High Level Synthesis of Embedded Systems Targeting Reconfigurable Architectures

Fateh Boutekkouk
Department of Computer Science
University of Larbi ben M'hedi
Route de Constantine, PO 358, Oum El Bouaghi, Algeria
fateh_boutekkouk@yahoo.fr

***Abstract*- In this paper, we propose and validate a flow integrating the Unified Modelling Language UML with existing high level synthesis tools targeting reconfigurable architectures. The flow tries to take advantages of UML as a standard for visual object modelling and functional simulation and some CAD tools for profiling, simulation and high level synthesis that work with Xilinx platforms. In order to validate our proposed flow, a case study on the H264 decoder is illustrated.**

I. INTRODUCTION

The productivity gap between semiconductor technology and methodology and tool support has become one of the biggest challenges in embedded systems design. To deal with this problem, specialists in the field have resorted to software engineering and borrowed from it many ideas and technologies to close this gap.

Since embedded systems development requires collaboration between customers, software and hardware teams, a visual common language is preferable to eliminate misunderstandings that can occur. This language must be able to capture customer requirements and then proceeds toward efficient software and hardware implementations in a well defined design flow. We believe that if done correctly, the Unified Modelling Language (UML) can be such a language.

UML [2] is a graphical object-oriented modelling language, originally, was used in software systems. The use of such graphical notation help designer to understand, capture and analyze the client requirements at early stages of development in a semiformal manner. In its basic form, it is applicable to a wide variety of systems (open language).

However, several key attributes of UML are important to embedded systems:

1. UML is abstract, and designers can focus on the high-level characteristics of the system, rather than implementation-specific factors.
2. Hardware and software designers would share a common language.
3. A rich set of notations suited for modelling different points of view.
4. Support for object-based structural decomposition and refinement.
5. Support for non functional constraints modelling.

Recent works aim at generating hardware description languages like VHDL, and SystemC from UML diagrams. The generated code is used either for simulation or synthesis purposes [3].

Despite of the effort in the direction of UML-based system-level design, there is no consistent design flow for embedded systems and the proposed methodologies and associated tools still lack completeness and interoperability. For this reason, many UML2.0 profiles have been proposed by both academia and industry. According to authors, UML2.0 can be tailored to different application domains by the definition of profiles. A profile extends an application specific UML sub-set using extension mechanisms offered by UML like stereotypes, constraints, and tagged values. Furthermore a profile must provide a methodology [3].

Our purpose is not to define a new UML profile, instead of, we try to create a bridge between UML and some existing high level synthesis tools targeting reconfigurable architectures. We note that UML will not replace the well practiced languages and CAD tools, instead, it builds on the top of them an abstract visual layer to facilitate the task of synthesis especially for software designers those are not familiar with hardware domain in general and reconfigurable architectures in particular. The paper is organized as follows: in Section 2 we first overview the related work. Our proposed flow is detailed in section 3. Section 4 is dedicated to discuss some results on the H264 decoder before concluding.

II. RELATED WORK

We can classify existing co-design flows targeting reconfigurable architectures into three main classes: conventional programming language, SystemC, and UML.

Conventional programming languages that are most used in system specification of embedded systems are C/C++, Java. Several approaches of design entry in C for reconfigurable coprocessors have been proposed [4, 8, 9, 10]. A compilation source application in C to a CPU and reconfigurable

co-processor is proposed by Callahan et al. [4]. Their hardware/software partitioning process is done at the basic block level. Nimble [9] is a framework that automatically compiles system-level applications specified in C to executables on the embedded reconfigurable architecture. In this framework, hardware/software partitioning algorithm performs fine-grained partitioning (at loop and basic-block levels) of an application. In contrast to these approaches our partitioning process is done at the method level. Java based design flow for networked reconfigurable systems is first proposed by Fleischmann et al. [6, 7].

The proposed co-design environment called JACoP, which contains a run-time manager to schedule methods for execution either on the Java virtual machine (JVM) or on the reconfigurable hardware. Such programming languages have the advantage of being executable, and thereby facilitate early verification and simulation. However, for the purpose of system level specifications, the use of these languages does not satisfy all the requirements. SystemC is recently used as system-level specification languages [11, 12].

Pelkonen et al. [12] proposed a system-level modelling methodology of dynamically reconfigurable hardware using SystemC. This methodology allows users to do design space exploration at the system-level, without the need to map the design first to an actual technology implementation. However, this methodology is far from complete; the accuracy of the results is required for further investigations.

