Bridging the High-level Model to Execution Platform for Design Space Exploration and Implementation

M. Benyoussef, J.-F. Boland
Electrical Engineering Department
École de technologie supérieure
Montréal, Québec, Canada

G. Nicolescu, G. Bois
Computer Engineering Department
École Polytechnique de Montréal
Montréal, Québec, Canada

J. Hugues
ISAE/DMIA
Toulouse, France

Abstract: This paper presents a new modeling methodology for embedded systems. This methodology fills the gap between high-level AADL models and the hardware execution platform described at low-level. It enables an architectural exploration phase at different levels of abstraction to refine and increase system's performances. The main objective of the proposed approach is to reduce the complexity of development, while improving system's robustness and enhance product quality. This is achieved through virtual prototyping of the complete system to perform early validation in the design flow.

Key word: Model Based Design, AADL, Virtual Platform, Levels of abstraction, Refinement, Architecture Exploration, Early Validation.

1. Introduction:

The complexity of embedded systems continues to rise rapidly [1], due to the rapid technological changes and the industrial demand that needs more sophisticated software applications and powerful electronics to implement them. Therefore, the need to design competitive embedded systems in terms of reliability, safety and robustness is becoming increasingly necessary.

In order to tackle embedded system design and manage the increasing complexity, engineers are oriented toward using high-level Model-Based Design (MBD) [2]. This approach employs abstraction to purposely omit some details of the system description in early stages. This generalisation process intends to make embedded system description clearer and easier to develop by keeping only essential information. However, high-level models are intrinsically imprecise and different points of view of the model need to be enhanced [7]. Through design refinement, performances of the embedded system are improved and optimized. As an example, the validation of traditional MBD generally considers only functional and timing requirements [8]. Target platform execution constraints are not considered. This can seriously:

- Affect performance analysis.
- Lead to major errors in the integration phase of the system.
- Slow down the production chain in debugging.

To deal with these limitations, new MBD methodologies and tools are needed to simplify the design process by offering a trade-off between design abstraction and accuracy of results through performances analysis. By doing so, the high and low-level of abstraction are bridged together in the same design flow so architectural exploration and refinement could be performed at different levels.

In this paper a new modeling approach is proposed. This approach combines the strength points from Model-Based Design methodologies using the AADL language [6] and virtual prototyping based on virtual platform environments. Our approach uses a tool chain transformation to bridge the high-level models and execution platform while providing rapid refinement and early validation for the execution platform.

The rest of the paper is structured as follows. The second section introduces the related work and highlights our contribution comparing to the related presented approaches. The third section illustrates the proposed methodology with explanation of each of its steps and the forth section shows the experimental results. Finally, the last section concludes with a summary of the work done in this project.

2. Related work

A number of related approaches have been proposed in the literature to support modeling design aspects. In the ANR project [9] authors describe how to use high-level modeling language (AADL) in combination with the Polychrony toolset. Through formal description, the tool allows timing analysis, validation and synthesis early in the design process. In [10] authors develop a new methodology to build and translate AADL models into a distributed application using the BIP tool chain. This approach allows runtime analysis to assess system viability and corrects the behaviour of the system. In [11] authors propose a rapid prototyping platform to develop distributed real-time embedded systems using high-level AADL model. They explain how to check non-functional requirements early in the design cycle using the Ocarina tool to perform timing/scheduling analysis and code generation. In [14] the AADS tool is developed to enable early verification of timing constraints and performances analysis of the AADL specification. The SCoPE tool integrates the POSIX API to support system level simulation. The focus of these related works is to analyse functional and non-functional requirements. Detail performances evaluation of the targeted execution platform is not supported. Our contribution aims to fill this gap by linking a design space exploration tool to higher-level modeling environment.

3. Proposed methodology

This paper presents a modeling framework to support the system co-design and architectural exploration through virtual prototyping. Our approach spans across different abstraction levels where the various elements of the system are refined progressively. At high-level of abstraction, the model contains only the software components, the connections instances and the behavioural description of the application. At low-level, the modeling includes hardware modules of the target platform and the mapping of the SW components on the virtual platform.

