SOC Design Lab

http://twins.ee.nctu.edu.tw/courses/soclab_04/index.html

Adopted from National Chiao-Tung University
IP Core Design

Outline

- ***Introduction to SoC***
- ARM-based SoC and Development Tools
- SoC Labs
- Available Lab modules in this course
- Summary

SoC: System on Chip

- System

  A collection of all kinds of components and/or subsystems that are appropriately interconnected to perform the specified functions for end users.

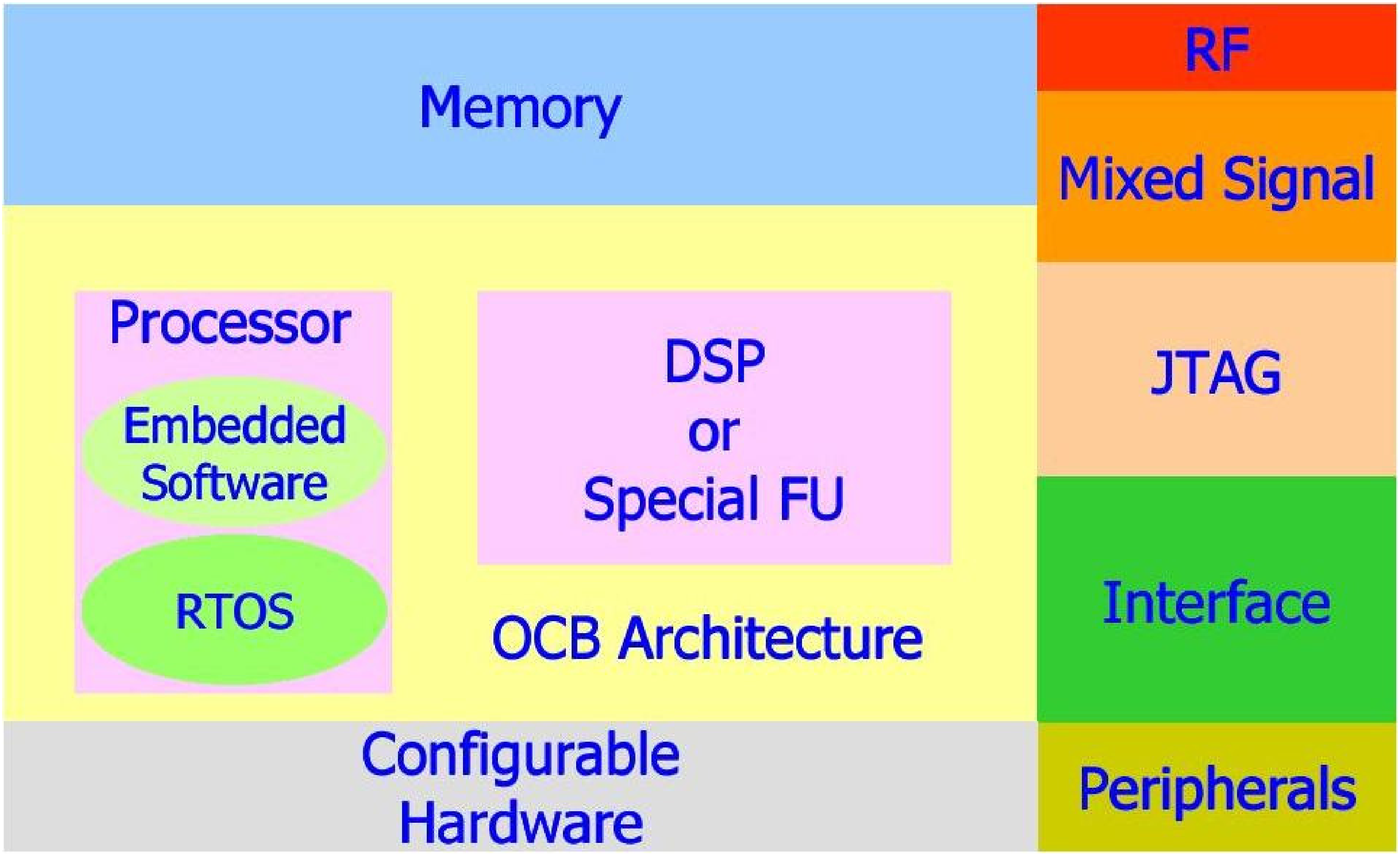
- A SoC design is a “product creation process” which
  - Starts at identifying the end-user needs
  - Ends at delivering a product with enough functional satisfaction to overcome the payment from the end-user

SoC Definition

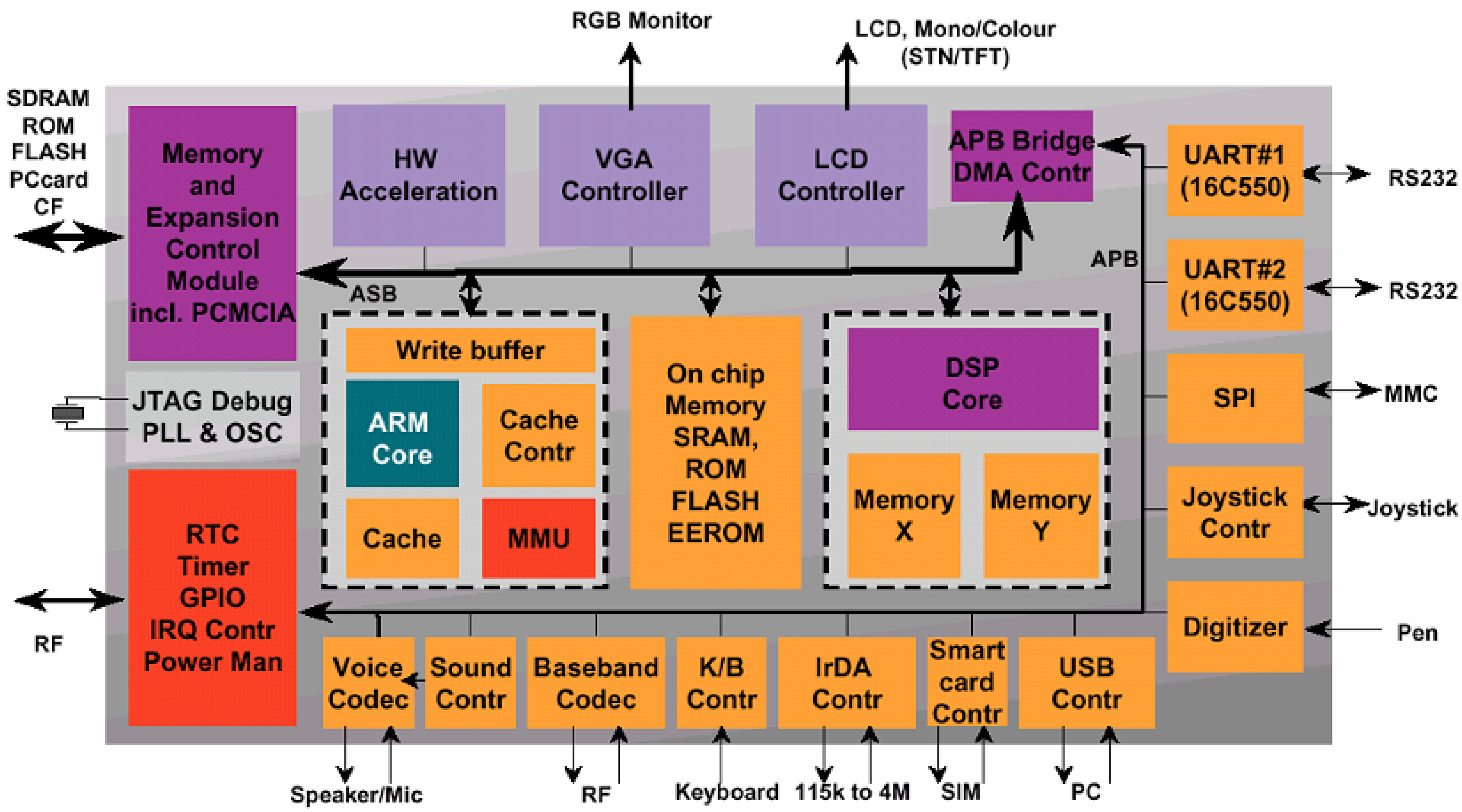
- Complex IC that integrates the major functional elements of a complete end-product into a single chip or chipset
- The SoC design typically incorporates
  - Programmable processor
  - On-chip memory
  - HW accelerating function units (DSP)
  - Peripheral interfaces (GPIO and AMS blocks)
  - Embedded software

