IP Core Design
Lab Overview

Kun-Bin Lee

NCTU
2001.9~ 2002.1

Abstraction

With the advance of IC manufacturing process, the new generation of design methodology, intellectual properties (IPs) together with platform-based design, is considered as one of the best approaches to conquer both the raising complexity of the design and the pressing schedule of time-to-market during developing System-on-a-Chip (SoC). Because the specific application of a platform is definite, the characteristics of IPs become well defined and these IPs can be, therefore, reused without redesigned. On the other hand, both software and hardware architecture can also be reused as the categorized applications have the same characteristics.

In this course, we are going to develop the IP authoring flow. The authoring flow includes *high-level modeling for design and verification*, *hardware/software coordination in embedded system*, *authoring for hardware IP and its associated software components - the driver*. Though our IP authoring flow is target at ARM-based SoC design platform, which dominates 70% market in embedded system, we believe the flow can be applied to other processor-based SoC design environment.

Introduction

It is being widely recognized that the reuse of IPs, hard, firm or soft cores, is becoming fundamental to closing the design gap for a successful SoC design; i.e., it conquers the design complexity and meets the time-to-market requirement. The compelling feature of IPs is that they are correct-by-construction, allowing the re-deployment of engineering resources to other critical aspects of the design. As shown in Figure 1, A SoC contains system-level design, hardware design, software design and hardware/software coverificaiton. To make the system integration more quickly, IP authoring should also fit to the SoC design flow. In traditional ASIC design flow shown in Figure 2, hardware and software development are mostly serialized and the software portion of the IP is less emphasized.

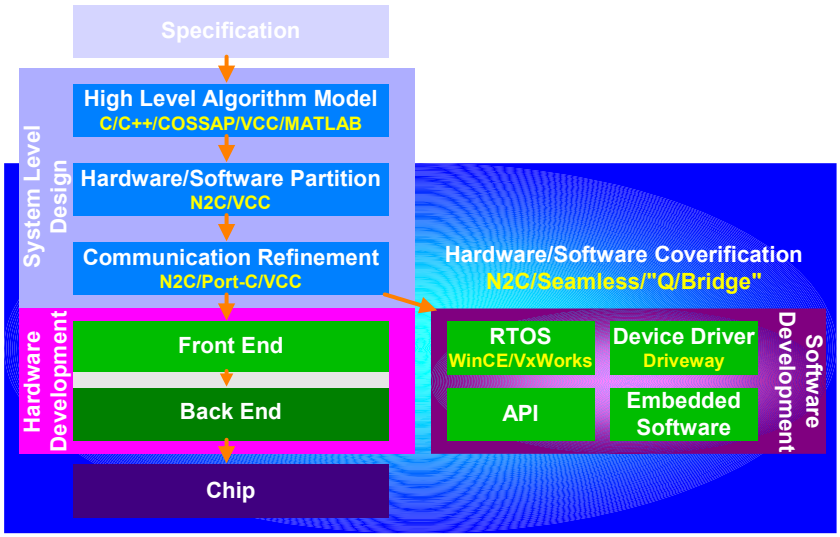

Figure 1 Soc design flow

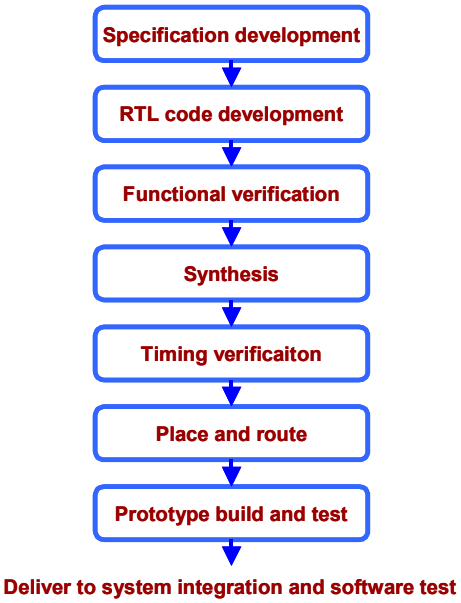

Figure 2 Traditional ASIC design flow.