Using UML as a front end for co-design flow for embedded systems is still quite new.

In [1] Beierlein et al. presented a UML-based co-design environment for run-time reconfigurable architectures, called Model Compiler for Configurable Architecture (MOCCA). They use the UML throughout all phases of development, from specification to synthesis. The concept of hardware-software co-design, Model Driven Architecture (MDA), and platform-based design are used in proposed development approach. They extended the UML by an action language to enable model execution, early verification and simulation, design space exploration, and automated and complete synthesis.

The work in [14], a UML-based design flow for Dynamic Reconfigurable Computing Systems (DRCS) is proposed. The proposed design flow is targeted at the execution speedup of functional algorithms in DRCS and at the reduction of the complexity and time-consuming efforts in designing DRCS. In particular, the most notable feature of the proposed design flow is a HW-SW partitioning methodology based on the UML 2.0 sequence diagram, called *Dynamic Bitstream Partitioning on Sequence Diagram (DBPSD)*. Besides, partitioning guidelines are also included in DBPSD to help designers make prudent partitioning decisions at the class method granularity.

Contrary to this work, our flow targets the enhancement of global system performance and performs the hardware/software co-simulation and synthesis tasks on reconfigurable architectures starting from UML models. The Hardware/Software partitioning is performed at the method granularity on objects diagram. Of course, we can perform partitioning at a coarse grained level such as object or fine grained level like elementary blocs. We choose the method level because it is situated between coarse and fine grained levels.

III. OUR PROPOSED FLOW

We have developed a design flow and related supporting tools. Figure 1 shows the detailed view of the proposed design flow, which is separated into four phases: specification, exploration, generation and integration. Our proposed flow starts by establishing UML models for both application and architecture.

We have adopted Rhapsody7.2 [13] as our UML modelling tool. After drawing UML models in Rhapsody, the tool can generate C, C++, Ada or Java code. Rhapsody provides an Object Execution Framework (OXF) which enables execution of objects.

Rhapsody in C is targeting embedded software development and makes it possible to create a platform-independent model. It covers automated code generation in C from UML diagrams including configuration of environment and *makefile* generation. This approach makes it possible to set up a cross compiler and linker to be used for the target platform. Visual models can be executed and animated for early verification and test. It also provides a good support for an iterative and agile development process. Rhapsody enables reuse of old legacy code by wrapping it into libraries linked with the model, and then only imports the header files into Rhapsody model. It is possible to configure how the Rhapsody code is generated by changing the properties for project, package or class. In some situations, this could be required to optimize the code size. In our case, application structural model is presented through object diagrams.

Each object is stereotyped with “singleton”. We use this stereotype because we have exactly one instance of each object. The internal behaviours of objects methods are implemented directly in the C programming language which is independent from any platform. In order to avoid the generation of dynamic variables, we customize the code generation process in Rhapsody.

Parallel to application modelling, Hardware architecture is modelled via UML structure diagram with a set of defined stereotypes and tagged values

that characterize hardware components and topology of the target architecture.

In our case, the target architecture is a reconfigurable platform (Xilinx) [15] with processor cores (PowerPC, MicroBlaze), programmable HW (Virtex4), Peripherals, HW/SW interfaces (FSL, APU, OPB, PLB), IP cores, and Operating System (OS) support for reconfigurable HW accelerator (linux).

After the generation of the application executable and the behaviour is verified, the next step consists in SW/HW exploration. In this step, the designer profiles the application executable code and analyzes the communication workload between methods.

According to the results of profiling and communication workload, the designer identifies the methods to be implemented in hardware and selects the communication scheme (i.e. point to point versus shared memory).

HW/SW partitioning is done at the method level in objects diagram by definition of a new stereotype called "HW" with a tagged value that specifies the IP name. We write a macro using VB API integrated in the Rhapsody environment to parse HW and SW methods and all calls from and to these methods. At this stage, we can for instance generate hardware wrappers and software drivers automatically. For this purpose, we have to introduce wrappers and drivers templates as header files in C. This is possible due to facilities offered by Rhapsody. Hardware methods serve as inputs for the Catapult HLS commercial tool for automatic VHDL code generation from C code.

Using such tool, the designer can optimize his/her design by applying some HLS optimization techniques such as loop unrolling, etc., and generate the RTL scheme of the design. He can also make a more accurate performance and area estimations. The last step consists in the integration of C/VHDL codes using XPS from Xilinx.