Source: "*Surviving the SoC revolution – A Guide to Platform-based Design*," Henry Chang et al, Kluwer Academic Publishers, 1999

SoC Architecture



SoC Example



RGB Monitor
LCD, Mono/Colour (STN/TFT)
SDRAM ROM FLASH PCcard CF
Memory and Expansion Control Module incl. PCMCIA
HW Acceleration
VGA Controller
LCD Controller
APB Bridge DMA Contr
UART#1 (16C550)
RS232
APB
UART#2 (16C550)
RS232
ASB
Write buffer
ARM Core
Cache Contr
Cache
MMU
On chip Memory SRAM, ROM FLASH EEROM
DSP Core
Memory X
Memory Y
SPI
MMC
JTAG Debug PLL & OSC
Joystick Contr
Joystick
RTC Timer GPIO IRQ Contr Power Man
RF
Digitizer
Pen
Voice Codec
Sound Contr
Baseband Codec
K/B Contr
IrDA Contr
Smart card Contr
USB Contr
Speaker/Mic
RF
Keyboard
115k to 4M
SIM
PC

SoC Application

- Communication
  - Digital cellular phone
  - Networking
- Computer
  - PC/Workstation
  - Chipsets
- Consumer
  - Game box
  - Digital camera

Benefits of Using SoC

- ❑ Reduce overall system cost
- ❑ Increase performance
- ❑ Lower power consumption
- ❑ Reduce size

Evolution of Silicon Design

Year	1997	1998	1999	2002
Process Technology	0.35u	0.25u	0.18u	0.13u
Design Cycle (month)	18 ~ 12	12 ~ 10	10 ~ 8	8 ~ 6
Derivative Cycle (month)	8 ~ 6	6 ~ 4	4 ~ 2	3 ~ 2
Silicon Complexity (gate)	200 ~ 500 k	1 ~ 2 M	4 ~ 6 M	10 ~ 25 M
Applications	Cellular, PDAs, DVD	Set-top boxes, Wireless PDA	Internet appliances, Anything portable	Ubiquitous computing Intelligent, inter-connected con-trollers

Source: "*Surviving the SoC revolution – A Guide to Platform-based Design*," Henry Chang et al, Kluwer Academic Publishers, 1999

Challenges in SoC Era

- Time-to-market
  - Process roadmap acceleration
  - Consumerization of electronic devices
- Complex systems
  - μCs, DSPs, HW/SW, SW protocol stacks, RTOS's, digital/analog IPs, On-chips buses
- Deep submicron effects
  - Crosstalk, electronmigration, wire delays, mask costs

How to Conquer the Complexity

- Use a known real entity
  - A pre-designed component (IP reuse)
  - A platform (architecture reuse)
- Partition
  - Based on functionality
  - Hardware and software
- Modeling
  - At different level
  - Consistent and accurate

What is IP?

- Intellectual Property (IP)
  - Intellectual Property means products, technology, software, etc. that have been protected through patents, copyrights, or trade secrets.
- Virtual Component (VC)
  - A block that meets the Virtual Socket Interface Specification and is used as a component in the Virtual Socket design environment. Virtual Components can be of three forms — Soft, Firm, or Hard. (VSIA)
- Also named mega function, macro block, reusable component

Reusable Component

- A design object

  This refers to the type of components for which a physical implementation exists that can be reused. For example, ALU chips or macrocells that can be embedded in larger chips, etc. These designs are largely implemented in specific technologies. Limited parameterization may be possible. These design objects typically exist in technology libraries from one or more suppliers.

  ---- EDA Industry Standard Roadmap 1996

Types of IP

	Design Flow	Representation	Libraries	Technology	Portability
Soft - Very flexible - Not predicable	System Design RTL Design	Behavioral RTL	N/A	Technology Independent	Unlimited
Firm - Flexible - Predicable	Synthesis Floorplanning Placement	RTL & Constraints Netlist	Reference Library - Footprint - Timing model - Wiring model	Technology Generic	Library Mapping
Hard - Not flexible - Very predicable	Routing Verification	Polygon Data	Specific Library - Characterized cells - Fixed process rules	Technology Fixed	Process Porting

IP Value

- Foundation IP – Cell, MegaCell
- Star IP – ARM (low power)
- Niche IP – JPEG, MPEGII, TV, Filter
- Standard IP – USB, IEEE1394, ADC, DAC
- ……..

IP Sources

- ❑ Legacy IP
  - from previous IC
- ❑ New IP
  - specifically designed for reuse
- ❑ Licensed IP
  - from IP vendors

Why IP

- Don't know how to do it
- Cannot wait for new in-house development
- Standard/Compatibility calls for it
  - PCI, USB, IEEE1394, Bluetooth
  - software compatibility