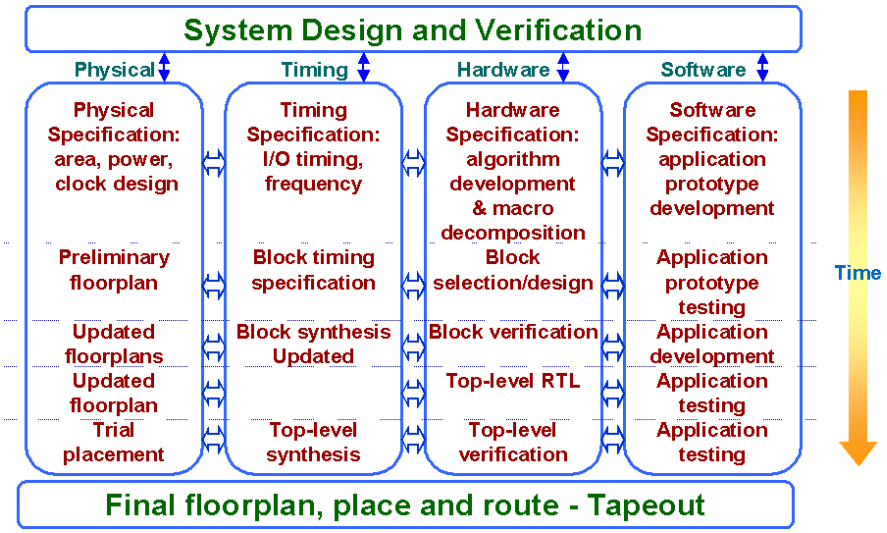

Figure 3 Spiral SoC design flow.

Instead of using traditional ASIC design flow, spiral SoC design flow, which involves parallel and concurrent development of hardware and software, is used in this course. Since the hardware design in RTL or behavioral level using HDL is mature, we focus on *high-level modeling for design and verification*, *hardware/software coordination in embedded system*, *authoring for hardware IP and its associated software components - the driver*. For one-semester graduate course, we have four labs and corresponding homework to guide students in IP authoring. The prerequisites for this course are cell-based VLSI design and C programming. A website located at http://twins.ee.nctu.edu.tw/courses/ip_core_01/index.html contains all information about this course, including handouts for both classes and labs, related readings and resources, and auxiliary file packages for the labs.