The design flow (see Figure 1) is composed of six steps: application specification (1), architecture modeling (2), ATL transformation (3), design space exploration (4), architecture refinement (5) and AADL model generation (6). The following sections present each of these steps.

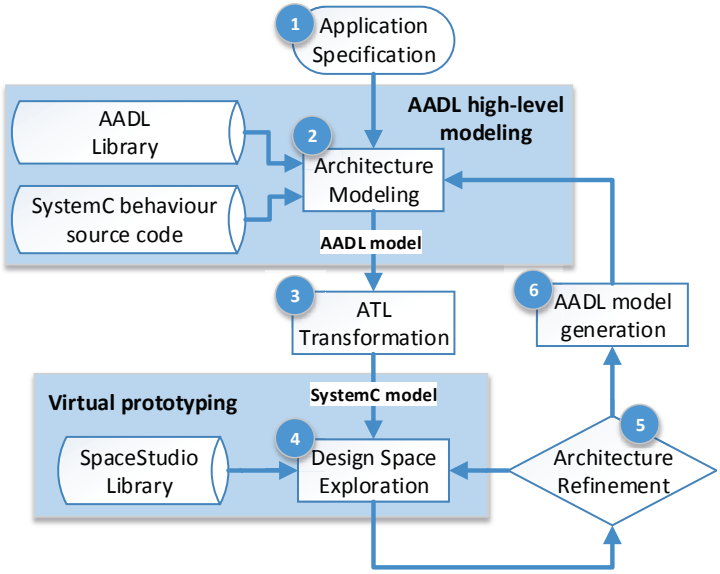

Figure 1 Proposed design flow

3.1. Application Specifications

The design flow begins with the specifications of the application (step 1). This specification can take different format: text documents, chronograms, bloc diagrams, etc.

In order to validate our methodology, an MJPEG decoder application will be used as a case study throughout this paper. Figure 2 presents the main functional blocks of the MJPEG. The input is an MJPEG stream stored in a memory array. Then, the DEMUX block scans the video data and sends the quantization table, the Huffman table and the data stream respectively to IQZZ and VLD blocks. The VLD block performs Huffman decoding, while the IQZZ inverse the quantization and un-zigzag transforms. The IDCT block performs an inverse discrete cosine transform. Finally, the LIBU transform received data into image line for the VGA controller.

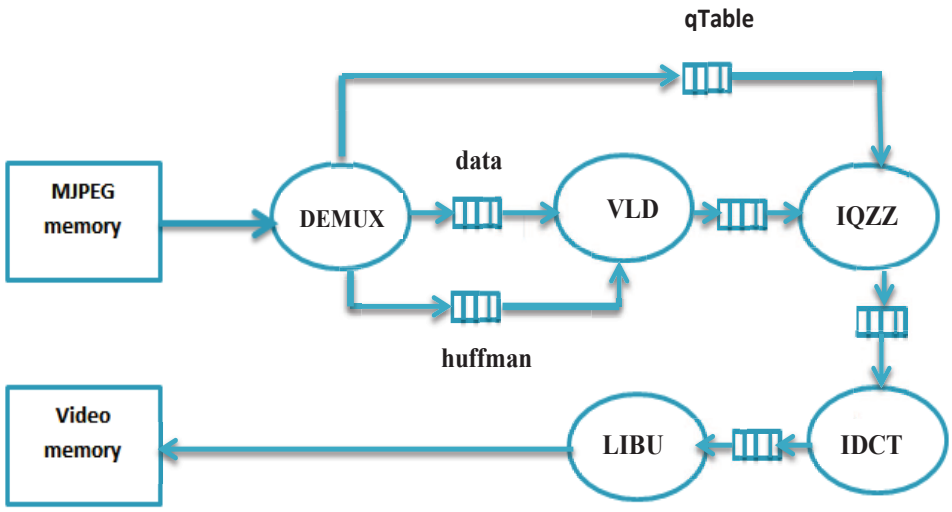

Figure 2 MJPEG decoder block diagram and communication paths

3.2. Architecture modeling

The architecture of the application is formalized in the second step of the design flow. This section gives an overview of the AADL modeling language and presents the details of step 2.

