CNRS INPG UJF

ISSN 1292-862

Shortening SoC Design Time with New Prototyping Flow on Reconfigurable Platform

ROUSSEAU, F.*
SASONGKO, A.* & JERRAYA, A.A.*

* TIMA Laboratory, 46 avenue Félix Viallet 38000 Grenoble France

ISRN TIMA--RR-05/05-01--FR

To be published in NEWCAS 05

Shortening SoC Design Time with New Prototyping Flow on Reconfigurable Platform

Frédéric ROUSSEAU, Arif SASONGKO, Ahmed A. JERRAYA
TIMA Lab, 46 av. Félix Viallet, 38031 GRENOBLE Cedex - FRANCE
{Frederic.Rousseau, Arif.Sasongko, Ahmed.Jerraya}@imag.fr

Abstract- **A modern SoC contains complex hardware components and a huge amount of software. These software parts, including the operating system, become so complex that their validation and debug is not feasible anymore by classical ISS based simulation approaches.**

Hardware prototypes are built to allow early software development and hardware-software integration. However the development of an application specific prototype takes a lot of time and effort. In this paper, we present an efficient approach for complex SoC prototyping on reconfigurable prototyping platform. This method provides all advantages of hardware prototyping solutions, but it avoids the time and cost required to build an application specific prototype.

I. INTRODUCTION

A modern SoC contains a great amount of embedded software and a large number of heterogeneous components: processors, DSPs, memories, sophisticated communication networks, and a set of hardware blocks (IPs). Software development and hardware-software integration are becoming one of the most critical paths in current System-on-Chip (SoC) development flows. EDC [1] survey shows that embedded system developers spend more than half of their time for the software development. The only solution to shorten SoC design cycle is to start the software development earlier and to allow concurrent design of hardware and software.

Building a hardware prototype allows SoC developers (1) to start software development earlier, (2) to start hardware-software integration before chip fabrication, and (3) to reduce the possibility of finding hardware bug later after fabrication.

Figure 1.a shows a classical development cycle for a typical SoC. In this scheme, hardware and software are developed sequentially and the integration of hardware and software, and debug of hardware/software interface can start only when the hardware is ready. In Figure 1.b, hardware prototyping allows shortening the design cycle. When the hardware specification is ready, a prototype can be built and used to develop software and debug hardware-software interfaces. Despite of the reduction of the design cycle, the development of the prototype still requires few months. This application-specific prototype is expensive in terms of man-power and time-to-prototype. Therefore, we believe that a highly reconfigurable prototyping platform is the solution, as it reduces the time required to build the application specific prototype as shown in Figure 1.c. This prototyping can be done in few weeks because the time to design and to fabricate the physical hardware board is removed.

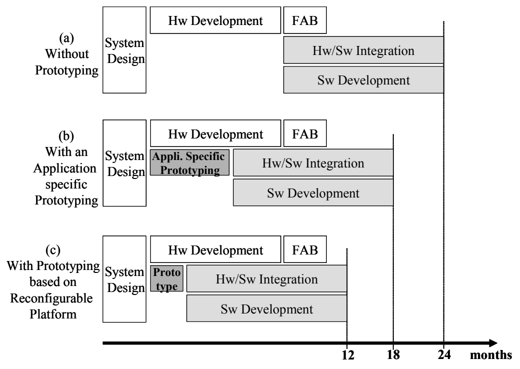


Figure 1: Time saving with prototyping

Unfortunately, in practice, one notices that the architecture of SoC and the architecture of the prototyping platform do not match. In that case, building the prototype may require reorganization of the platform and eventually adaptation of the soc architecture.

In this paper, we present a new approach for mapping a SoC architecture on reconfigurable platform. This approach is partially automated and allows building a SoC prototype in short time.

