Multi-Processor SoC Design Methodology using a Concept of Two-Layer Hardware-dependent Software

Sungjoo Yoo, Mohamed-Wassim Youssef, Aimen Bouchhima, Ahmed A. Jerraya
TIMA Laboratory, Grenoble France

Mario Diaz-Nava
ST Microelectronics, Grenoble France

Abstract
In conventional multiprocessor SoC (MPSoC) design methods, we find two problems: lack of SW code portability and lack of early SW validation. The problems cause a long design cycle. To resolve them, we present a concept of two-layer hardware-dependent software (HdS). The presented HdS consists of hardware abstraction layer to abstract the sub-system architecture and SoC abstraction layer to abstract the global MPSoC architecture. During the exploration of global and sub-system architectures, the application programming interfaces of presented two-layer HdS allow to keep the SW independent from architectural change. The simulation models of two-layer HdS enable to validate the entire system including the SW and HW design early in the design steps. We show the effectiveness of the presented methodology in the MPSoC architecture exploration of an OpenDiVX encoder system design.

1. Introduction
Multi-processor SoCs (MPSoCs) are generally built as a set of sub-systems interconnected through a communication network called network-on-chip. MPSoCs may include sub-systems executing application SW. Conceptually, the SW can be decomposed into three parts: one part is application SW running on the sub-system, another part is the SW (conventionally called *hardware abstraction layer, HAL*) that depends on the local architecture of sub-system, and the other is the SW (which we call *SoC abstraction layer, SAL*) that depends on the global architecture of MPSoC.

Current practices in MPSoC design hardly separate these three parts. SW is generally designed as a single program where all the three layers are mixed. This induces at least the following two problems: lack of SW portability and lack of early SW validation.

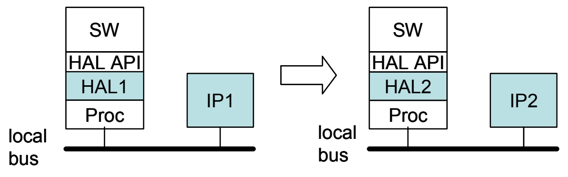
Lack of SW portability: The SW is not portable over different MPSoC architectures. This incurs two inconveniencies. One is the SW is not reusable over different SoC designs. The other is that HW architecture exploration is difficult since when we change a HW architecture, we need also to change the SW which will run on the new HW architecture.

Lack of early SW validation: As the SW and HW design continues, the designer needs to be able to validate the current SW design before the HW design is not yet completely finished. In conventional practices, since the SW is a single program, we cannot validate a part of SW independently from the other part of SW. Thus, the validation of SW is done late in the design steps after both SW and HW design is finished, using cycle-accurate HW/SW cosimulation or a prototyping system.

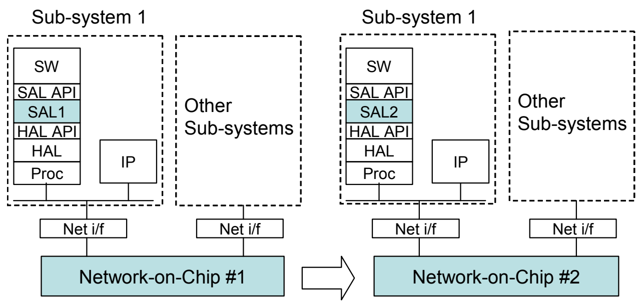
To overcome the above problems, we propose the usage of two hardware dependent software (HdS) application programming interfaces (APIs): HAL and SAL APIs. The APIs allow SW portability by dividing the SW into three parts (application SW, sub-system architecture dependent one, and global architecture dependent one) that we mentioned in the beginning of this section. To resolve the problems of early SW (and HW) validation, we present the simulation models of the two HdS APIs.

2. Related Work and Our Aim
From the HW abstraction viewpoint, there are a few standardization activities for hardware abstraction. Universal Device Interface (UDI) is a standard of HW access in device driver [1]. VSIA is developing HdS-API [2]. Recently, the industry starts to develop HdS APIs such as Mobile Industry Processor Interface (MIPI) API standard [3]. As parallel programming model, there are two types of programming model for multiprocessor systems: message passing, e.g. MPI [4] and shared memory, e.g. OpenMP [5].

In our work, our aim is to efficiently utilise the above concepts of existing APIs of HW abstraction and parallel programming model to resolve the above two problems in MPSoC design.



(a) SW portability by HAL API in sub-system architecture exploration



(b) SW portability by SAL API in global architecture exploration

Figure 1 SW portability in architecture exploration

3. SW Portability in Architecture Exploration

Figure 1 (a) shows a simple sub-system architecture consisting of a processor, processor local bus, and a peripheral IP. HAL API gives an abstraction of sub-system architecture to upper SW layer (application SW + SAL). Thus, when exploring different sub-system architectures, e.g. changing the processor type, peripheral IPs, etc., HAL API gives SW code portability. In the figure, the peripheral IP is assumed to be changed from IP1 to IP2. In the architecture change, since HAL API does not change, upper SW layer does not change, either. According to the architecture change, HAL needs to be changed (in the figure, from HAL1 to HAL2).