Generally, there are two ways to integrate a customized IP core into a MicroBlaze based embedded soft processor system. One way is to connect the IP on the On-chip Peripheral Bus (OPB). The OPB is part of the IBM Core ConnectTM on-chip bus standard. The second way is to connect the user IP to the MicroBlaze dedicated Fast Simplex Link (FSL) bus system. If the application is time-critical, the user IP should be connected to the FSL bus system; otherwise, it can be connected as a slave or master on the OPB. If the customized core is connected to the dedicated FSL interface, it is then possible to use predefined C functions to use the user core in the application software [15].

IV. CASE STUDY

As a case study, we have tested our proposed methodology on the H264 decoder. The functional block diagram of the H264 is shown in figure2.

Figure 3 shows the UML object diagram for the H264 decoder. It is comprised of eighteen (18) singleton objects.

All methods are implemented in C code. Figure 4 shows the *coretrans* object that includes the inverse transform function (enter) which is stereotyped with "HW".

Figure 5 shows the UML modelling for the MicroBlaze soft core. Table 1 illustrates the profiling results for the different methods of the H264 decoder.

A. *Code generation*

This section focuses on the inverse transform function and its translation to hardware using HLS with the logic needed to interface the MicroBlaze processor.

First, we generate RTL code using Mentor Catapult from pure C code. The use of such tool helps greatly the VHDL code generation problem, but manual analysis is likely to be unavoidable to address efficiently the sensitive problem of interfaces. In our case, the inverse transform function processes small and homogeneous 4x4 blocks, so a point to point communication looks like a suited solution. In this particular case, the processing in itself is fast (a few cycles) compared to the data transfers (16 data * 2).

Efficient transfers are mandatory to grant effective and significant speedups, and this can difficultly be processed efficiently without manual intervention.

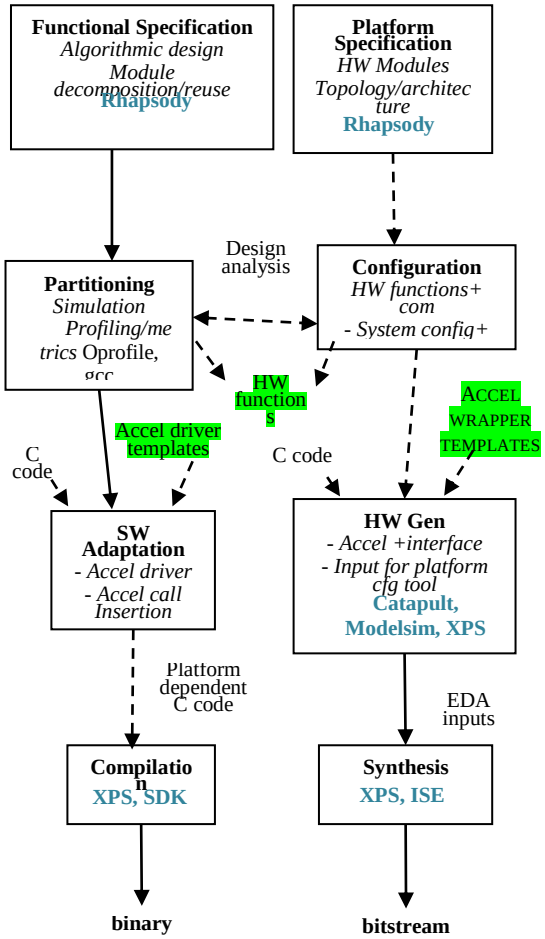

Fig. 1. Flow based on cooperation of UML and EDA tools. Full arrows represent automatic code generation, dotted lines are manual or semi automated steps. Transversal steps such as validation are not represented but should cover all abstraction