The rest of the paper is organized as follows. Section 2 introduces generic models of SoC and reconfigurable platform. Section 3 describes the prototyping flow on reconfigurable platform, and explains the systematic prototyping process. Section 4 illustrates the efficiency of the proposed approach through the prototyping of an OpenDivX application. Finally, section 5 gives conclusions and presents some perspectives for this work.

II. MODELS FOR PROTOTYPING

A. Hardware/software architecture model

A generic architecture model for SoC is given in Figure 2.a as presented in [2]. This architecture model divides the application into three parts: software computation node, hardware computation node and communication network. Three layers compose the software part: application software, operating system (OS), and hardware abstraction layer (HAL). The hardware part consists of four layers which are: processors, communication wrappers (Intellectual Properties "IP" wrappers and processor wrappers), communication networks, and IP.

Application software is a set of software tasks that performs the behavior described in the specification. OS and HAL manage execution of tasks and utilization of resources (including hardware) by the tasks. Separation of HAL from the OS enables the OS and application software to be ported easily to different processors. HAL abstracts the hardware and some parts must be written in assembler. Standard processors normally execute these software parts. Communication wrappers (processor and IP wrappers) are used as access points to the communication network. These wrappers implement translation between processor buses and the communication network. They are described at RT level when the prototyping flow starts. The communication network is a set of connections and protocols to perform communication between software computation nodes and hardware computation nodes. It can be crossbar, buses, point-to-point connections or combination of them.

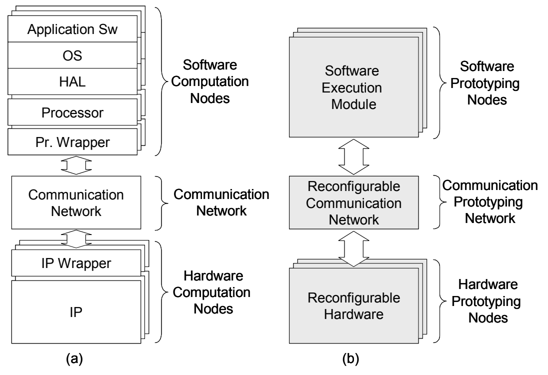

Application Sw
OS
HAL
Processor
Pr. Wrapper
Software Computation Nodes
Communication Network
Communication Network
IP Wrapper
IP
Hardware Computation Nodes
(a)
Software Execution Module
Software Prototyping Nodes
Reconfigurable Communication Network
Communication Prototyping Network
Reconfigurable Hardware
Hardware Prototyping Nodes
(b)

Figure 2: (a) generic SoC architecture model (b) generic reconfigurable platform model

B. Generic platform model for prototyping

In a generic prototyping platform for SoC, (Figure 2.b), we distinguish three parts: software prototyping nodes, hardware prototyping nodes, and a communication prototyping network.

A software prototyping node consists of processor, memory, and reconfigurable device (i.e. FPGA). The reconfigurable device adapts the processor to the communication network.

A hardware prototyping node is made of configurable device. Usually, this configurable device is used to implement a specific hardware block (IP), and also to adapt this IP to the communication network. The hardware prototyping node may also include memory and analog reconfigurable device [3].

The communication prototyping network is a module which is able to implement various types of communication networks. This module consists of a configurable device and it implements protocols and routing between other nodes of the platform.

III. PROTOTYPING ON RECONFIGURABLE PLATFORMS

A. Assignment and targeting

Ideally, all the prototyping nodes of the platform are configurable without any limitation (memory map, interrupt mechanism, etc). In this case, prototyping is very simple with only two sequential steps: assignment and targeting.

In the assignment step, it is required to make association between parts of the SoC architecture and prototyping nodes of the reconfigurable platform. This is not a hardware/software partitioning step because the application is already described in RT level. We assume here that assignment is done manually.

Targeting is done using standard processes such as compilation, linking, logic synthesis, placement and routing to build the prototype.

Most existing reconfigurable platforms come with a set of tools to help the designer to prototype the application on the platform [4][5][6][7][8]. However, in most cases, prototyping is restricted to targeting.