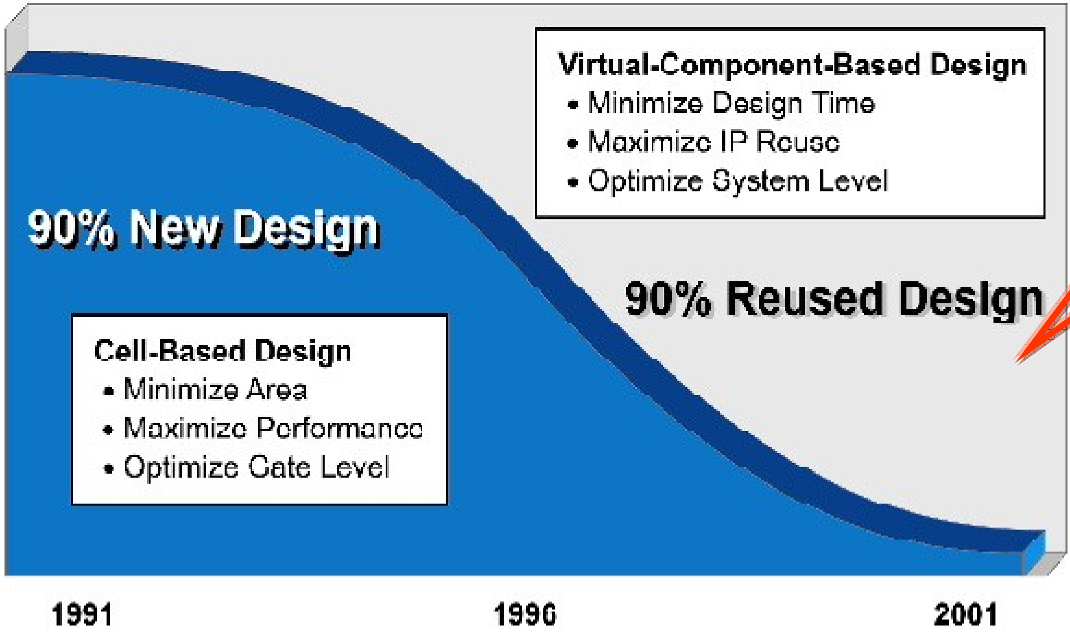
Differences in Design Between IC and IP

- Limitation of IC design
  - Number of I/O pin
  - Design and Implement all the functionality in the silicon
- Soft IP
  - No limitation on number of I/O pin
  - Parameterized IP Design: design all the functionality in HDL code but implement desired parts in the silicon
  - IP compiler/Generator: select what you want !!
  - More high level auxiliary tools to verify design
  - More difficult in chip-level verification
- Hard IP
  - No limitation on number of I/O pin
  - Provide multiple level abstract model
  - Design and Implement all the functionality in the layout

IP-Based SoCs

- An Evolutionary Path
  - Early days
    - IP/Cores not really designed for reuse (no standard deliverables)
    - Multiple Interfaces, difficult to integrate
  - IPs evolved: parameterization, deliverables, verification, synthesis
  - On-Chip bus standards began to appear (e.g, IBM CoreConnect, ARM AMBA, OCP-IP)
- Reusable IP + Common on-chip bus architectures
  - 1998: max number of cores > 30 cores
    core content between 50% and 95%

Design Reuse Evolution



- IP reuse
- Architecture reuse (Platform design)

Today's platform is a virtual component for tomorrow's platform
-VSIA PBD SG

Source: EEDesign, http://www.eedesign.com/story/OEG20020301S0104

History and Status of IP Reuse

- 1999 Reuse methodology manual
- Biggest change
  - Reuse is no longer a proposal
  - A solution **practiced** today for state-of-the art SOC design
- New business model emerges
  - IP provider and IP integrator
  - Like TSMC break IDM
- Top three IP providers (2002)
  - ARM, Rambus, MIPS
- Industry forum
  - worldwide: Virtual socket interface alliance (VSIA)
  - Taiwan: Taiwan SOC consortium

Design for Use First

- To be reusable, **useable first**
- Good design techniques
  - Good documentation,
  - good code,
  - thorough commenting,
  - well designed verification environments,
  - robust scripts

Then Design for Reuse

- ❑ Designed to solve a **general problem**
  - Configuration and parameters
- ❑ Designed for use in multiple technologies
- ❑ Designed with **standards-based interfaces**
- ❑ Designed with complete **verificatio**n process
  - Robust and verified
- ❑ Design verified to a high level of confidence
  - Physical prototype, silicon proven, demo system
- ❑ Design with fully **documented** in terms of appropriate applications and restrictions

Some Questions for Design Reuse

- ❑ Why should I design for reuse ?
- ❑ Should I plan all my design for reuse ?
- ❑ What is design reuse ?
- ❑ How to design reuse ?
- ❑ Who shall do design reuse ?
- ❑ When shall I do design reuse ?

Why “Reuse” are not used ?

I don’t have time to learn these “reuse” stuff, e.g. coding guidelines, documents, and various verification. I have to get my design done. It lengthen the design cycle.

- It’s true that
  - Reuse cost is very **expensive**
    - Explicit design reuse requires a dedicated team
    - 10x or an order of magnitude higher
- But it **rewards**
  - Benefit
    - 2-3X benefit for implicit reuse
    - 10X-100X productivity gain in successive design for explicit reuse
  - Don’t you have any coding style ? It’s a matter of habit.
  - Don’t you have to write any form of documents, too ?
  - Don’t you have to verify your design, too ?
    - Shortcut working style saves time now, but it pays more latter

Whether to adopt reuse is a managerial and cultural issue.

Challenge for Designers

- Not whether to adopt reuse
- But how to employ it **effectively**
  - Easy to integrate (**interface**)
  - Robust to integrate (**function**)
- Trends
  - Buy whatever IP they can and developing reusable macros only when necessary
    - problems: IP with functional bugs and poor documentation
  - **Standard-based interfaces**
  - **Converging** rather than diverging in their basic structure, with differentiation by software or specialized computational units

Paradigm Shift

- Conventional flow
  - System/design house for RTL design, synthesis
  - ASIC vendor for physical implementation

