Design Space Exploration: Bridging the Gap Between High-Level Models and Virtual ExecutionPlatforms

M. Benyoussef, J.-F. Boland, G. Nicolescu, G. Bois, J. Hugues

Design Space Exploration: Bridging the Gap Between High-Level Models and Virtual Execution Platforms

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
DMIA Department
ISAE
Toulouse, France

Abstract: This paper presents a novel embedded systems modeling framework that fills the gap between high-level AADL models and low-level hardware virtual execution platforms. This approach allows refinement and improvement of system performance through exploration of architectures at different levels of abstraction. The aim of the proposed approach is to achieve virtual prototyping of the complete system in order to allow validation to begin early in the design flow, thereby accelerating its development while improving system performances. The proposed framework is validated using an MJPEG video decoder application. Experimental results show how virtual prototyping allows the system architect to fine-tune performances by reallocating tasks between processors and/or retargeting tasks to hardware modules. Fives candidate architectures have been explored to find an optimal solution that delivers 12 times the performances compared to an all-software mapping.

Keywords: Model-based engineering, AADL, SystemC, virtual prototyping, refinement, architecture exploration, early validation, ESL

1. Introduction

The complexity of embedded systems continues to rise rapidly [1] due to technological advances and industrial demand for electronics powerful enough to implement increasingly sophisticated software applications. High-quality designs are becoming increasingly necessary in order to meet the demand for embedded systems that are competitive in terms of reliability, safety and robustness.

In order to meet the challenge of improving embedded system design and managing the increasing complexity, engineers are turning to model-based engineering (MBE) [2]. This generalisation process uses abstraction to limit the details of system description to essential information only. However, this simplification comes at a cost: high-level models are inherently imprecise and hide features that may need to evolve in order to optimize system performance. Meanwhile, validation of traditional model-based design generally considers functional and timing requirements only, and ignores target platform execution constraints [3]. This can seriously affect performance analysis, lead to major errors in the integration phase of the system, and cause a bottleneck at the debugging stage of the product development chain.

One of the goals of new model-based engineering methodologies and tools is to simplify the design process by offering a trade-off between design abstraction and accuracy of results through performance analysis. High and low levels of abstraction are thus bridged in the same design flow so that architectural exploration and refinement can be performed at different levels.

In this paper a new modeling framework is proposed, combining the advantages of model-based engineering methodologies using the AADL language [4] and virtual prototyping based on virtual platform environments. Our approach uses a tool chain transformation to bridge the gap between high-level models and the virtual execution platform to allows more accurate design space exploration.

The following section introduces related work and highlights our contribution while comparing related approaches. The third section illustrates the proposed methodology with explanation of each of its steps, and the fourth section shows our experimental results. Finally, the last section concludes with a summary of the work conducted so far on this project and a list of future work to improve the methodology.

2. Related work

Various related approaches have been proposed in the literature to support different aspects of design modeling. The authors of an ANR-funded project describe the use of high-level modeling language (AADL) in combination with the Polychrony toolset [5]. Through formal description, the tool allows timing analysis, validation and synthesis early in the design process. Others have developed a new methodology to build and translate AADL models into a distributed application using the BIP tool chain [6]. This approach allows the use of runtime analysis to assess system viability and to refine system behaviour. The use of a rapid prototyping platform to develop distributed real-time embedded systems using high-level AADL models has been proposed [7]. These authors 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 another study, the AADS tool is developed to allow early verification of timing constraints and performance analysis of the AADL specification [8]. The SCoPE tool integrates POSIX API to support system-level simulation. The focus of these studies is analysis of functional and non-functional requirements. Detailed performance evaluation of the targeted execution platform is not supported. Our contribution aims to fill this gap by linking a design-space exploration tool to a higher-level modeling environment.

3. Proposed methodology

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

Presented in Figure 1, the design flow is composed of six steps: specification of the application (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 in detail.