3.2.1. AADL overview

AADL (Architecture Analysis & Design Language) [5] is a modeling language used for real time embedded system design. The various AADL components library allow users to describe the software and the computer platform architecture of the system. Architecture interfaces, components interaction and binding mechanism are defined in the same model. Tools like OSATE2 (Open Source AADL Environment Tool) [6] can be used to create the AADL model and verify functional and non-functional properties.

3.2.2. AADL high-level modeling

Step 2 aims to describe the application the software system architecture. This high-level model is created using the AADL components library. This library contains the semantic to represent threads, processes and communication ports. The behaviour of each block of the application is defined as a separated SystemC source code that will be referenced in the AADL description. Figure 3 shows the block diagram corresponding to the AADL model of the MJPEG decoder application.

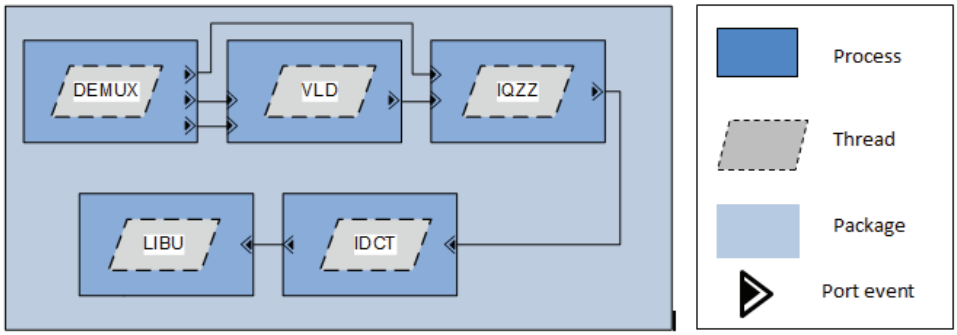

Figure 3 Block diagram corresponding to the AADL model of the MJPEG decoder application

To create the AADL model, a package structure type is used which contains the organisation of the system architecture components. As can be seen in Figure 3 the package contains five process components, which communicate with each other through an AADL port event connection. Each process includes one thread. There are a total of five threads: IDCT_thread, IQZZ_thread, VLD_thread, LIBU_thread, DEMUX_thread. Figure 4 shows an example of a thread description. The AADL subprogram IDCT_Function contains a reference to the SystemC source code to define the true functionality of the thread. Also, the connection interface is described in the AADL feature type section inside the thread declaration.

```
subprogram IDCT_Function
properties
      Source_Language => (System_C);
      Source_Text     => ("My_idct.c");
      Source_Name     => "My_idct";
end IDCT_Function;

thread IDCT
features
      IQZZ_In: in event data port IQZZ_Data;
      IDCT_Out: out event data port IDCT_Data;
properties
    Dispatch_Protocol => Periodic;
    Compute_Entrypoint=> classifier (IDCT_Function);
end IDCT
```

Figure 4 AADL example of a thread description code

3.3. ATL transformation:

The third step of the design flow uses the ATL transformation tool chain to bridge the high-level model to the execution virtual platform low-level developed by the Open People Project [4]. It automatically transforms AADL models to SystemC models. The ATL transformation is based on the semantic language translation to respect the identification rules for SystemC and AADL (For example AADL does not care about upper case and lower case but SystemC does). AADL components are translated into namespace classes to express the AADL structure in SystemC models, for example to every AADL package correspond a C++ namespace. To do that a SystemC runtime library is developed that contain all types and all classes equivalent to AADL concepts. Figure 5 shows an extract code of AADL to SystemC model. So the generated SystemC model defines the software architecture structure, which contains a multi-threaded C++ application that will be mapped on HW/SW co-design platform.