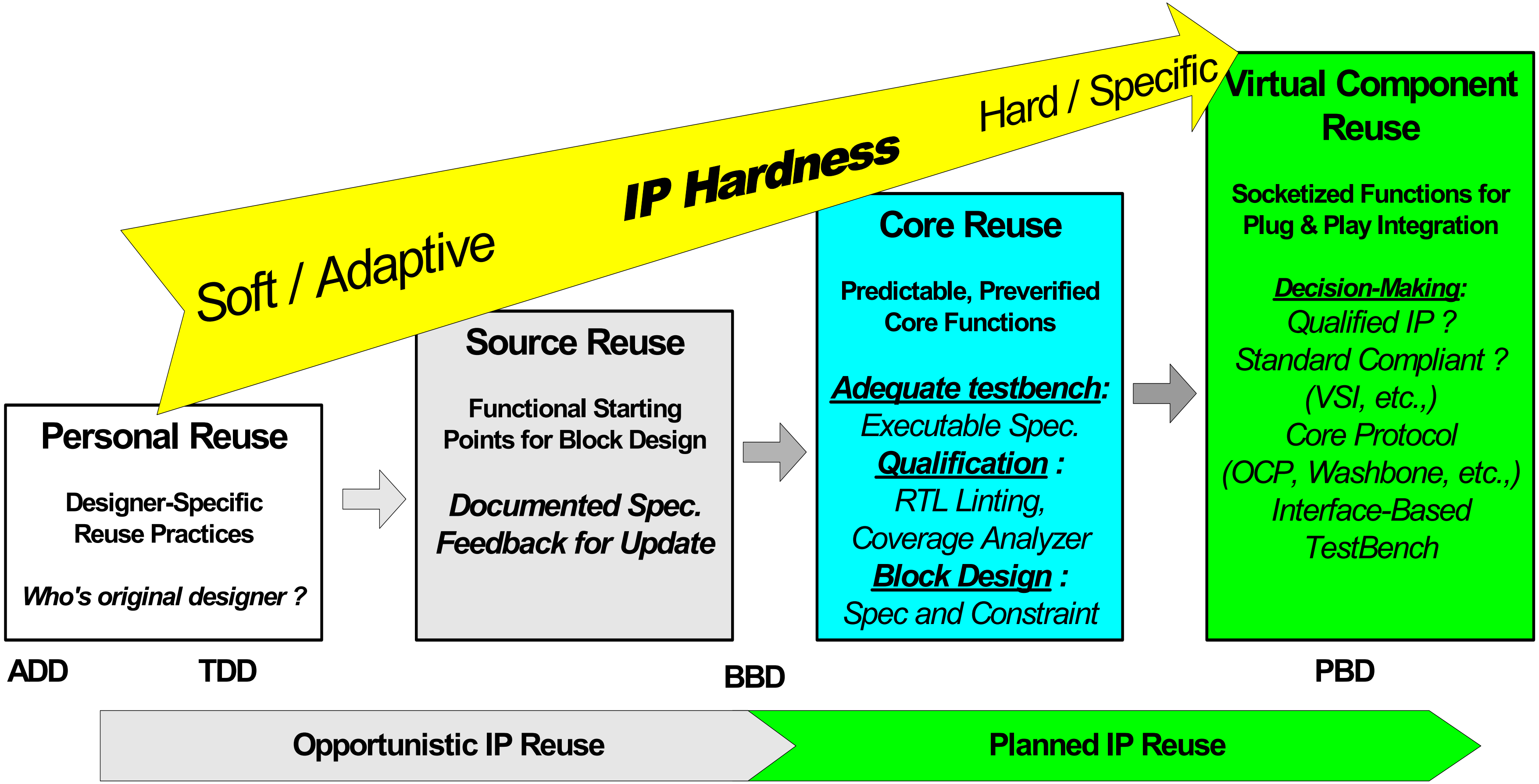
But SOC design is too complicated to be handled in-house solely

- New flow
  - System/design house provide H/W spec. only and focus on valued-added S/W and application
  - ASIC vendor offer IPs and integration service

Survive for New Climate

- ❑ For system/design house
  - – Improve S/W and system architecture skill to differentiate the products
- ❑ For ASIC house
  - – Traditionally, ASIC vendors do not have RTL-based design expertise
  - – Learn to develop, integrate and manage reusable IPs

Reuse :The Key to SoC Design



Outline

- ❑ Introduction to SoC
- ❑ ***ARM-based SoC and Development Tools***
- ❑ SoC Labs
- ❑ Available Lab modules in this course
- ❑ Summary

ARM-based System Development

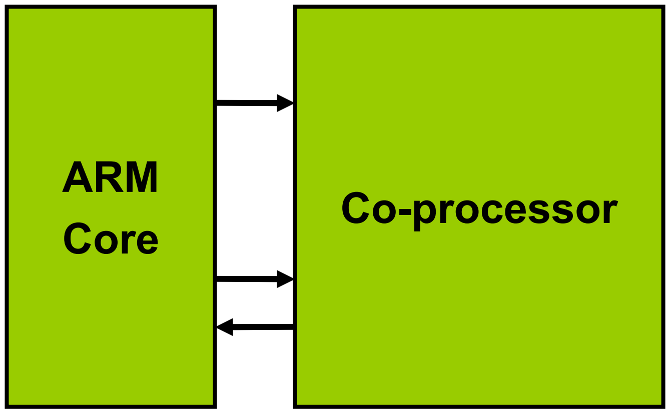
- ❑ Processor cores
- ❑ ARM On-Chip Bus: AMBA
- ❑ Platform: PrimeXsys
- ❑ System building blocks: PrimeCell
- ❑ Development tools
  - – Software development
  - – Debug tools
  - – Development kits
  - – EDA models
  - – Development boards

ARM Architecture Version

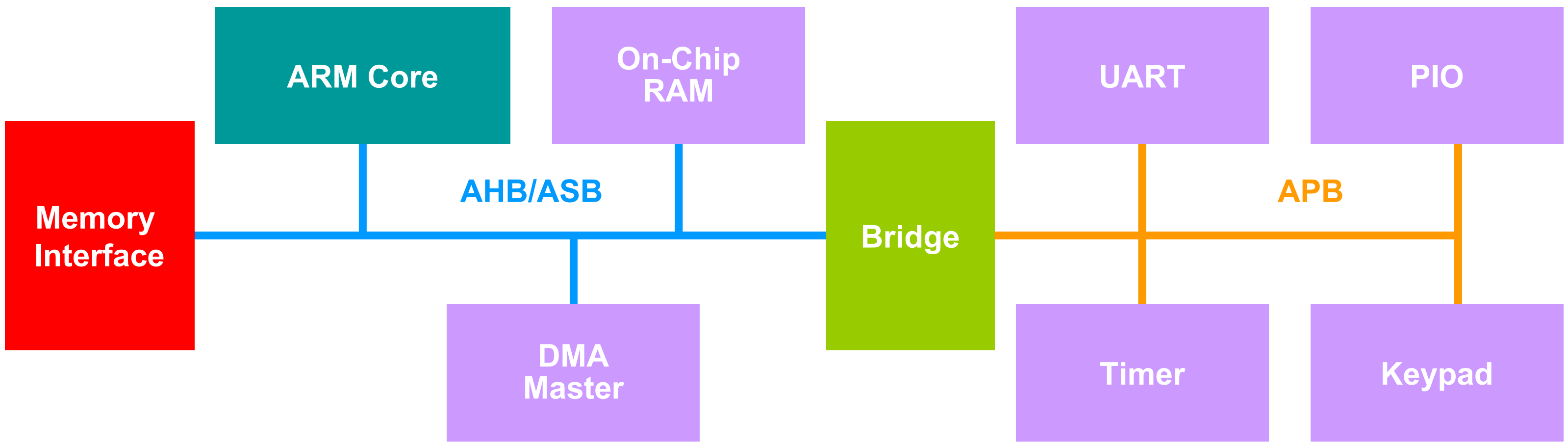
Core	Architecture
ARM1	v1
ARM2, ARM2as, ARM3	v2
ARM6, ARM60, ARM610, ARM7, ARM710, ARM7D, ARM7DI	v3
ARM7TDMI, ARM710T, ARM720T, ARM740T	v4T
StrongARM, ARM8, ARM810	v4
ARM9TDMI, ARM920T, ARM940T	v4T
ARM9E-S, ARM10TDMI, ARM1020E	v5TE
ARM7EJ-S, ARM926EJ-S, ARM1026EJ-S	v5TEJ
ARM11	v6

ARM Coprocessors

- Application specific coprocessors
  - e.g., for specific arithmetic extensions
  - Developed a new decoupled coprocessor interface
  - Coprocessor no longer required to carefully track processor pipeline.