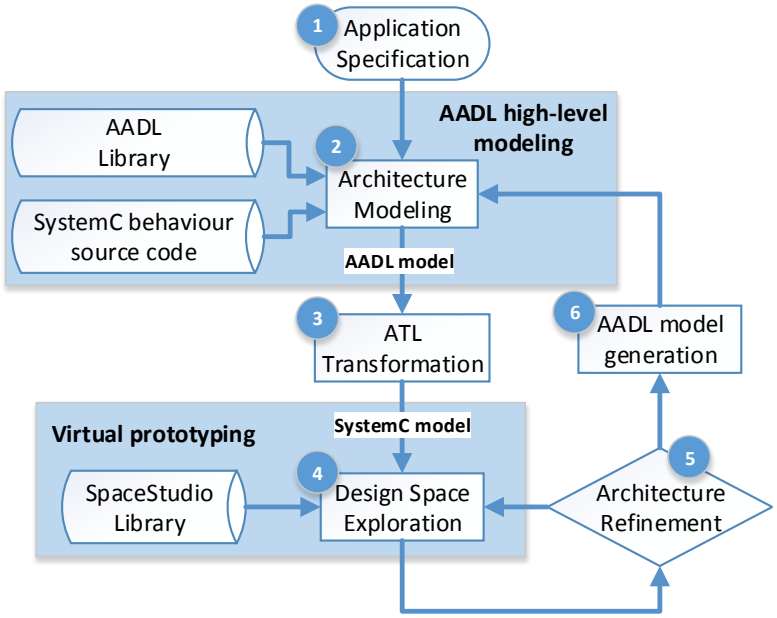

Figure 1 – Proposed design flow

3.1 Specification of the application

The design flow begins with the specification of the application (Step 1). Various formats are suitable: text documents, chronograms, block diagrams and so on. In order to validate our methodology, an MJPEG video decoder application is 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. The DEMUX block then scans the video data and sends the quantization and Huffman tables respectively to the IQZZ and VLD blocks along with the data stream. The VLD block performs Huffman decoding, while the IQZZ inverses the quantization followed by an un-zigzag transform. The IDCT block performs an inverse discrete cosine transform. Finally, the LIBU transforms received data into image lines for the VGA controller.

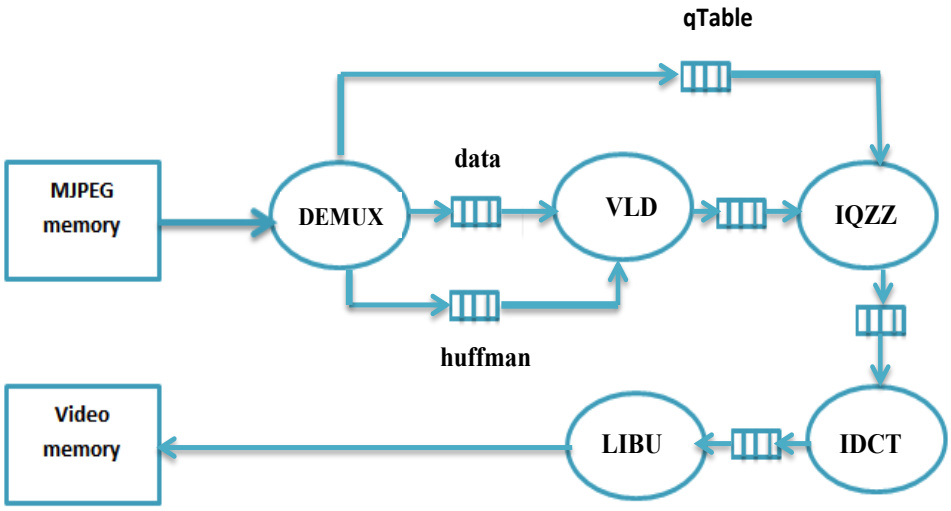

Figure 2 – MJPEG decoder block diagram and communication paths

3.2 Architecture modeling

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

3.2.1 Overview of AADL

Architecture analysis and design language or AADL [9] is a modeling language used for real-time embedded system design. The various AADL components (in a library) allow users to define the software and computer platform architectures of the system. Architecture interfaces, component interactions and binding mechanisms are defined in the same model. Tools such as OSATE 2 (open-source AADL tool environment) [10] can be used to create the AADL model and verify functional and non-functional properties.

3.2.2 High-level modeling using AADL

The purpose of step 2 is to describe the application software system architecture. This high-level model is created using the AADL components library. This library contains the semantics used to represent threads, processes and communication ports. The behaviour of each block in the application is defined using a separate 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.