B. Prototyping flow

Most of the time, the reconfigurable platform does not match the SoC architecture. So, we propose a prototyping flow to deal with this case. Indeed, building a prototype may require reorganization of the platform and eventually adaptation of the SoC architecture. This approach is made of four kinds of actions (Figure 3): (1) assign each part of the application to the platform, (2) configure the platform to match as much as possible the SoC architecture, (3) adapt (if still necessary) the SoC architecture, and (4) target the adapted architecture to the configured platform. Assignment and targeting have already been explained, so we only detail configuration and adaptation.

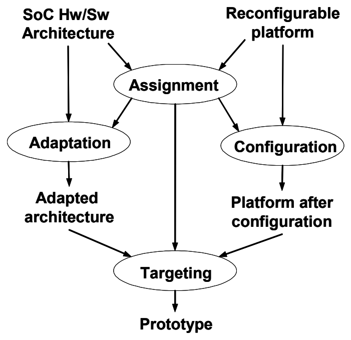


Figure 3: Prototyping flow on a reconfigurable platform

C. Configuration

Configuration is required when components of the platform do not directly match all modules of the SoC architecture. In this case, targeting cannot be done directly and reorganization of the platform is required. A typical example is when communication network of the platform cannot handle the communication protocols of the SoC architecture. Thus, the platform is reconfigured in order to map the communication protocols of the SoC architecture on a hardware node rather than on the communication prototyping network. This hardware node is usually connected to the communication prototyping node, so a translator block is required and must be inserted between them. This translator is a form of configuration to match the architecture of the platform to the

SoC architecture. Refer to [9] for a detailed example of configuration.

D. *Adaptation*

Adaptation is required when SoC architecture does not fit the configured platform, usually because some platform parameters are fixed. Adaptation consists in altering the application to match the property of the reconfigurable platform. For example when the platform has a fixed memory map, the HAL or OS has to be adapted to the memory map of the platform.

The software part of a SoC is composed of several layers, but only HAL is hardware dependent. This means that we should only write (or generate) a new HAL that interfaces the configured prototyping platform (the hardware) and the upper software layer.

We use our HAL generator tool to automate the adaptation step. An abstract SoC architecture is used as the input to the HAL generator. This tool is already used to generate the HAL for the SoC [2][9]. This generator selects the necessary functions from a library to produce fully executable software. Several other software interface generation tools are explained in the literature and can be used for this purpose [10].

IV. APPLICATION EXAMPLE: DIVX

A. *DivX encoder*

DivX [11] is a technology for compressing digital video without reducing visual quality. It is based on MPEG-4 compression standard. It can compress MPEG-2 Video (currently used in DVD) up to 10 % of its original size. Unfortunately it requires so complex computation that a real time implementation of the encoding algorithm becomes difficult.

A multiprocessor architecture allows accelerating the execution time. To that end, the application code is partitioned to be executed on several processors and an OS is generated for each processor.

In this section, we focus on the prototyping process of the DivX. The objectives are (1) to validate the partitioned application software, (2) to verify functionality of OS, and (3) to validate data exchanges.

This prototype is not intended to have the same performance of the final SoC. In fact, the clock frequency of the processor in the prototyping platform is very low (20 MHz), and the communication network of this platform is fixed to single layer AMBA bus with limited bandwidth.

B. *Hardware/software architecture*

DivX application is partitioned into two modules with master-slave configuration in our experiment. The master performs the top level encoding process. The slave performs calculation of movement vector and movement compensation. The software part consists of application task, OS, and HAL. The hardware part consists of two processors, connected by a point to point communication. The communication protocol is based on a hardware FIFO.

C. *The prototyping platform*

In this work, we use the Arm Integrator Platform. This platform is designed to develop hardware and software for SoC with AMBA bus [12] and ARM processors (see Figure 4). This platform fits in standard ATX size PC chassis using standard mounting holes and PC power connectors.