ARM On-Chip Bus



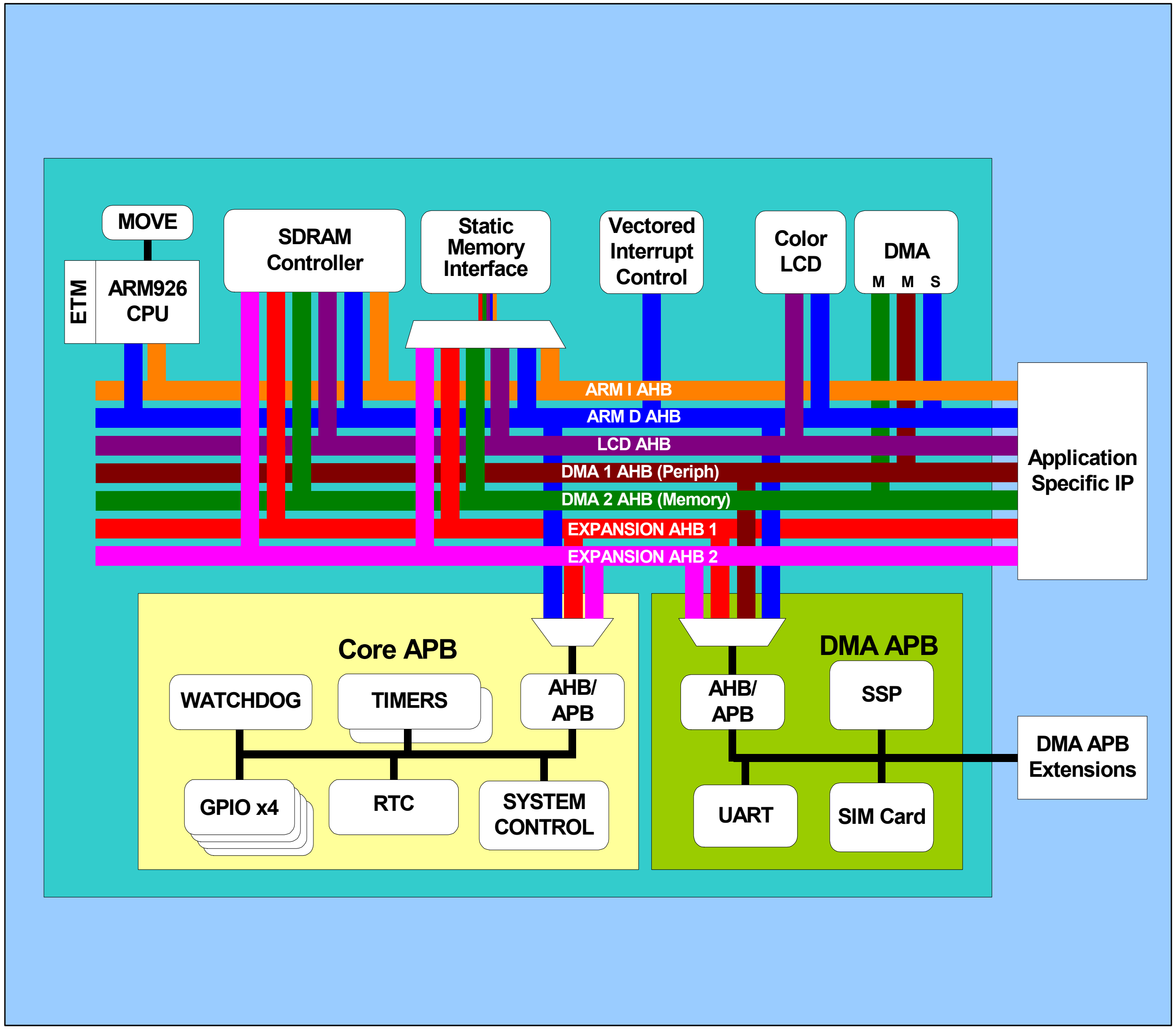
A typical AMBA system

AHB: Advanced High-performance Bus

ASB: Advanced System Bus

APB: Advanced Peripheral Bus

ARM PrimeXsys Wireless Platform



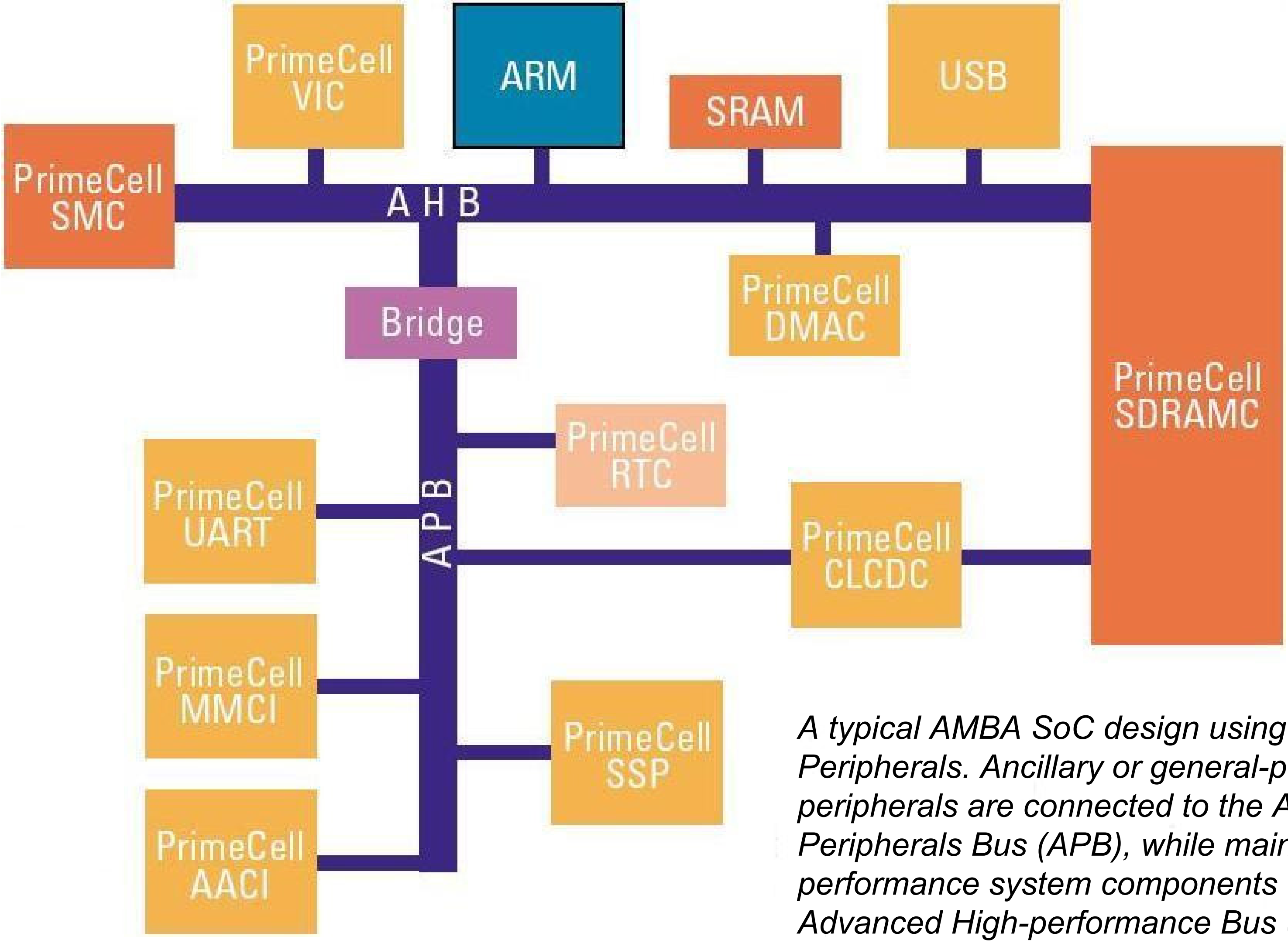
PrimeXsys

- It is no longer the case that a single Intellectual Property (IP) or silicon vendor will be able to supply all of the IP that goes into a device.
- With the ***PrimeXsys*** range, ARM is going one step further in providing a known framework in which the IP has been integrated and proven to work.
- Each of the ***PrimeXsys*** platform definitions will be application focused – there is no ‘one-size-fits-all’ solution.
- ARM will create different platform solutions to meet the specific needs of different markets and applications.

PrimeCell (1/2)

- ARM ***PrimeCell*** Peripherals are re-usable soft IP macrocells developed to enable the rapid assembly of SoC designs.
- Fully verified and compliant with the AMBA on-chip bus standard, the ARM PrimeCell range is designed to provide integrated right-first-time functionality and high system performance.
- Using the ARM PrimeCell Peripherals, designers save considerable development time and cost by concentrating their resources on developing the system design rather than the peripherals.

PrimeCell (2/2)