AADL

```
package mjpeg_aadl
public
  thread IDCT
…
  end IDCT;
  thread  implementation
IDCT.impl
…
```

SystemC

```
namespace AADL_mjpeg_aadl {

namespace mjpeg_aadl {class idct
:     public     AADL::threadType
{public:idct(AADL::moduleName
name):AADL::threadType(name){}};}

namespace   mjpeg_aadl   {typedef
My_idct idct_DOT_impl;}…}
```

Figure 5 AADL vs SystemC model

It is important to note that the communication ports in the AADL model are not yet included into the automatic transformation of the AADL model. The communication links will be defined manually in step 4 using Space Studio functions.

3.4. Design space exploration

The backend of our proposed methodology (e.g. Steps 4 and 5 of Figure 1) is achieved by an ESL framework. This framework is enabled by the SpaceStudio™ tool suite, a complete HW/SW co-design platform with the unique ability to transform functions (threads) between hardware (HW) and software (SW) as designers decide on the makeup of their system [12]. In the following sections, we present the design framework and its components.

3.4.1. Overview

From our SystemC model (Figure 1), the system's application is specified as a set of concurrent tasks communicating through explicit interfaces. Next, several potential architectures are provided by the SpaceStudio library, where architectural parameters that may vary include the number of processors and cores, the number of busses, the HW/SW partitioning of tasks, the mapping of software tasks to processor cores and the configuration of architectural components. For each potential architecture and mapping, SpaceStudio automatically generates a SystemC TLM-2.0 virtual platform of the system's hardware components, and embedded software binaries for each processor core in the platform. By taking advantage of the SystemC library definitions and TLM-2.0 interface standards a single language C/C++ can be used. This allows us to create a fully-modeled functional software representation of a hardware/software SoC design. Then a performance assessment of each architecture is enabled by profiling simulations at different levels of abstraction. HW resources and power consumption can also be estimated. Finally, selected architecture(s) is translated in AADL for further analysis.

3.4.2. Virtual prototyping

One of the key elements of an electronic system level (ESL) methodology is the concept of platform-based design. Platform-based design allows extensive reuse of components, which reduces the time-to-market for the first release of a product, maintenance, and subsequent releases [13].

Configuration of the virtual platform

This step consists in configuring the virtual platform of the system using SpaceStudio component library. The designed virtual platform permits the execution of the application model and enables early validation and performance evaluation of the software contents. No hand coding is needed to configure the platform, the user only need to instantiate components from the library.
As shown in Figure 6, the configuration manager option of SpaceStudio allows selecting the desired components (see Available components), as well as modifies module parameters (see Current configuration content). Parameters allow enhancing system performances, performing architectural exploration or varying the configuration type. Examples of parameters are the number of processors, the CPU frequency, the inter-connexion type and its latency, address range, cache type and memory size. To implement our MJPEG decoder application an ARM Corte-A9 MPCore (core 0 and core 1) with an OS on each core (Asymmetric Multiprocessing), two sets of peripherals (e.g. PIC) and a RAM block have been selected and configured.

