estimated all fixed performance metrics in order to take them as a reference for all modifications done later on: %MEM=27.8%, %CPU=99.6%, Time=138.74s and PSNR=36.480kb/s.

2) Impact of variation of the number of buffered frames

In this test, we used 4 different numbers of buffered frames: 10, 15, 20 and 25. We fixed all MIPS of processors at 200 and we used the same parameters of encoding. Table 1 gives results of quality of the encoded video. On the horizontal axis, there is relative encoding time (in s). On the vertical axis, there is relative PSNR (in kb/s) which depicts the quality of the encoded sequence.

Table 1. Effect of varying the number of buffered frames.

	PSNR (kb/s)	Time (s)
10 frames	36.392	77.40
15 frames	36.162	96.24
20 frames	36.254	97.53
25 frames	36.109	110.26
10 frames	36.392	77.40

This table shows that for all used number of buffered frames, the time needed to encode the video decreased comparing to the default execution. For example, it de-

creased from 138.74s to 77.40s when we used 25 buffered frames. When we compared the time measured for each number of buffered frames for cases 10, 15 and 25, we noticed that the time decreased for a higher used number of buffered frames. However, when we compared cases 15 and 20, we noticed that the time needed increased by 1.29s. In fact, the time depends mainly on the number of loads of buffered frames to encode the following frames. Thus, when we have same number of loads for two different numbers of buffered frames, we have close execution time.

A higher PSNR means that we have a better quality. For these cases, the quality of the encoded video slightly decreased comparing to the reference case. This decrease can be explained by a bad choice of reference frames (I-frames). In fact, the encoder processor has to choose a reference frame from just this limited number of buffered frames.

3) Impact of variation of the encoder parameters

As said before, in the used H264 code there is a lot of metrics which can be modified through command-line. We used different presets (encoder parameters) which differ from the default one by turning on one (in most cases) codec's parameter. We operated with Bitrate/Quality by using different bitrates: 100, 500, 1800 kb/s. H264 includes several profiles. We used baseline, main and high profiles. Besides, H264 offers preset options allowing to trade off compression efficiency against encoding speed. In addition, H264 has options that control the VBV (Video Buffer Verifier) which is used to constrain the output bitrates. In test, we manipulated two options that control the VBV. We fixed the buffer's size of the VBV at 400 kb and the rate at 300 kb/s. Table 2 illustrates results of used presets for 25 buffered frames.

Table 2. Effect of varying encoder parameters.

	PSNR (kb/s)	Time (s)
Bitrate 100	29.215	51.30
Bitrate 500	35.236	70.02
Bitrate 1800	38.305	113.10
Profile Main	36.113	79.80
Profile High	36.126	78.20
Profile Baseline	36.100	77.76
Preset veryfast	35.863	49.60
VBV	35.563	82.35

Depending on what the user needs for his payload device, he can make a fast comparison. Thus, all conclusions ("better", "worse", "faster", etc) will be made from this point of view.

4) Exploration results

It is now easy to select the best system configuration that satisfies the fixed performance objectives for our case study (see 4.3). Through previous exploration, we have to fix the number of buffered frame to 25 and preset “veryfast” in order to use the minimal HW resource and time while keeping an acceptable QoS.

6. Conclusion

To handle the complexity and flexibility needed in the development of MPSoCs, virtual prototyping platforms would be an interesting alternative. In this context, we proposed an interactive exploration tool of MPSoC at virtual prototyping level. The main contribution of this work is that we enabled interactive exploration through a graphical layer allowing easy interactive exploration. Other implementation steps are possible as perspectives of this work:

- Automate the configuration of the design with graphical web layer which runs on the top of OVP;
- Proof of concept of the proposed approach with heterogeneous MPSoC design to extend comparison of prototypes;
- Define HdS API in order to adapt the map of any application with HW layer with a simple call of HdS API.

References

1. Popovici, K. “Multilevel Programming Environment for Heterogeneous MPSoC Architectures”, PhD Thesis , TIMA Laboratory, Grenoble, France, 2008.
2. Gerin, P. “Flexible and Executable Hardware/Software Interface Modeling For Multiprocessor SoC Design Using SystemC”, ASP-DAC '07: Proceedings of the 2007 Asia and South Pacific Design Automation Conference, 2007, 390-395.
3. Mello, A. “Parallel Simulation of SystemC TLM 2.0 Compliant MPSoC on SMP Workstations”, DATE '10: Proceedings of the Conference on Design, Automation and Test in Europe, 2010, 606-609.
4. Ben Atitallah, R. “Modèles et simulation des systèmes sur puce multiprocesseurs : estimation des performances et de la consommation d'énergie”, PhD Thesis , University of sciences and technologies of Lille, 2008.
5. Madl, G. “Combining Transaction-level Simulations and Model Checking for MPSoC Verification and Performance Evaluation ", ACM Transactions on Design Automation of Electronic Systems, 2009.
6. Popovici, K. “Simulink based hardware-software codesign flow for heterogeneous MPSoC”, SCSC'07: Proceedings of the summer computer simulation conference, 2007, 497-504.
7. Popovici, K. “Mixed Hardware Software Multilevel Modeling and Simulation for Multithreaded Heterogeneous MPSoC”, VLSI-DAT'07: VLSI Design, Automation and Test, 2007,1 - 4.
8. Huang, K. “Simulink-Based MPSoC Design Flow: Case Study of Motion-JPEG and H.264”, DAC '07: Design Automation Conference, 2007, 39-42.

9. Atat, Y. “Simulink-based MPSoC Design: New Approach to Bridge the Gap between Algorithm and Architecture Design”, ISVLSI '07: Proceedings of the IEEE Computer Society Annual Symposium on VLSI, 2007, 9-14.
10. Chen Yeh, T. “On the interface between QEMU and SystemC for hardware modeling”, ISNE'10: International Symposium on Next-Generation Electronics, 2010, 73.
11. http://www.ovpworld.org/technology.php#sectionFOS.
12. Pablo, E. “Integration of Virtual Platform Models into a System-Level Design Framework”, University of Texas, 2010.
13. Agrawal, P. “Hybrid Simulation Framework for Virtual Prototyping Using OVP, SystemC & SCML”, Journal of Systems Architecture: the EUROMICRO Journal, 2010, 99-111.
14. Iulian, N. “A new HW/SW co-design method for multiprocessor system on chip applications”, ISSCS'09: International Symposium on Signals, Circuits and Systems, 2009.
15. Ben Atitallah, R. “A fast MPSoC virtual prototyping for intensive signal processing applications”, MICPRO: Microprocessors and Microsystems Embedded Hardware Design Journal, 2012,176–189.
16. www.videolan.org/developers/x264.html.
17. Wang, Y. “Survey of Objective Video Quality Measurements”, WWIC'10: Proceedings of the 8th international conference on Wired/Wireless Internet Communications, 2010, 240-251.