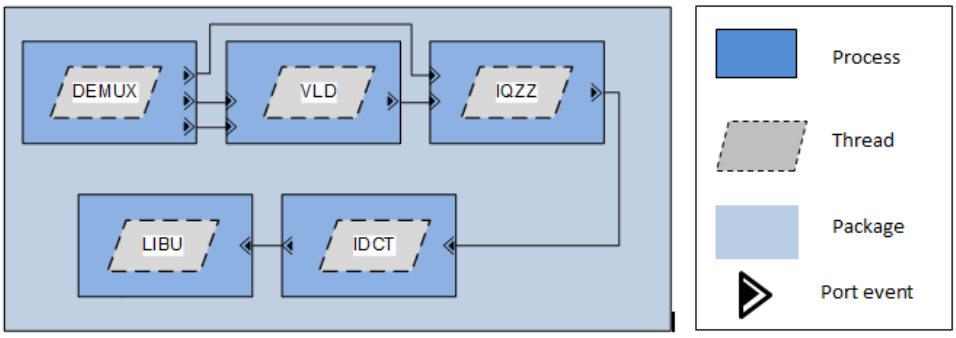

Figure 3 – Block diagram corresponding to the AADL model of an MJPEG decoder application

To create the AADL model, we used a package structure containing the organisation of the system architecture components. As shown in Figure 3, the package contains five process components that communicate with each other through an AADL port event connection. Each process includes one of the five threads: IDCT, IQZZ, VLD, LIBU, or DEMUX. Figure 4 shows an example of a thread description. The AADL subprogram IDCT_Function contains a reference to the SystemC source code that defines the true functionality of the thread. 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 – Example of AADL thread description code for the IDCT function

3.3 ATL model transformation

The third step of the design flow uses a transformation tool chain to bridge the high-level model to the low-level virtual execution platform. This tool chain, based on the ATL model transformation language, was developed by the Open People Project [11] to transform AADL models to SystemC models automatically. AADL components are translated into namespace classes to express the AADL structure in SystemC models. For example, for every AADL package there is a corresponding C++ namespace. A SystemC runtime library containing all types and all classes equivalent to AADL concepts was developed for this purpose. Figure 5 shows an excerpt of equivalent AADL and SystemC codes for a model. The generated SystemC model defines the software architecture, which contains a multi-threaded C++ application that will be mapped onto a hardware/software co-design platform.

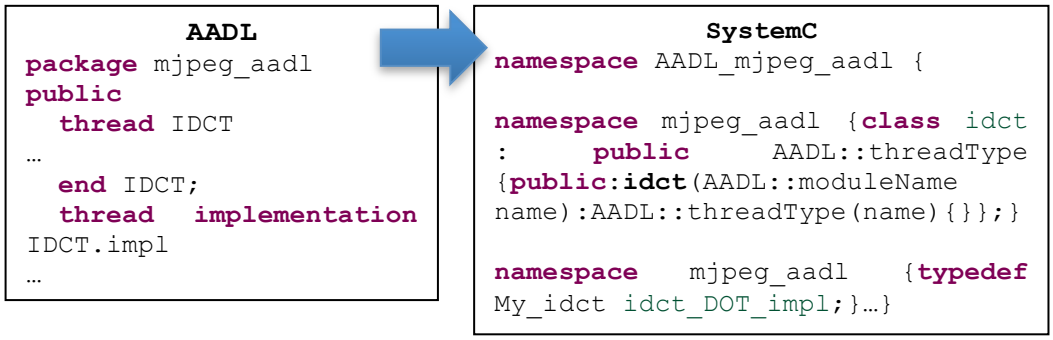

Figure 5 – AADL to SystemC code transformation

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

3.4 Design space exploration

The back end of our proposed methodology (i.e. steps 4 and 5 in Figure 1) is obtained using an electronic-system-level (ESL) framework enabled by the SpaceStudio™ tool suite. This is a complete hardware/software co-design platform with the unique ability to transform functions (threads) between hardware and software 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

In the SystemC model, the application is specified as a set of concurrent tasks communicating through explicit interfaces (input of Step 4). The SpaceStudio library then provides several possible architectures, of which the variables include the number of processors and cores, the number of buses, the hardware/software partitioning of tasks, the mapping of software tasks to processor cores, and architectural component configuration. For each possible architecture and mapping, SpaceStudio automatically generates a SystemC TLM-2.0 virtual platform of the system 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. Simulations are then conducted at different levels of abstraction in order to obtain a profile of architecture performance. Hardware resource usage and power consumption can also be estimated. Finally, the selected architecture(s) is (are) translated into 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 re-use of components, which reduces the design cycle time (time-to-market) for the first release of a product, maintenance, and subsequent releases [13]. The following presents the three steps to create the virtual platform in SpaceStudio.

