

Simulink-Based MPSoC Design Flow: Case Study of Motion-JPEG and H.264

Kai Huang¹, Sang-il Han^{2,3}, Katalin Popovici³, Lisane Brisolará⁴, Xavier Guerin³,
Lei Li¹, Xiaolang Yan¹, Soo-Ik Chae², Luigi Carro⁴, Ahmed Amine Jerraya⁵

¹ Institute of VLSI Design, Zhejiang University, China, {huangk, lilei, yan}@vlsi.zju.edu.cn

² Seoul National University, Korea, {sihan, chae}@sdgroup.sni.ac.kr

³ TIMA Laboratory, Institute de Informatica, France, {katalin.popovici, xavier.guerin}@imag.fr

⁴ Federal University of Rio Grande do Sul, Brazil, {lisane, carro}@inf.ufrgs.br

⁵ CEA-LETI, MINATEC, France, ahmed.jerraya@cea.fr

ABSTRACT

System-level design methodologies have been introduced as a solution to handle the design complexity of embedded multiprocessor SoC (MPSoC) systems. In this paper we describe a system-level design flow starting from Simulink specification, focusing on concurrent hardware and software design and verification at four different abstraction levels: Simulink Combined Algorithm and Architecture Model (CAAM), Virtual Architecture, Transaction-accurate Model and Virtual Prototype. We used two multimedia applications, Motion-JPEG and H.264, to evaluate this design flow. Experimental results show that our design flow can generate various MPSoC architectures from Simulink CAAM correctly and efficiently, allowing processor and task design space exploration at different abstraction levels.

Categories and Subject Descriptors

C.3 [Special-purpose and application-based systems]: Real-time and embedded systems. I.6 [Simulation and Modeling]

General Terms: Design, Experimentation

Keywords: MPSoC design, automatic code generation, simulation, Simulink, Motion-JPEG, H.264

1. INTRODUCTION

With the increasing demand for complex embedded applications, heterogeneous multithreaded MPSoC architectures are becoming attractive solutions, because of their high computation power and flexible programmability [1]. However, as more processors and hardware components are integrated, designing and programming these complex multiprocessor architectures has become a major challenge. In conventional design approaches, hardware and software are usually considered separately, and their integration is done only when the hardware is fully defined. The lack of early coordination between hardware and software designs can cause unacceptable delay and cost overheads.

To address the MPSoC design complexity problem, hardware/

software co-design and co-verification should be performed at different abstraction levels [2]. SystemC [3] has been established as the preferable design language for this purpose. ROSES [4] and COWARE [5] tools are proposed to generate the software and hardware automatically from a SystemC transaction level model (TLM) of the MPSoC. However, SystemC TLM is still much oriented towards hardware designers, and does not allow system designers to specify complex systems at higher abstraction level.

Simulink [6] has been widely adopted as the prevailing environment for modeling and simulating complex systems at an algorithm-level of abstraction. However, mapping and refining algorithm models from a Simulink input to complete MPSoCs is still an open issue. Real-Time Workshop [7] and dSpace [8] can generate only software code for single- and multi-processor system from Simulink model, respectively. System Generator for DSP™ [9] is a high-level tool for designing multiprocessor systems with hardware logic targeted to FPGAs from a specific hardware/software partitioned Simulink model.

In this paper, we make use of a Simulink-SystemC MPSoC design flow that focuses on mixed hardware and software refinement from a high-level Simulink model [10]. This Simulink based MPSoC design flow allows concurrent hardware/software design and verification at four abstraction levels: Simulink CAAM, Virtual Architecture, Transaction-accurate model, and Virtual Prototype. These levels comprise a high-level specification down to detailed low-level implementation. This flow enables efficient hardware/software architecture refinement and early performance validation. In this paper, we evaluate this design flow through two case studies, Motion-JPEG decoder and H.264 baseline decoder, to show how our Simulink-based design flow can be used for hardware and software refinement, as well as system functional validation, performance analysis, and architecture exploration.

The rest of the paper is organized as follows: Section 2 presents the proposed Simulink-based MPSoC design flow. Section 3 describes the case studies for MJPEG decoder and H.264 decoder using the proposed design flow, and section 4 concludes the paper.

2. DESIGN FLOW

The Simulink-based multiprocessor SoC design flow (Figure 1) follows the same system-level design methodology presented in [11]. Starting with an application specification, the first step consists in producing the initial functional Simulink model. Step 2 transforms the application model to a CAAM model, also described in Simulink, which includes conceptual threads, CPU

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2007, June 4–8, 2007, San Diego, California, USA.