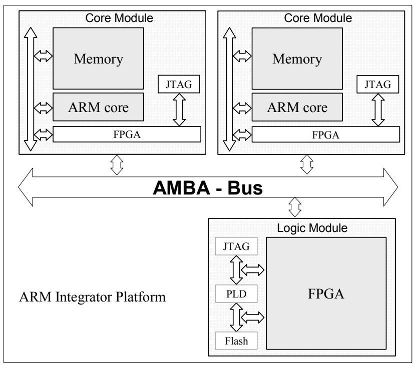


Figure 4: Arm Integrator Platform

Hardware prototyping node in this platform is called logic modules [13] and made of FPGA. The physical connection to the main board is also adapted to AMBA bus. Beside FPGA, this module has PLD and flash memory to manage and store configuration of the FPGA, adjustable clock and reset generator, and peripherals (LED, DIP switch, JTAG, …).

Software prototyping modules are called core modules. These modules can also be used stand alone. Several ARM processors are available and we use ARM7TDMI, and ARM966E-S [14]. Along with the processor, in each module, there are some memories (256 Kbytes to 1 Mbytes SSRAM and 128 Mbytes SDRAM), FPGA and some interfaces. The FPGA implements the AMBA interface, reset controller, and memory controller.

D. *Assignment*

In the assignment step, we decide to realize as much function as possible in software. This decision was driven by the constraint in the ARM integrator platform where the communication is fixed through the AMBA bus. And the FPGA between processor and AMBA bus can not easily be changed as it has also some other functions.

So we use in the platform the same two processors (ARM7 & ARM9) as ones in the SoC architecture. The AMBA bus should be hidden, and drivers are supposed to be changed to implement a point to point communication. In fact, a great part of the communication needs some modifications. Indeed, inserting FIFO is impossible in the platform, so we will implement this FIFO by software, distributed in shared memory part of both processors.

Figure 5 illustrates the assignment process. Each part of the specification is associated to a part of the platform. The result is an assignment table which guides all the next steps: adaptation, configuration, and targeting.

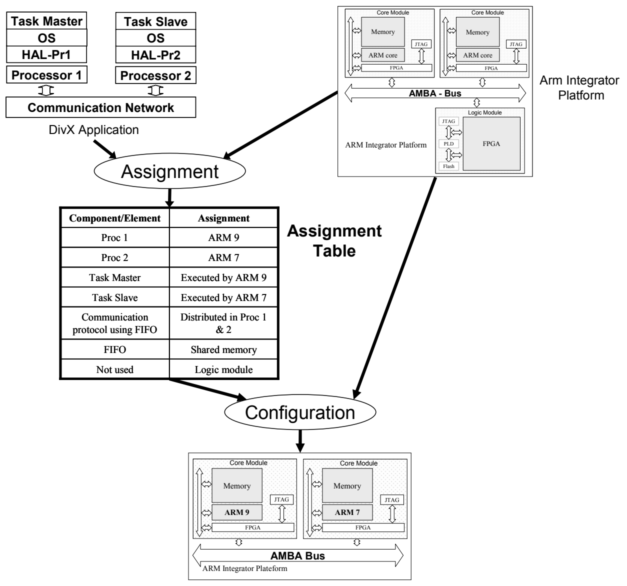
E. *Configuration*

We configure the platform to use one ARM 9 and one ARM 7 as decided during the assignment (Figure 5). In this experiment, the configuration process is very simple.

F. *Adaptation*

This step concerns three different aspects in this experiment. At first, the memory mapping imposed by the platform is quite different from the one of the application. This means that in all parts of the low level code of the HAL (in C or assembly language), we need to modify them to respect the memory mapping of the platform. Hopefully, with the HAL generator, a part of all these transformations can be done automatically.

Then, a part of the code to access the network (SWI routine, boot code, debug interface) should be modified to respect constraints of the platform.