A typical AMBA SoC design using PrimeCell Peripherals. Ancillary or general-purpose peripherals are connected to the Advanced Peripherals Bus (APB), while main high-performance system components use the Advanced High-performance Bus (AHB).

ARM's Point of View of SoCs

- Integrating hardware IP
- Supplying software with the hardware
- ARM has identified the minimum set of building blocks that is required to develop a platform with the basic set of requirements to:
  - Provide the non-differentiating functionality, pre-integrated and pre-validated;
  - Run an OS;
  - Run application software;
  - Allow partners to focus on differentiating the final solution where it actually makes a difference.

ARM-based System Development

- Processor cores
- ARM On-Chip Bus: AMBA
- Platform: PrimeXsys
- System building blocks: PrimeCell
- ***Development tools***
  - Software development
  - Debug tools
  - Development kits
  - EDA models
  - Development boards

Main Components in ADS (1/2)

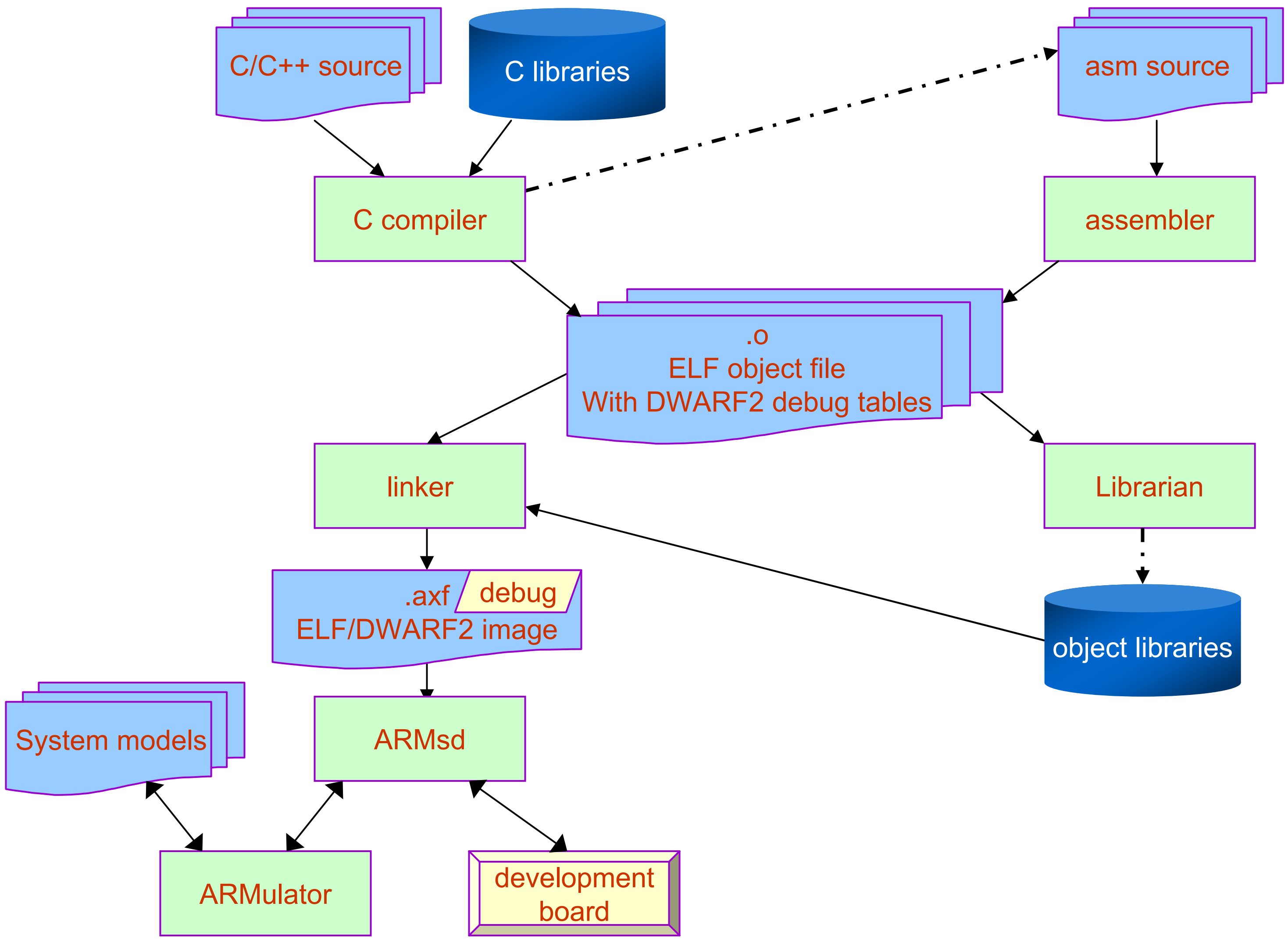
- ANSI C compilers – armcc and tcc
- ISO/Embedded C++ compilers – armcpp and tcpp
- ARM/Thumb assembler - armasm
- Linker - armlink
- Project management tool for windows - CodeWarrior
- Instruction set simulator - ARMulator
- Debuggers - AXD, ADW, ADU and armsd
- Format converter - fromelf
- Librarian – armar
- ARM profiler - armprof