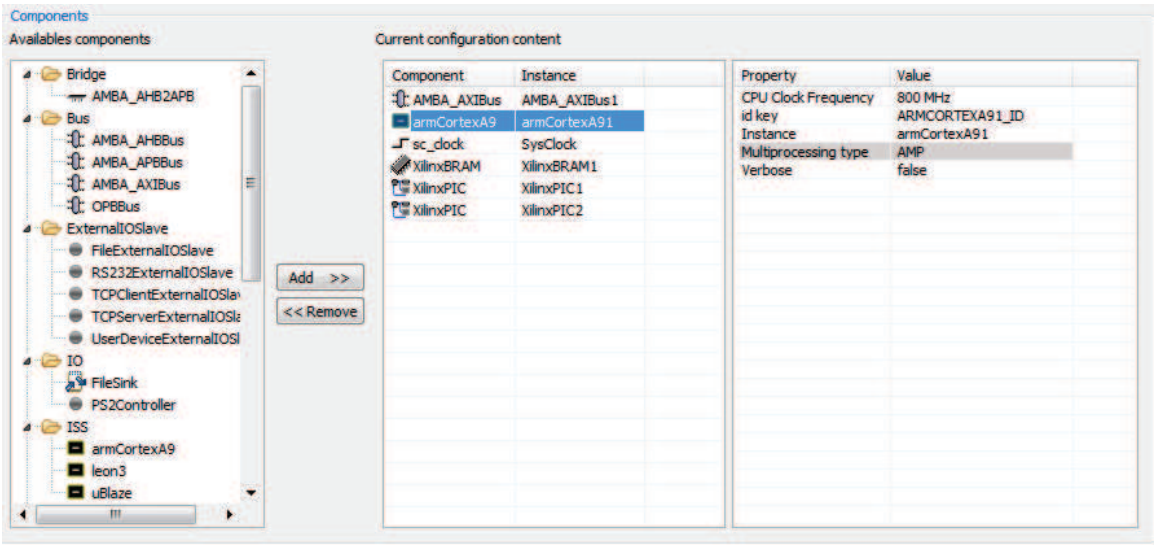

Figure 6 Space Studio virtual platform configuration manager

Mapping process

After configuring the hardware virtual platform, SystemC models generated from step 3 (ATL transformation) are imported in Space Studio. As shown in Figure 7, the five modules of the MJPEG decoder application are now available as "User Blocks" in Space Studio binding table. Then, the mapping process is performed.

Figure 7 illustrates on possible mapping (solution): a hardware/software solution with IDCT and LIBU on core 0, DEMUX, IQZZ on core 1 and VLD connected as a coprocessor on the AMBA AXI channel. Note also that the ARM Cortex-A9, the BRAM, the two PICs and a VGA controller are also connected to the same channel.

Other solutions (mapping) can be determined only by modifying (clicking) the current matrix connexion of Figure 9. Its unique HW/SW transformation capability allows the user to specify and re-specify the mapping of application tasks to either software running on a processor, or as dedicated hardware, without having to re-design or re-code functional blocks and without extensive integration work (e.g., communication/bus interface).

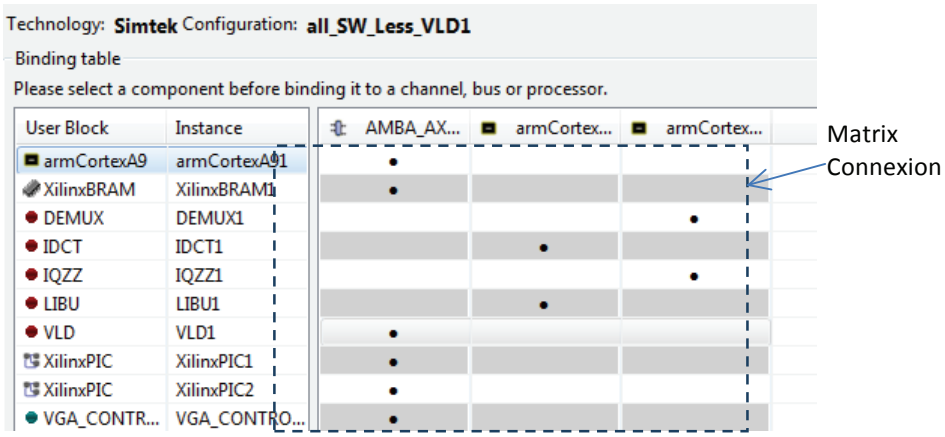

Figure 7 Space Studio binding table used for mapping the application on the hardware virtual platform

Component inter-connexion

For each potential mapping, SpaceStudio automatically generates and builds a SystemC TLM-2.0 virtual platform modeling the hardware components (processor models, busses, memories, peripherals, as well as the hardware mapped application tasks) and their connections. It also

automatically associates the built software binaries to their respective processor models in the platform.