Copyright 2007 ACM 978-1-59593-627-1/07/0006 ...\$5.00.

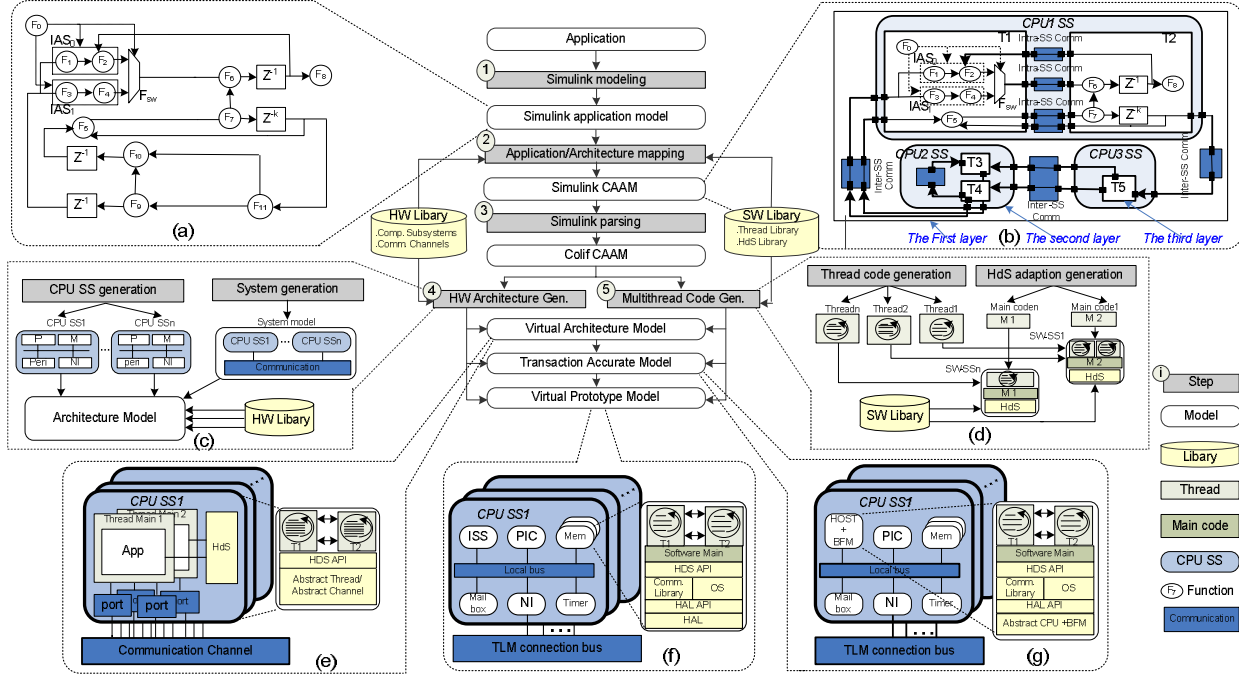


Figure 1. Simulink based MPSoC design flow

subsystems, and communication channels. Step 3 translates the CAAM into an XML based intermediate representation called COLIF [12]. This model is used by a *hardware architecture generator* (step 4) to produce multiprocessor hardware architecture models. The resulting architectures are composed of CPU subsystems, memory subsystem, and a communication network. The architecture may be produced at different abstraction levels (Virtual Architecture, Transaction-accurate Model and Virtual Prototype). On the other side, a *multithread code generator* (step 5) generates the corresponding software code for these generated architectures.

2.1 Hardware and Software Mixed Models

The mixed hardware software architecture concept allows modeling a multithreaded heterogeneous MPSoC at several abstraction levels, independent from the description language used by the designer. We will consider four abstraction levels: 1) CAAM, 2) Virtual Architecture, 3) Transaction-Accurate model, and 4) Virtual Prototype [2].

2.1.1 Combined Algorithm and Architecture Model

Simulink CAAM is a unified model that combines aspects related to the architecture model (i.e. processing units available in the chosen platform) into the algorithm model (i.e. multiple threads executed on the processing units). As shown in Figure 1(b), Simulink CAAM is a three-layered hierarchical structure using four specific Simulink subsystems: *CPU-SS*, *Inter-SS COMM*, *THREAD-SS*, and *Intra-SS COMM*. The first system layer describes system architecture that contains CPU subsystems (*CPU-SS*) and inter-subsystem communication channels (*Inter-SS COMM*). The second subsystem layer describes CPU subsystem that contains software threads (*THREAD-SS*) and intra-subsystem communication channels (*Intra-SS COMM*). The third thread layer describes software thread that contains Simulink blocks and links. The designer can easily transform a Simulink algorithm model to a Simulink CAAM by using the Simulink graphical user interface

and the proposed design flow automatically generates the required three different mixed hardware and software SystemC models from it. The designer partitions Simulink blocks into different subsystems according to feedback from simulations at different abstraction levels until the design requirements are satisfied.