Component/Element	Assignment
Proc 1	ARM 9
Proc 2	ARM 7
Task Master	Executed by ARM 9
Task Slave	Executed by ARM 7
Communication protocol using FIFO	Distributed in Proc 1 & 2
FIFO	Shared memory
Not used	Logic module

Figure 5: Assignment and configuration for DivX application

And finally, the FIFO controller, done in hardware in the SoC, is distributed in software in the platform, which means that when a processor writes or reads data in the FIFO, it has to send an interrupt to the other processor. It has also to write what kind of operation it has done, which requires modification of the interrupt routine, and a new memory space to keep this control information.

G. *Targeting*

HAL, OS, and application tasks are compiled with armcc, armcpp and armasm. Finally all of these codes are linked to build executable images for each processor and downloaded onto the platform.

H. *Results, analysis, and difficulties*

The experiment shows that the prototype can be delivered in few weeks. This prototype gives us the methods to achieve our objectives. We manage to validate the parallelized application codes and many bugs were found and fixed. The prototype is able to treat about 100 QCIF images per hour. This allows debugging using a long video sequence and finds bugs that are related to data dependant code.

The OS is validated to run on multiprocessor configuration. Most problems were due to synchronization and resource sharing between OS. This kind of problems is not easy to find with simulation because it takes a long execution time to be detected and the time required would have been too long.

We achieved three objectives. The platform helps developing and debugging the application software before the hardware is ready. We estimate that hardware/software integration is done in large extent: application code and OS are validated. The code developed on the platform is easy to adapt on the target system with very few minor modifications (HAL and drivers). The platform allows a better understanding of the target hardware system, mainly about multiprocessor synchronization and boot.

We found also other problems. For instance, the debugger does not support multiprocessor debugging process, so a breakpoint on code of a processor can not stop or trigger execution of other processors, which made the debugging harder.

V. CONCLUSIONS AND PERSPECTIVES

In this paper, we presented a novel approach for complex SoC prototyping. This approach is based on a highly reconfigurable platform and a partially automatic prototyping flow. It enables to achieve prototype of complex SoCs with less time and effort. The use of highly reconfigurable platform helps designers to cover a large application field and thus to cope with cost issues. This prototyping flow shortens significantly the SoC design cycle through its ability to act as software development environment.

REFERENCES

[1] Embedded Systems Development Survey Volume 1, 2003, http://www.evansdata.com/n2/index.shtml, Evans Data Corporation.

[2] W. Cesário et al., "HW/SW Interfaces Design of a VDSL Modem using Automatic Refinement of a Virtual Architecture Specification into a Multiprocessor SoC: a Case Study", DATE 2002, Paris, March, 2002.

[3] Adrian Brat, Ian Macbeth, "Design and Implementation of a Field Programmable Analog Array", International Symposium on FPGA, Monterey, USA, 1996.

[4] ARM Ltd, http://www.arm.com/armtech/primeXsys.

[5] M. Dorfel, R. Hofmann, "A Prototyping System for High Performance Communication System",12th RSP, Leuven, Belgium, June, 2001.

[6] E. Mosanya, et al., "Platform for Codesign and cosynthesis based on FPGA", 7th RSP, Thessaloniki, Greece, June, 1996.

[7] WinSystems, Inc., http://www.winsystems.com.

[8] Aptix Corporation, http://www.aptix.com/.

[9] A. Sasongko and all, "Prototyping of Embedded Applications to Configurable Platform Driven by Communication Constraints", ACM Transactions On DAES, Vol. 8, Kluwer, 2003.

[10] Coware, Inc., http://www.coware.com/coware/N2C. html.

[11] OpenDivX, www.projectmayo.com/projects/index.php

[12] ARM Ltd, AMBA Specification (Rev 2.0). ARM Limited, 1999.

[13] ARM Ltd, Integrator/LM-XCV600E+ Integrator/LM-EP20K600E+ User Guide. ARM Limited, 1999.

[14] ARM Ltd, Integrator/AP User Guide. ARM Limited, 1999.