Figure 8 illustrates the link established between platform components through an explicit interface predefined by SpaceStudio. It describes a portion of the generated code that shows some peripheries (ISS_adapter 1 & 2, VGA Controller, XilinxBRAM…) connected to the AXI Bus.This capability reduce development time and coding effort.

```
ISSAdapter2.WriteFifoIFPort[5](ISSAdapter2_FIFO_2.WriteFifoIFExport);
AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_DebugModule1.slave_sock);
AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_ISSAdapter1.slave_sock);
AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_ISSAdapter2.slave_sock);
AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_VGA_CONTROLLER1.slave_sock);
AMBA_AXIBus1.master_sock(AMBA_AXIBus1_SlaveAdapter_XilinxBRAM1.slave_sock);
```

Figure 8 Component Communication Interface

3.5. Architecture refinement

In summarize, this step perform a design exploration by modifying architectural parameters, or by refining the high-level AADL model, by adjusting the hardware/software partitioning, and finally, by keeping the same virtual platform to analyze and improve the impact on the system's performance. Even then, the rapidity of using this agile approach results in a faster loop (e.g. hours) of refinement compare to using traditional RTL-based approaches (e.g. days and weeks).

3.6. AADL model generation

Once the optimal hardware architecture is constituted and the best solution met the initial specification, another type of assessment could be conducted at this level; an AADL hardware platform model of the solution will be generated from SpaceStudio tool in order to verify and analyse other aspects that was not be covered by this tool (e.g.: Reliability, Safety, Security, Robustness, Cost…), this phase not yet developed in this work, will be explored for the next steps to improve our design flow.

4. Experimental results

The Figure 9 below show the platform architecture constructed; it include an arm cortex a9 processor, an AMBA-AXI bus, 2 processor interrupt controller, memory block, VGA controller and 2 ISS adapter. The ARM processor has a frequency of 800 MHz and run in AMP (Asymetric Multi-Processing)mode, with uc II OS/RTOS .

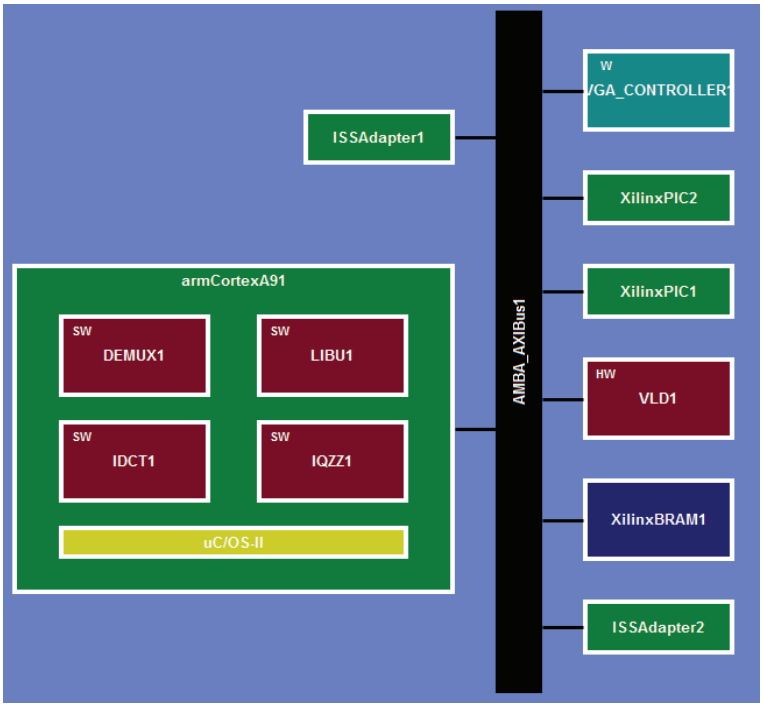

Figure 9 Execution virtual platform

Once the virtual prototype of the complete system has been created, the execution of the application launched, captured the simulation results and started evaluating the system performances. Figure 10 shows simulation results for a processors load. More precisely, for functions targeted as software running on processors, data logs can be used to generate pie charts to analyze loading. Here the load is illustrated on a Cortex A9 for all software mapping of the MJPEG.