2.1.2 Virtual Architecture

As shown in Figure 1(e), Virtual Architecture (VA) consists of abstract CPU subsystems communicating with each other via abstract communication channels. The abstract CPU subsystem provides software threads with Hardware dependent Software (HdS) APIs for thread and communication management. In this abstraction level, the detailed OS and hardware are not yet designed or selected. Thus, the VA enables one to develop and verify multithreaded software in an early stage.

2.1.3 Transaction-accurate Model

Transaction-accurate (TA) model as shown in Figure 1(g) consists of TA CPU subsystems and a TLM communication bus. Each subsystem is composed of an abstract CPU and hardware modules such as memories, buses, peripherals, etc. All modules in the hardware architecture are bus cycle-accurate. The abstract CPU provides the OS library with Hardware Abstraction Layer (HAL) API and Bus Functional Model (BFM) functions for context switching, interrupt handlings, communication I/O controls and memory access. The software stacks executing on the abstract CPUs are built by linking application threads and an OS library targeted to the abstract CPUs. The main purpose of TA is to design and verify hardware components and their device drivers. Since the software stack is directly executed on the simulation host, the mixed hardware-software simulation is accelerated.

2.1.4 Virtual Prototype

As shown in Figure 1(f), the processors in the Virtual Prototype (VP) platform are refined to Instruction Set Simulator (ISS) instead of CPU BFM in TA model. Each software stack consists of

application threads, an OS library, and a HAL and it can be executed on the target processor or the target ISS. The main purpose of VP is cycle-accurate simulation for fine-grain design space exploration and verification of software stacks, which can be executed directly on the real target platform.

2.2 Architecture Resources

We used two different processors, ARM7 and Xtensa. The ARM7 processor is used for system control, while the Xtensa reconfigurable processor is configured to accelerate the computing of some special algorithms. The currently used Xtensa ISS is generated with a 32-bit multiplier and a 16-bit MAC, without instruction and data caches, and without any instruction extension (TIE). Two types of communication mechanisms are used: Global FIFO (GFIFO) and Software FIFO (SWFIFO). The GFIFO allows for a CPU subsystem to transfer data to another CPU subsystem by using shared global memory and mailbox for data synchronization between different processors. The SWFIFO, which uses local memory as a buffer to transfer data from one thread to another, is used as a communication channel between two threads mapped on the same CPU subsystem.

The hardware library includes bus cycle-accurate CPU models embedding ARM and Xtensa ISSs, as well as bus cycle accurate hardware models for AMBA bus, SRAM bus, SRAM, programmable interrupt controller, timer, mailbox, and bus bridges. The software includes thread libraries, communication libraries, and device drivers targeted to both ARM7 and Xtensa processors.

2.3 Multithreaded Code generator

In the proposed design flow, processes for thread generation and software refinement, which are shown in Figure 1 (d), start from a Colif CAAM, and generate multithreaded C code as well as HdS code for the three lower abstraction levels. Each thread code includes local/global variables, behavior codes, and communication primitives corresponding to Simulink links, Simulink blocks, and communication blocks, respectively, as a result of static scheduling based on data dependency. Each main code consists of threads and intra-communication channels.

2.4 HW architecture generator

In generating hardware models for a system, as is depicted in Figure 1 (c), the *hardware architecture generator* first produces a set of codes for its subsystem architecture, which are refined from its Colif CAAM. Then, it also produces a system architecture code that instantiates all the CPU subsystems and the communication network among them. These two codes should be linked with the component codes from the HW library to make an executable code on the host computer.

3. CASE STUDIES

3.1 Motion-JPEG decoder

First, a Simulink functional model was built for a MJPEG decoder. This model is composed of 7 S-function (Simulink), 7 delays, 26 links, 4 IASs (If-Action Subsystems). Then, we partitioned this model into multiple threads in hardware and software to obtain the CAAM shown in Figure 2. In the subsystem CPU1, there are two threads: one for system-level control and the other for variable length decoding (VLD), which are connected by a SWFIFO. Inverse discrete cosine transform (IDCT) is divided into two parts: IDCTa and IDCTb, each of which is executed in

CPU2 and CPU3, respectively. All the three CPU subsystems are communicated through GFIFOs.