1) Configuration of the virtual platform

This step consists of configuring the virtual platform of the system using the SpaceStudio component library. The virtual platform allows execution of the application model as well as software early validation and performance evaluation. No manual coding is needed to configure the platform; the user needs only to instantiate components from the library. As shown in Figure 6, the configuration manager option of SpaceStudio allows us to select the desired components (“Available components”) and to modify module parameters (“Current configuration content”) in order to enhance system performance, perform architectural exploration or vary the configuration type. Examples of typical parameters are the number of processors, the CPU frequency, inter-connection type and latency, address range, cache type and memory

size. To implement our MJPEG decoder application, an ARM Cortex-A9 dual-core (core 0 and core 1) with an OS on each (asymmetric multiprocessing), two sets of peripherals (e.g. PIC) and a RAM block were selected and configured.

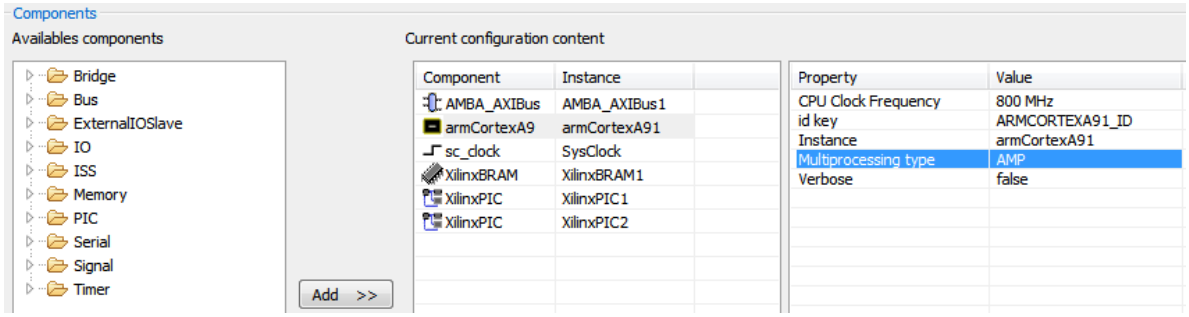

Figure 6 – SpaceStudio virtual platform configuration manager

2) Mapping process

Once the hardware virtual platform is configured, the SystemC models generated in Step 3 (ATL transformation) are imported into SpaceStudio. As shown in Figure 7, the five modules of the MJPEG decoder application are now available as "User Blocks" in the SpaceStudio binding table. The mapping process is performed at this stage. Figure 7 illustrates one possible mapping solution: a hardware/software solution with IDCT and LIBU on core 0, DEMUX, IQZZ on core 1 and VLD connected as a co-processor on the AMBA AXI channel. Note that the ARM Cortex-A9, the BRAM, the two PICs and a VGA controller are also connected to that channel. Other mapping solutions can be determined by modifying the connection matrix of Figure 7. The unique hardware/software transformation capability of SpaceStudio allows the user to specify and re-specify the mapping of application tasks either to 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. on the communication/bus interface).

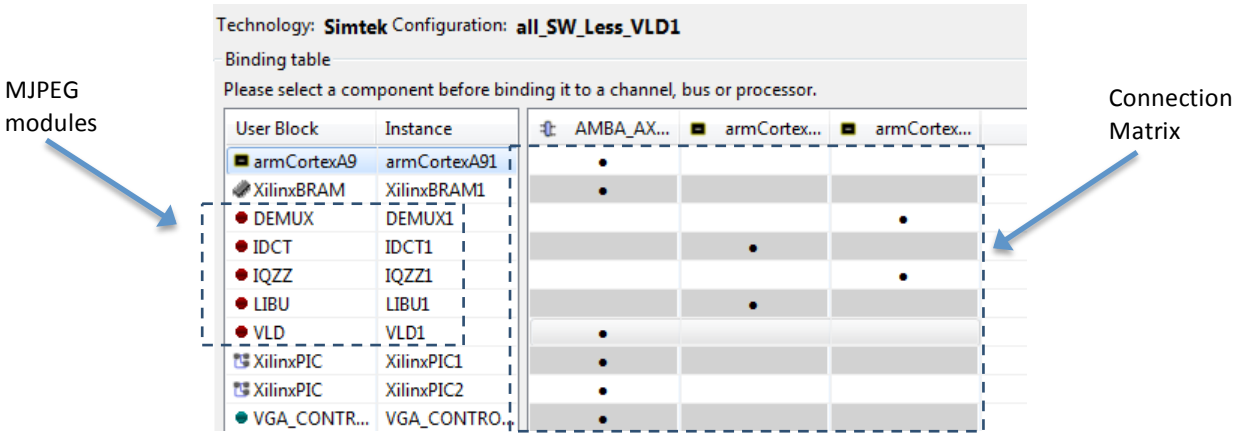

Figure 7 – SpaceStudio binding table used for mapping the application on the virtual hardware platform