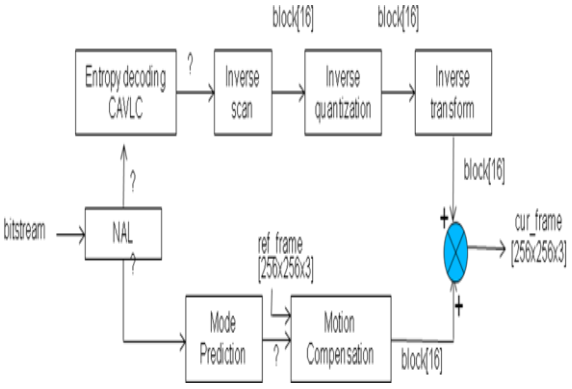

Fig. 2. Functional blocs diagram of H264 decoder

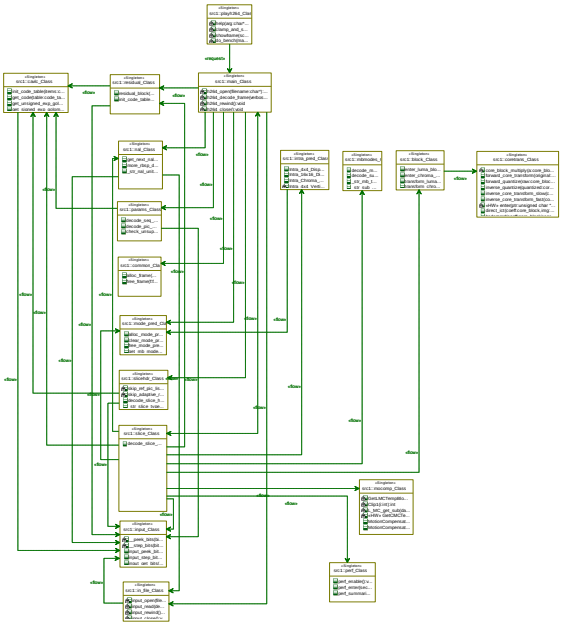

Fig. 3. UML object diagram of H264 decoder

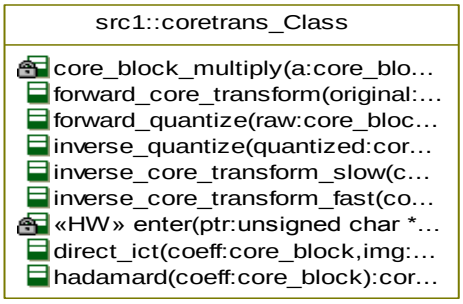

Fig. 4. Coretrans object including the enter method (inverse transform) stereotyped by 'HW'

We defined a VHDL template wrapper that extends the original FSL templates provided by Xilinx tools, and able to consider the processing time of the accelerator with constraints of the data transfers. We also developed template driver functions for the accelerator to be called from the application code. Both templates have been developed with the concern to be applicable in general cases, some cases may require more or less adaptation depending on the data communication requirements.

In our case study, the RTL VHDL code is generated by a HLS tool (CatapultC Synthesis) this way, with the following simple interface: 16 block_in ports, 16 block_out ports plus a *clk* and *rst* signals. Those signals are then wrapped to be compatible with the FSL interface that will make possible the connection of a coprocessor to the MicroBlaze processor.

Figure 6 shows the synthesis result of the inverse transform method.

A wrapper partially generated by XPS is thus added. The original FSM has to be modified with a wait for the end of co-processing before writing back the data. The software interface in this case is basic through the use of two assembly instructions (put/get).

B. Global performance discussion

A realistic evaluation of speedup must take into account the penalty of data transfers to/from the coprocessor. In this evaluation, we have thus compared the C code of the inverse transform executing on the PowerPC, with a C code making a coprocessor call with exactly the same data sets. The results is an acceleration of 2.41, while on a pure processing point of view, this acceleration amount is approximate. This is due to data communication that is sequential with FSL.

TABLE I
RESULTS OF PROFILING FOR THE H264 DECODER FUNCTIONS

%	Symbol Name	Functionality
25.4309	GetLMCTempBlock	Motion compensation
21.0332	L_MC_get_sub	Motion compensation
10.7746	Clip1	Motion compensation
9.6356	MotionCompensateT	Motion compensation
6.9927	GetCMCTempBlock	Motion compensation
5.1527	inverse_quantize	Inverse quantization
5.1203	direct_ict	Inverse transform
4.2957	Enter	Inverse transform
3.3019	coeff_scan	Inverse Scan
1.1246	get_code	CAVLC
0.8510	decode_slice_data	--
0.6445	enter_luma_block	--
0.6193	residual_block	--
0.5533	FillMVs	Mode prediction
0.5017	MotionCompensateM	Motion compensation
0.3613	input_get_one_bit	--

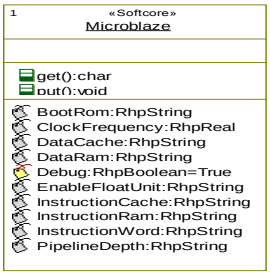

Fig. 5. UML modelling of the MicroBlaze soft core

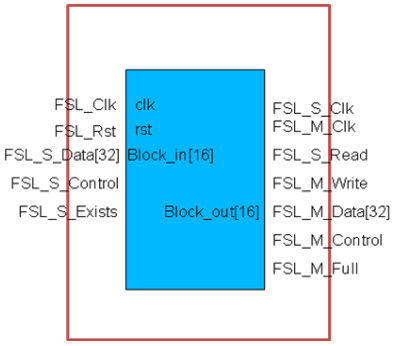

Fig. 6. Result of the inverse transform method synthesis

IV. CONCLUSION AND FUTURE WORK

In this paper, we presented a new flow that uses UML as a front end for embedded systems co-design targeting a reconfigurable architecture. The proposed flow tries to take advantages of both UML and EDA design tools for high level synthesis and co-simulation. As a first validation of our flow, we have applied the flow to model the H264 decoder.