ADS: ARM Developer Suite

Main Components in ADS (2/2)

- C and C++ libraries
- ROM-based debug tools (ARM Firmware Suite, AFS)
- Real time debug and trace support
- Support for all ARM cores and processors including ARM9E, ARM10, Jazelle, StrongARM and Intel Xscale

The Structure of ARM Tools



DWARF: Debug With Arbitrary Record Format **ELF: Executable and linking format**

View in CodeWarrier

- The CodeWarrior IDE provides a simple, versatile, graphical user interface for managing your software development projects.
- Develop C, C++, and ARM assembly language code
- Targeted at ARM and Thumb processors.
- It speeds up your build cycle by providing:
  - comprehensive project management capabilities
  - code navigation routines to help you locate routines quickly.

ARM Emulator: ARMulator (1/2)

- A suite of programs that models the behavior of various ARM processor cores and system architecture in software on a host system
- Can be operated at various levels of accuracy
  - Instruction accurate
  - Cycle accurate
  - Timing accurate

ARM Emulator: ARMulator (2/2)

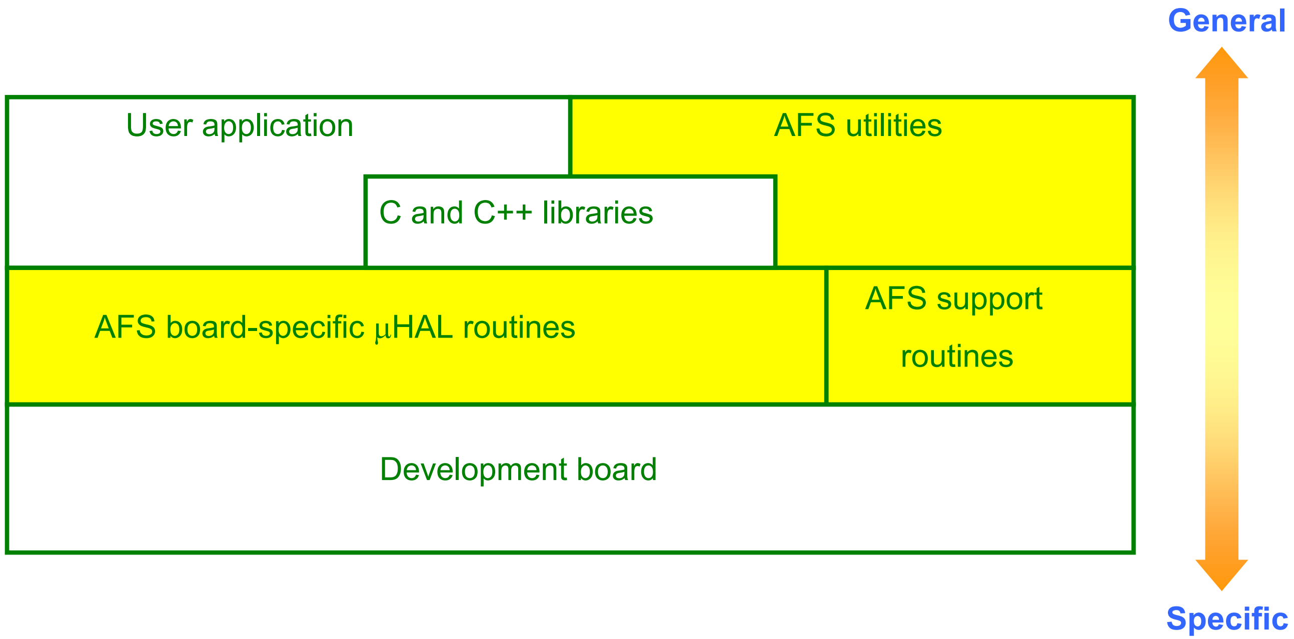
- Benchmarking before hardware is available
  - Instruction count or number of cycles can be measured for a program
  - Performance analysis
- Run software on ARMulator
  - Through ARMsd or ARM GUI debuggers, e.g., AXD
  - The processor core model incorporates the remote debug interface, so the processor and the system state are visible from the ARM symbolic debugger
  - Supports a C library to allow complete C programs to run on the simulated system

ARM μHAL API

- μHAL is a Hardware Abstraction Layer that is designed to conceal hardware difference between different systems
- ARM μHAL provides a standard layer of board-dependent functions to manage I/O, RAM, boot flash, and application flash.
  - System initialization software
  - Serial port
  - Generic timer
  - Generic LEDs
  - Interrupt control
  - Memory management
  - PCI interface

µHAL Examples

- ❑ µHAL API provides simple & extended functions that are linkable and code reusable to control the system hardware.



AFSF: ARM Firmware Suit

ARM Symbolic Debugger (ARMsd) (1/2)

- ARMsd: ARM and Thumb symbolic debugger
  - can single-step through C or assembly language sources, set breakpoints and watchpoints, and examine program variables or memory
- It is a front-end interface to debug program running either
  - under emulation (on the ARMulator) or remotely on a ARM development board (via a serial line or through JTAG test interface)

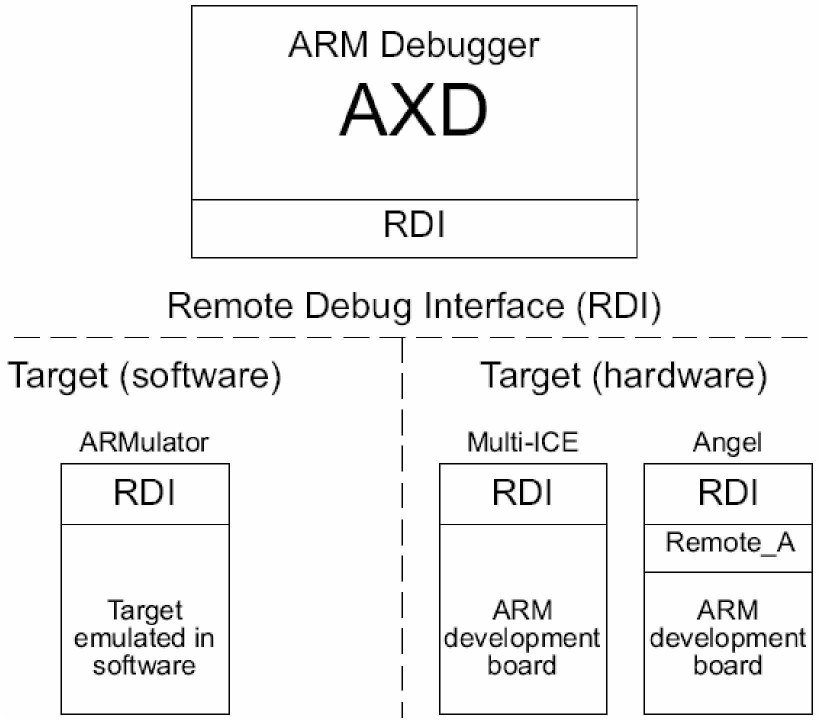
ARM Symbolic Debugger (ARMsd) (2/2)

- It allows the setting of
  - breakpoints, addresses in the code
  - watchpoints, memory address if accessed as data address

  → cause exception to halt so that the processor state can be examined

Debugger-Target Interface

- To debug your application you must choose:
  - a debugging system, that can be either hardware-based on an ARM core or software that simulates an ARM core.
  - a debugger, such as AXD, ADW, ADU, or armsd.