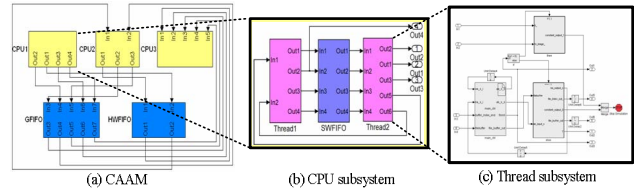


Figure 2. Simulink CAAM model for Motion-JPEG decoder

From the Simulink CAAMs, we generated the hardware and software mixed models at different abstraction levels (i.e. VA, TA, VP) and measured the simulation time and the total number of execution cycles for each architecture configuration in all the abstraction levels by simulating a 10-frame QVGA (320x240) JPEG stream.

3.1.1 Simulation time

Figure 3(a) shows the simulation time for the Motion-JPEG decoder models with three ARM cores at the different abstraction levels as well as RTW. The simulation time of RTW result is 0.16 seconds, which is the shortest achievable time, because RTW result is a sequential program on the host machine, while the simulation time for Simulink CAAM, VA, TA, and VP are 6, 1.8, 29, and 812 seconds, respectively. This result shows that the architecture models at different abstraction levels provide trade-off alternatives between simulation time and architecture detail.

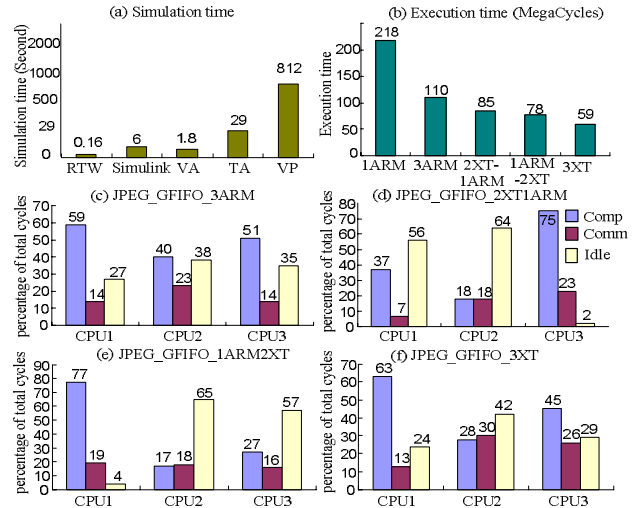


Figure 3. Experimental results for Motion-JPEG decoder

3.1.2 Execution time

At the VP level, we measured the total numbers of execution cycles for five different architectures as shown in Figure 3(b), where x ARM- y XT represents a platform with x ARM7 processor(s) and y Xtensa processor(s). The platform with three Xtensa processors shows 3.7 speeding up than that with a single ARM7 processor. With the proposed design flow, designers can evaluate several platforms by simple manipulation of the input Simulink algorithm model, within a couple of hours. To find bottlenecks and optimize the generated platforms, we obtained cycle-accurate information for four different MPSoC platforms, as illustrated in Figure 3(c-f), where *comp*, *comm*, and *idle* stand for

computation, communication, and idle cycles, respectively. For example, in Figure 3(f), the computation, communication, and idle cycle of CPU1 are 63%, 13%, and 24%, respectively, while those of CPU2 are 28%, 30%, and 42%, respectively. From this result, one can identify that it is necessary to move computation blocks from the CPU1 subsystem to the CPU2, in order to improve performance. Moreover, using these results, designers can optimize task allocation to the processors so that each processor has enough margins for future addition of functions. Note that the design space exploration based on cycle-accurate simulation at an implementation-level is a complement to automatic design space exploration at a high abstraction level, which is our future work.

Table 1. Task assignment for three architectures

2ARM	4ARM	6ARM	Function Block
CPU1	CPU1	CPU1	Global control and MB VLD
CPU2	CPU2	CPU2	Chroma decoding
	CPU4	CPU3	Luma VLD
	CPU3	CPU4	Luma DF
	CPU4	CPU5	Luma IQ/IT
		CPU6	Luma MC/SC and Luma Rec

3.2 H.264 Baseline Decoder

In this case study, we analyzed the effect of different task partitioning of an H.264 decoder using the proposed MPSoC design flow. We built a Simulink model for the H.264 baseline decoder, which includes 83 S-functions, 24 delays, 286 data links, 43 if-action-subsystems, 5 for-iteration subsystems and 101 pre-defined Simulink blocks. The whole decoding process is basically an iteration of a 16x16 MacroBlock (MB) decoding process. The MB decoding process consists of six function blocks: (1) global control and MB VLD, (2) Luma VLD, (3) Luma Deblock Filter (DF), (4) Luma Inverse Zigzag and Inverse Quantization (IQ), (5) Luma Motion Compensation (MC) and Spatial Compensation (SC), and (6) Chroma decoding and each function block consists of one or more S-functions. In this experiment, we built three platforms with two, four, and six ARM processors and GFIFOs for inter-subsystem communications. To measure the execution cycles, we used FOREMAN QCIF sequence as the input stream.