According to profiling results, designer decides which method will be implemented in hardware. This decision is made manually and depends on designer experience. The designer also selects the most appropriate communication scheme. The subsequent steps consist in VHDL code generation using CatapultC, simulation using ModelSim, and software/hardware co-synthesis using the XPS/SDK tools from Xilinx. As a perspective, we plan to automate the wrappers and drivers generation from UML models. This is possible since the platform including templates for hardware wrappers and software drivers is modelled at UML level.

REFERENCES

[1] T. Beierlein, D. Frohlich, and B. Steinbach. Model-driven compilation of UML-models for reconfigurable architectures. In Proc. of the Second RTAS Workshop on Model-Driven Embedded Systems (MoDES'04), May 2004.

[2] G. Booch, J. Rumbaugh, and I. Jacobson. Unified Modeling Language User Guide. Addison-Wesley, 1999.

[3] F. Boutekkouk, M. Benmohammed, S. Bilavarn, and M. Auguin. UML2.0 profiles for Embedded Systems and Systems On a Chip (SOCs). In JOT (Journal of Object Technology), January 2009.

[4] T. Callahan and J. Wawrzynek. Instruction-level parallelism for reconfigurable computing. In Proc. of the 8th International Workshop on Field-Programmable Logic and Applications, From FPGAs to Computing Paradigm, pages 248–257. Springer-Verlag, Berlin, August 1998.

[5] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. ACM Computing Surveys, 34(2):171–210, June 2002.

[6] J. Fleischmann, K. Buchenrieder, and R. Kress. A hardware/software prototyping environment for dynamically reconfigurable embedded systems. In Proc. of the 6th International Workshop on Hardware/software Codesign, pages 105–109. IEEE Computer Society, March 1998.

[7] J. Fleischmann, K. Buchenrieder, and R. Kress. Java driven codesign and prototyping of networked embedded systems. In Proc. of the 36th ACM/IEEE Design Automation Conference (DAC'99), pages 794–797. ACM Press, June 1999.

[8] S. Kimura, M. Yukishita, Y. Itou, A. Nagoya, M Hirao, and K. Watanabe. A hardware/software codesign method for a general purpose reconfigurable co-processor. In Proc. of the 5th International Workshop on Hardware/Software Co-design (CODES/CASHE'97), pages 147–151. IEEE Computer Society, March 1997.

[9] Y. Li, T. Callahan, E. Darnell, R. Harr, U. Kurkure, and J. Stockwood. Hardware-software co-design of embedded reconfigurable architectures. In Proc. of the 37th ACM/IEEE Design Automation Conference (DAC'00), pages 507–512. IEEE Computer Society, June 2000.

[10] N. Narasimhan, V. Srinivasan, M. Vootukuru, J. Walrath, S. Govindarajan, and R. Vemuri. Rapid prototyping of reconfigurable coprocessors. In Proc. of the International Conference on Application Specific Systems, Architectures, and Processors (ASAP), pages 303–312. IEEE Press, August 1996.

[11] K. D. Nguyen, Z. Sun, P. S. Thiagarajan, and W. F. Wong. Model-driven SoC design via executable UML to SystemC. In Proc. of the 25th IEEE International Real-Time Systems Symposium (RTSS'04), pages 459–468. IEEE Computer Society, December 2004.

[12] A. Pelkonen, K. Masselos, and M. Cupak. System-level modeling of dynamically reconfigurable hardware with SystemC. In Proc. of the 10th Reconfigurable Architectures Workshop (RAW'03), 17th International Symposium on Parallel and Distributed Processing (IPDPS'03), pages 174–181. IEEE Computer Society, April 2003.

[13] Rhapsody case tool reference manual. I-Logix Inc. http://www.ilogix.com.

[14] C.H. Tseng. UML-Based Rapid Prototyping Design Flow for Dynamically Reconfigurable Computing Systems. Master thesis, National Chung Cheng University, China, June 2005.

[15] www.Xilinx.com.