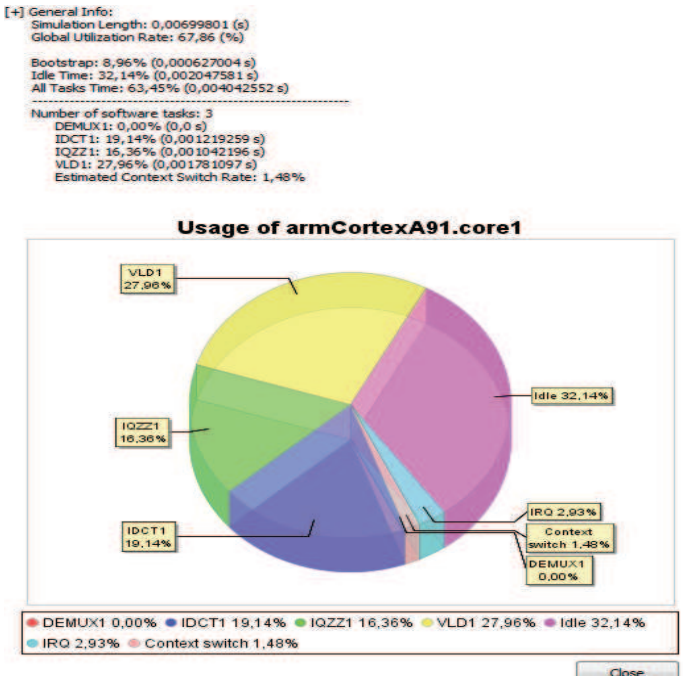

Figure 10 Simulation results for a processor load

However, system architects can proceed to architecture refinement process to speed up performance by reallocating software tasks between processors, introducing additional processors, and/or retargeting tasks to hardware as co-processors on a bus/channel. Tableau 1

show an example of 5 candidate architectures that have been explored according to different mapping possibilities in only few minutes, For instance, as can be observed in Figure x that VLD1 has the highest load (28%) on the Cortex A9. Therefore, it can be a good decision to move VLD1 from SW to HW (Figure 9).

Tableau 1 Architectural exploration according to the mapping process

Architecture candidates	Mapping on SW	Mapping HW
1	all	-
2	DEMUX1, IQZZ1, LIBU1, IDCT1	VLD1
3	DEMUX1, LIBU1, IQZZ1	VLD1, IDCT1
4	DEMUX1, LIBU1	VLD1, IDCT1, , IQZZ1
5	DEMUX1	LIBU1, VLD1, IDCT1, IQZZ1

Figure 11 show how performance increases as more functions are targeted to hardware (HW) versus software (SW), starting from an All-SW mapping.

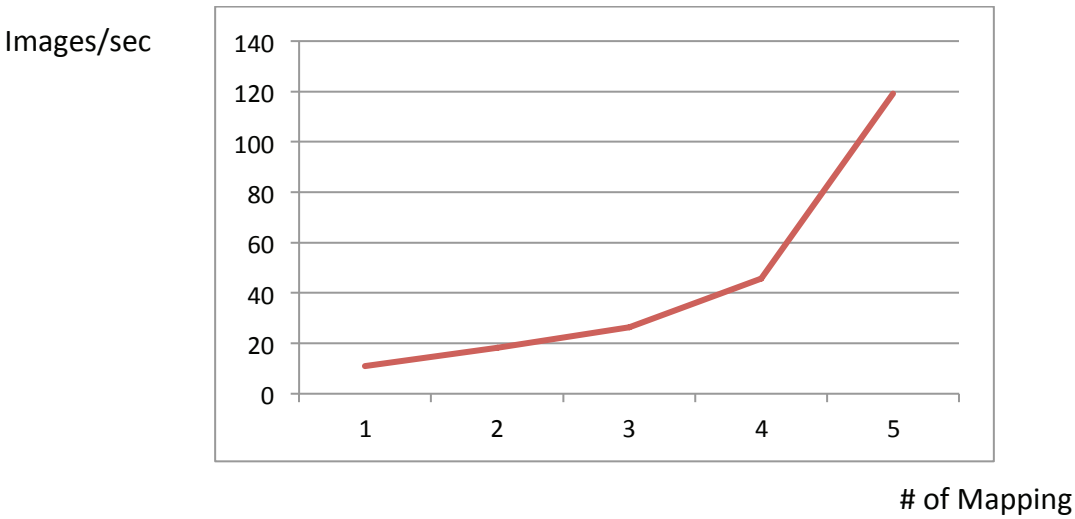

Figure 11 Speed up performances from mapping 1 (1x) to 5 (12x)

5. Conclusion

In this work we have proposed a novel modeling methodology for embedded system design that reduces considerably the complexity of design, development time and increases system performance. The proposed approach consists of developing an AADL high-level model of the system behaviour, and then performs a SystemC code generation to be executed on a virtual customisable platform.