3) Component inter-connexion

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

(ISS_adapter 1 & 2, VGA Controller, XilinxBRAM, etc.) connected to the AXI channel. This capability reduces 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 this step, design exploration is performed using some combination of architectural parameter modification, refinement of the high-level AADL model and adjustment of hardware/software partitioning. By keeping the same virtual platform to analyze and improve system performance, refinement cycles are accelerated compared to traditional RTL-based approaches (i.e. hours rather than days or weeks).

3.6 AADL model generation

Once the hardware architecture is optimized while respecting the initial specification, other evaluations may be desirable. An AADL hardware platform model of the solution can be generated from SpaceStudio in order to verify and analyze aspects not covered by this tool (e.g. reliability, safety, security, robustness, cost, etc.). This phase will be explored in future work on improvement of the design flow.

4. Experimental results

Figure 9 shows a candidate architecture that includes a single ARM Cortex-A9 dual core processor, AMBA AXI channel, BRAM memory block, VGA controller, plus two programmable interrupt controllers (one per processor) and ISS adapters. The ARM processor has a frequency of 800 MHz configured in the asymmetric multi-processing mode, running the µC OS II (RTOS).

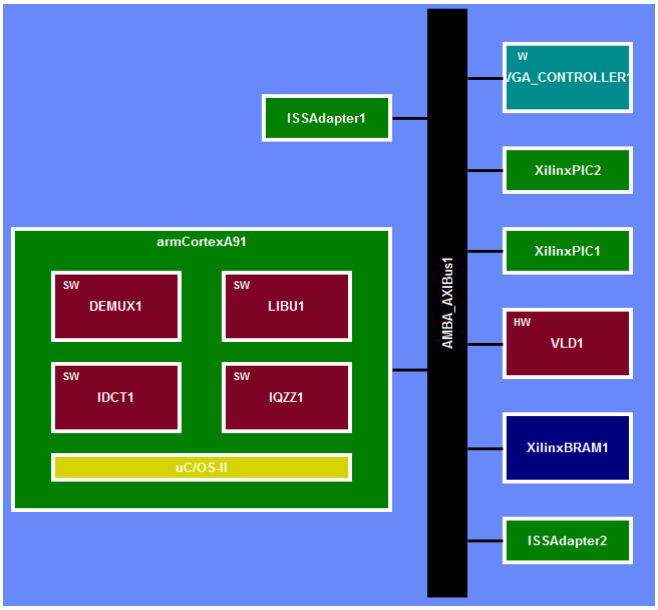

Figure 9 – SpaceStudio graphical interface representing the virtual platform

Once the virtual prototype of the system has been created, the execution of the application can be launched, with a data log capturing the simulation results for system performance evaluation purposes.

For functions targeted as software running on processors, the data logs can be used to generate pie charts that show CPU loading. Figure 10 shows simulation results for processor load distribution when all blocks of the application are mapped in software (on an ARM Cortex-A9 core).