Figure 2 (b) shows the case of global architecture change. SAL API gives an abstraction of global architecture to upper SW layer (application SW). In this case, the network is changed from NoC #1 (e.g. connection-less packet switch network) to NoC #2 (e.g. connection-oriented circuit switch network). Since the SAL API remains unchanged, application SW code does not change in the architecture change.

4. Simulation Models of HdS APIs

Figure 2 shows the concept of simulation model of HdS API. The HdS simulation model emulates a HdS API (HAL or SAL API) on a host simulation environment, e.g. SystemC [6] or Unix. For instance, if the application SW is written using HAL API, the simulation model emulates the HAL API, e.g. emulating context switch on Unix. the application SW can be executed natively on a simulation host without using simulators such as instruction set simulator. Such a native execution enables a very fast simulation.

To evaluate the performance of application SW on the HW architecture, timed simulation can be performed for the application SW and the HW. For the timed simulation, the application SW code is annotated with its execution delay (measured or estimated by the designer) [7]. The simulation model of HdS API synchronizes both timed SW and HW simulation at RTL or TLM [8].

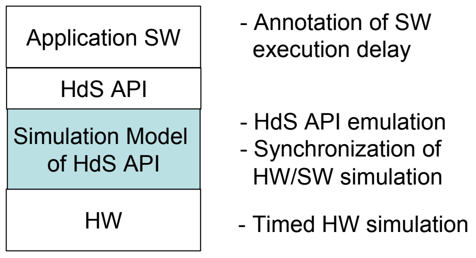

Application SW
HdS API
Simulation Model of HdS API
HW
- Annotation of SW execution delay
- HdS API emulation
- Synchronization of HW/SW simulation
- Timed HW simulation

Figure 2 Simulation model of HdS API

5. Experiments

To prove our concept of two-layer HdS and the effectiveness of simulation model of HdS API, we designed an OpenDiVX [9] encoder system on a multiprocessor prototyping platform [10]. We parallelised the initial sequential code of OpenDiVX into four concurrent tasks: one master and three slave tasks. Tasks communicate with each other via SAL API, i.e. Message Passing Interface API (MPI_Send/Recv). HAL API includes function calls such as context switch, enable/disable_interrupt, read/write_bus, etc.

For global architecture exploration, as shown in Figure 3, we explored two different global architectures changing the number of ARM processors on the prototyping platform. Since SAL API gives an abstraction of underlying global architectures, we use the same application SW code of master and slave tasks in the global architecture exploration. The code size of SAL and HAL is small (5.7—6.3KB) compared with the application code size (254KB and 461KB for master and slave tasks, respectively).

For sub-system architecture exploration, we replace an ARM processor (in Case 1 of Figure 3) with a ST220 VLIW processor [11]. In this case, we need to design a new HAL (15KB) for the new ST220 processor. However, since only the sub-system architecture changes, the upper SW layer, i.e. OpenDiVX code written in SAL API and SAL code, does not change.

The simulation model of HdS API allowed to simulate application SW at SAL and HAL API levels. Compared with the speed and accuracy of execution of the OpenDivX on the prototyping platform (all processors at 25MHz), the high-level simulation of application SW yields 3.5~17 times higher speed and up to 86% accuracy.

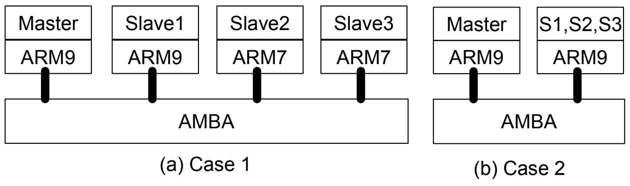


Figure 3 Two global architectures

6. Summary

In this paper, we presented a concept of two-layer hardware-dependent software: SAL and HAL. The concept allows to keep SW code unchanged during the architecture exploration. The simulation models of HdS APIs enable the entire system validation at SAL and HAL API levels, early in the design flow. The accuracy and runtime of high-level simulation show that the presented simulation model enables early SW validation in a fast and accurate way.

Acknowledgement

This work was supported by ToolIP, Medea+ project and ArchiFlex, French government funded project.

References

[1] Uniform driver interface, www.projectudi.org
[2] HdS Development Working Group, www.vsia.org
[3] Mobile Industry Processor Interface, www.mipi.org
[4] Message passing interface, www-unix.mcs.anl.gov/mpi
[5] OpenMP, www.openmp.org
[6] SystemC, www.systemc.org
[7] M. Lajolo, *.et .al*, “A Compilation-based Software Estimation Scheme for Hardware/Software Co-simulation”, Proc. CODES, 1999.
[8] SystemC based SoC Communication Modelling for the OCP Protocol, www.ocpip.org/socket/systemc
[9] Project Mayo, www.projectmayo.com/index.php
[10] ARM Integrator, ARM Ltd.
[11] ST220, ST Microelectronics.