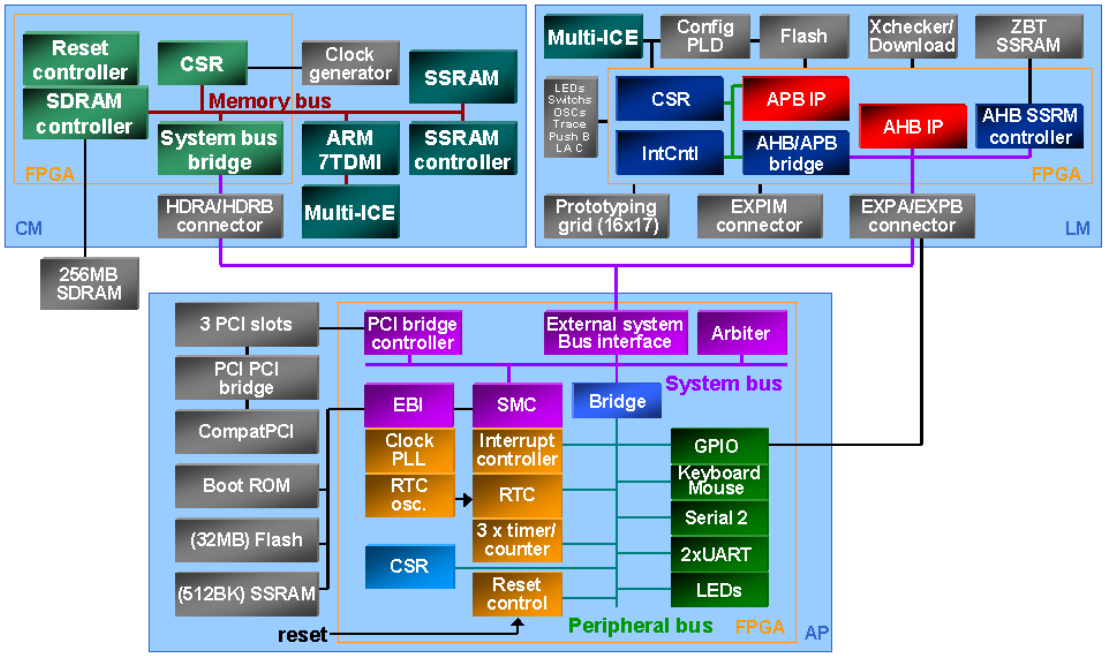

Figure 4 Target platform: ARM Integrator.

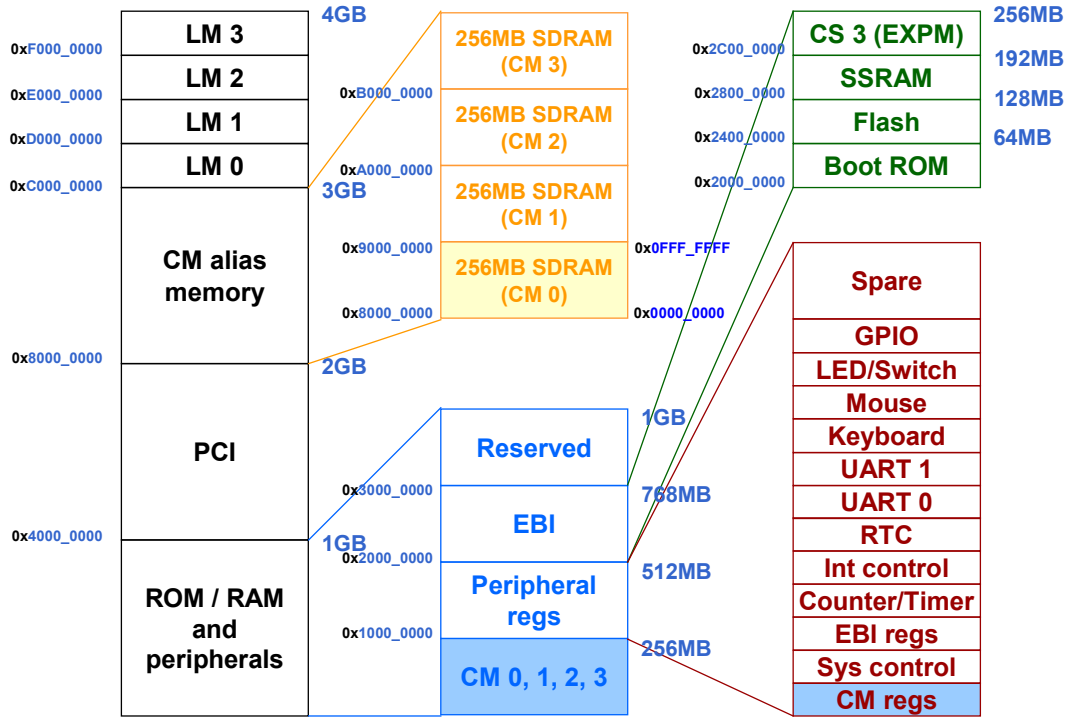

Figure 5 System memory map of ARM Integrator.

Organization of this material

The rest of this material is organized as follows. In Lab 1, The ARM development tools and environment are introduced to familiarize students with software development environment, writing code (driver) for ARM-based platform design and software cost (code size) estimation. The debug environment can also be used to debug both software and hardware design running at the target platform, which is ARM Integrator platform in this course. Mixed instruction sets, ARM and Thumb interworking, is learned to balance the performance and code density of an application.