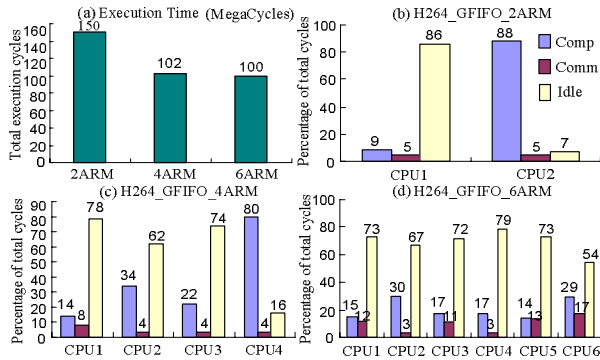


Figure 4. Performance of different task partition strategies

We implemented different task partitions, as shown in Table 1, from coarse to fine granularities for three architectures in this experiment, and obtained the results shown in Figure 4. According to simulation results for the platform with two ARM processors

illustrated in Figure 4(b), we identified that the computation load of CPU2 is heavy, and hence partitioned its tasks in a finer granularity, separating these tasks into three CPU subsystems. The new configuration is then evaluated and as shown in Figure 4(c), the computation cycles of CPU2, CPU3 and CPU4 are 34%, 22% and 80%, respectively. To better distribute the computation load, we partitioned the computation blocks of CPU4 into three CPU subsystems in order, building the 6ARM architecture.

From the results of the above three different task partitioning, one can find that the architecture with four ARM processors is the best tradeoff between performance and cost. However, there are more possibilities to improve the performance of the H.264 decoder: for example, replacing CPU4 with a dedicated processor with higher performance.

4. CONCLUSION

We proposed a Simulink based MPSoC design flow for concurrent hardware-software design and verification. In this design flow, a Simulink CAAM is gradually refined into hardware and software using three different abstraction level models, the Virtual Architecture Model, the Transaction-accurate Model and the Virtual Prototype. In the case of the Motion-JPEG decoder, we provided trade-off alternatives between simulation time and architecture details while exploring the design space with the three abstraction levels for various MPSoC architectures. In the case of the H.264 baseline decoder, we estimated performance of the targeted architecture with cycle-accurate VP level simulation, determining the architecture with better performance with task partitioning refinement. As a future work, we plan to improve the performance of the generated architectures by adopting processors with dedicated instruction sets, and to integrate the design flow with automatic design space exploration flow.

5. REFERENCES

- [1] A. A. Jerraya, et al, Guest Editors. IEEE Computer, Special Issue on MPSoC. v. 38, n. 7, pp. 36-40, July 2005.
- [2] A.A. Jerraya, W. Wolf, "Hardware/Software Interface Codesign for Embedded Systems", Computer, Volume 38, Number 2, pp. 63-69, February 2005.
- [3] Open SystemC Initiative. <http://www.systemc.org/>.
- [4] W. Cesario et al., "Multiprocessor SoC Platforms: A Component-Based Design Approach", IEEE Design & Test of Computers, v. 19, n. 6, Nov-Dec, 2002.
- [5] Coware, <http://www.coware.com/>.
- [6] Simulink, Mathworks. <http://www.mathworks.com>
- [7] Real-Time Workshop. <http://www.mathworks.com>.
- [8] RTI-MP, <http://www.dspaceinc.com/ww/en/inc/home/products/sw/impsw/rtimpblo.cfm>.
- [9] Xilinx, System Generator, <http://www.xilinx.com/>.
- [10] K. Popovici et al. Efficient Software development platforms for Multimedia Applications at Different Abstraction Levels", RSP Workshop, 2007.
- [11] K. Keutzer, et al., "System-level design: Orthogonalization of concerns and platform-based design". IEEE Trans. On CAD of Integrated Circuits and Systems. v.19, n.12, 2000.
- [12] W. Cesario, et al., "Colif: A design Representation for Application-Specific Multiprocessor SoC", IEEE Design and Test of Computers, v.18, n.5, pp. 18-20, 2001.