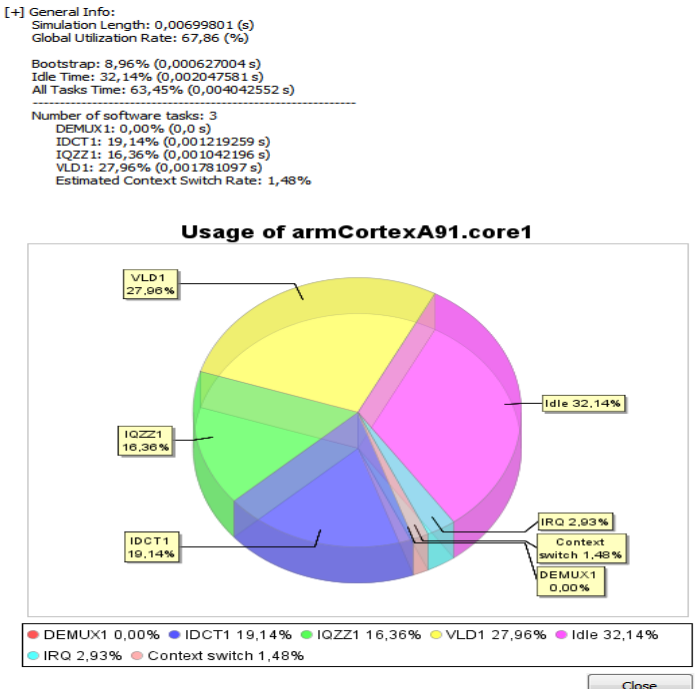

Figure 10 – Simulation results showing the distribution of processor load

System architects can proceed to an architecture refinement process that speeds up performance, for example by reallocating software tasks between processors, introducing additional processors, and/or retargeting tasks to hardware as co-processors on a bus/channel. Table 1 shows an example of five mapping architecture candidates that were explored in just a few minutes. As shown in Figure 10, VLD1 has the highest load (28 %) on the ARM Cortex-A9. To accelerate processing, VLD1 could be moved from software to hardware, as it appears in Figure 9.

Table 1 – Architectural exploration via the task retargeting process

Architecture candidate	Mapping on software	Mapping on hardware
1	All tasks	-
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 shows the increases in system performance achieved by retargeting functions from software to hardware, starting from an all-software mapping. The graphic presents the number of images per second decoded by the MJPEG for the five architecture candidates. We can see that the architecture candidate #5 can process 12 times more images per second than the all-software mapping (#1).

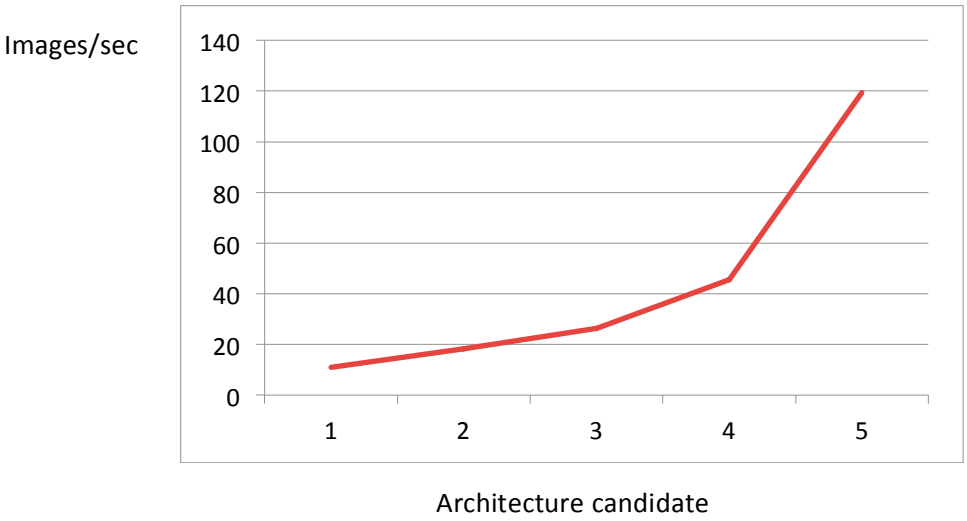

Figure 11 – Gain (12-fold) in processing speed for the MJPEG obtained using the mappings in Table 1

5. Conclusion

In this work, we have proposed a novel modeling framework for embedded systems design that considerably reduces design process complexity and development time, while providing means of increasing system performance. Our model-based engineering approach consists of developing an AADL high-level model of the system behaviour and then generating SystemC code to create a model that can be executed on a customisable virtual platform. We intend to improve the design flow in future work by exploring the following solutions: (1) including the communication interface in the transformation chain and ensuring its automatic extension from the AADL model to the virtual prototyping environment; (2) developing a new procedure to perform many non-functional analyses using an AADL model generated from a SpaceStudio, thus improving system robustness and enhancing product quality, and finally (3) using an avionics application as a case study to test and validate the resulting design flow.

6. Acknowledgment

This work was conducted as part of CRIAQ project AVIO509 and supported financially by CMC Electronics, CAE, CRSNG, MITACS, CMC Microsystems and the Consortium for Research and Innovation in Aerospace in Québec (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] Peter Wilson, H. Alan Mantooth, « Model-Based Engineering for Complex Electronic Systems », Newnes, 2013.

[3] 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.

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

[5] 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.

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

[7] J. Hugues, B. Zalila, L.Pautet, and F. Kordon., “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.

[8] 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.

[9] 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.

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

[11] 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.

[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, 2012.

[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.