To improve the design flow we intend to explore other solutions as future work; (1) we will include the communication interface in the transformation chain and ensure its automatic extension from the AADL model to the virtual prototyping environment, (2) a new procedure will be developed to generate an AADL model from Space studio tool specification concerning the virtual platform to perform many non-functional analysis, (3).finally an avionic application will be used as a case study to test and validate the final design flow.

6. Acknowledgment

This work as been developed as part of the CRIAQ AVIO509 project and financially supported by CMC Electronics, CAE, CRSNG, MITACS and CRIAQ.

7. References

[1] Kopetz, Hermann, "The Complexity Challenge in Embedded System Design," Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on , vol., no., pp.3,12, 5-7 May 2008.

[2] Kaynak, O.; Jezernik, K.; Szeghegyi, Agnes, "Complexity reduction of rule based models: a survey," Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on, vol.2, no., pp.1216, 1221, 2002.

[3] Peter H. Feiler , David P. Gluch; ”Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language”,SEI Book 2012.

[4] P.Bomel, D. Blouin, “Functional Validation of AADL Models via model Transformation to SystemC with ATL”5th international workshop on Model Based Architecting and construction of embedded system, Austria 2012.

[5] Feiler, P.H.; Lewis, Bruce A.; Vestal, S., "The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems," *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, vol., no., pp.1206,1211, 4-6 Oct. 2006.

[6] Peter Feiler, “Open Source AADL Tool Environment (OSATE)”, AADL Workshop, Paris, October 2005.

[7] Verelst, J., "The influence of the level of abstraction on the evolvability of conceptual models of information systems," *Empirical Software Engineering, 2004. ISESE '04. Proceedings. 2004 International Symposium on* , vol., no., pp.17,26, 19-20 Aug. 2004.

[8] Fleurey, F; Steel, J; Baudry, B., "Validation in model-driven engineering: testing model transformations", First International Workshop on Model Design and Validation, Copenhagen, Denmark, 2 Nov 2004.

[9] Yue Ma; Huafeng Yu; Gautier, T.; Talpin, J.; Besnard, L.; Le Guernic, P., "System synthesis from AADL using Polychrony," *Electronic System Level Synthesis Conference (ESLsyn), 2011*, vol., no., pp.1,6, 5-6 June 2011.

[10] M. Y. Chkouri and M. Bozga. “Prototyping of Distributed Embedded Systems Using AADL”, the proceedings of the ACESMB 2009 workshop conjunction with MODELS 2009.

[11] J. Hugues, B. Zalila, L.Pautet, and F. Kordon. 2008. From the prototype to the final embedded system using the Ocarina AADL tool suite. *ACM Trans. Embed. Comput. Syst.* 7, 4, Article 42 (August 2008).

[12] Moss, L., Guérard, H., Dare, G., Bois, G., "Recent Experience on an ESL Framework for Rapid Design Exploration using Hardware-Software Codesign for ARM-Based FPGAs", SAME 2012 Conference, October 2 & 3, 2013.

[13] Bois, G., Moss, L., Filion, L., and Fontaine, S., “Experiences based on a virtual platform in ESL Models and their Application,” Eds. Brian Bailey and Grant Martin, Springer, pp. 273-308, 2010.

[14] R. Varona-Gómez, E. Villar, A.I. Rodríguez, F. Ferrero, E. Alaña, “Architectural Optimization & Design of Embedded Systems based on AADL Performance Analysis”, American Journal of Computer Architecture 2012.