Profiling utility can be used to estimate percentage time of each function in an application. This information can be used as one of the references to decide which portion of the application should be designed as hardware to meet performance constraints. The cost of a program includes Read Only (RO) data, Read Write (RW) data and Zero-Initialized (ZI) data. Embedded systems often implement complex memory configurations. For example, an embedded system might use fast, 32-bit RAM for performance-critical code, such as interrupt handlers and the stack, slower 16-bit RAM for application RW data, and ROM for normal application code.

In Lab 2, the resource of target platform, ARM Integrator, is introduced, as shown in Figure 4 and Figure 5. This platform has the general characteristics of platform-based design, i.e., a set of pre-built, well-defined, well-verified hardware and software components. While the hardware components is clearly showed in Figure 4, the software components include:

- the hardware abstraction layer, uHAL library, of those hardware shown in Figure 4
- a set of Application Programming Interface (API), including API for Flash memory and PCI
- a ported OS, uC/OS-2
- basic applications, e.g., boot monitor, and example codes to make use of ARM Integrator

Lab 2 also introduces the booting procedure and interrupt handler of an embedded system, hardware/software communication through memory mapped I/O or uHAL library, and the real condition of the use of difference memory organization, which has to be taken the following considerations into account:

- performance & constraint of different types of memory (SSRAM, SDRAM, and Flash, and possible the cache)
- data alignment and data layout (in which type of memory)
- available data bus and memory bandwidth

ARM Integrator platform enables the integration of software and hardware IP and associated drivers, and reduces development times and increases levels of confidence in the final silicon by allowing early prototyping of an environment similar to the final system (using programmable and standard components). To extent the functionality of ARM Integration, three approaches can be applied: semihosting, user-programmable logic elements, and General Purpose Input/Output (GPIO). Semihosting, a mechanism whereby the target communicates I/O requests made in the application code to the host system, rather than attempting to support the I/O itself, is introduced in lab 2. Extension using user-programmable logic elements is introduced in Lab 4. The use of GPIO is not included in this course.

In Lab 3, high-level modeling for design and verification is introduced to help detect integration problems earlier in the design cycle. Virtual prototype, one types of HW/SW co-verification environments, is chosen in this course to make best use of available tools, ARM Developer Suite (ADS). In virtual prototype, the processor is modeled as an Instruction Set Simulator (ISS) and hardware functional blocks are represented with C models. The virtual prototype allows designers to do the following:

- Make trade-offs by modifying system parameters and checking the results
- Test interrupt handlers

- Develop and test drivers for your IPs
- Test the correctness of the application algorithms.

Using the ARMulator, it is possible to build a complete, clock-cycle accurate software model of a system including MMU, physical memory, peripherals, and OS. Since this is likely to be the highest-level model of the system, it is one of the best places to perform the initial evaluation of design alternatives before detailed RTL design. Once the design is reasonably stable, hardware development will probably move into a timing-accurate design environment, but software development can continue using the ARMulator-based model.

In Lab 4, value-added IP is attached to the AMBA™ backbone and ARM Integrator system infrastructure by using the user-programmable logic elements. This allows users to prototype, test and integrate their own HDL hardware IP and associated drivers before final silicon. While the FPGA design is relatively mature, this lab focuses on the design flow for ARM-based design environment, AMBA-compliant IP and hardware/software coordination.

Each Lab has its corresponding exercises and homework. The exercises are designed to familiarize students with the content of the Lab, while the homework is designed to familiarize them with the design flow shown in Figure 1 by using JPEG as an example: hardware/software partition through profiling (homework 1), make use of existing hardware resource and hardware/software communication mechanism (homework 2), hardware IP modeling and software driver authoring (homework 3), and finally FPGA-proven AMBA-complaint IP authoring (homework 4).