Debugger

- A debugger is software that enables you to make use of a debug agent in order to examine and control the execution of software running on a debug target.
- Examples: AXD, ADU, ADW, armsd
  - armsd (ARM Symbolic Debugger)
  - ADU (ARM Debugger for UNIX)
  - ADW (ARM Debugger for Windows)
  - AXD (both Windows and UNIX versions)
    - AXD is the recommended debugger. It provides functionality that is not available in the other debuggers. ADW and ADU will not be supplied in future versions of ADS.
    - The main improvements in AXD, compared to the earlier ARM debuggers, are:

      - a completely redesigned graphical user interface offering multiple views

      - a new **command-line interface**

AXD: the ARM eXtended Debugger

Debug Agent

- A debug agent performs the actions requested by the debugger, for example:
  - setting breakpoints
  - reading from memory
  - writing to memory.
- The debug agent is not the program being debugged, or the debugger itself
- Examples: ARMulator, Angel, Multi-ICE

Debug Target

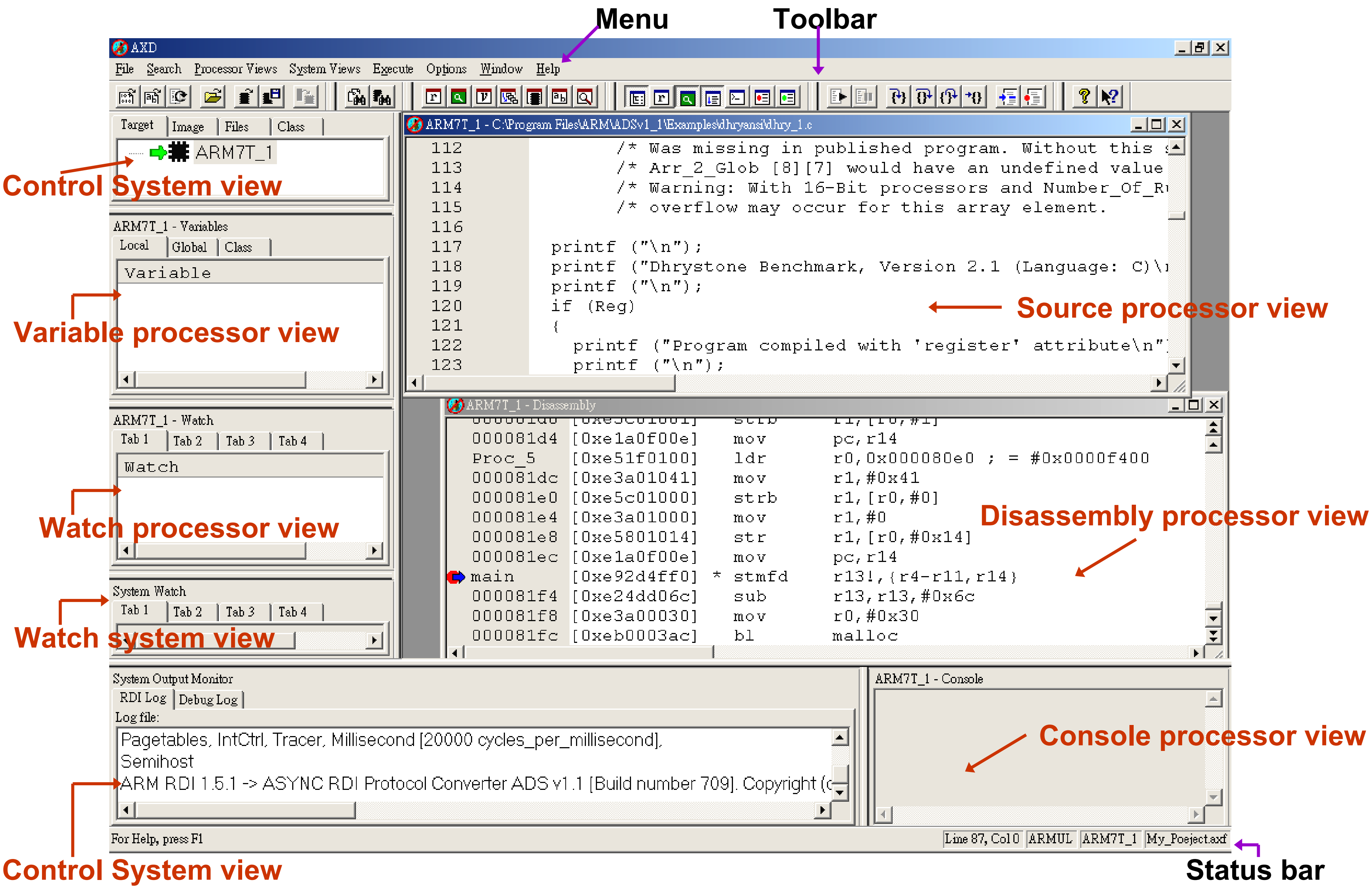
- Different forms of the debug target
  - early stage of product development, software
  - prototype, on a PCB including one or more processors
  - final product
- The form of the target is immaterial to the debugger as long as the target obeys these instructions in exactly the same way as the final product.
- The debugger issues instructions that can:
  - load software into memory on the target
  - start and stop execution of that software
  - display the contents of memory, registers, and variables
  - allow you to change stored values

Views in AXD

- Various views allow you to **examine** and **control** the processes you are debugging.
- In the main menu bar, two menus contain items that display views:
  - The items in the **Processor Views menu** display views that apply to the **current processor only**
  - The items in the **System Views menu** display views that apply to the entire, possibly **multiprocessor**, target system

AXD: the ARM eXtended Debugger

AXD Desktop


Menu
Toolbar
Control System view
Variable processor view
Watch processor view
Watch system view
Source processor view
Disassembly processor view
Console processor view
Control System view
Status bar
AXD
File Search Processor Views System Views Execute Options Window Help
Target Image Files Class
ARM7T_1
ARM7T_1 - Variables
Local Global Class
Variable
ARM7T_1 - Watch
Tab 1 Tab 2 Tab 3 Tab 4
Watch
System Watch
ARM7T_1 - C:\Program Files\ARM\ADSv1_1\Examples\dhryansi\dhry_1.c
ARM7T_1 - Disassembly
System Output Monitor
RDI Log Debug Log
Log file:
Pagetables, IntCtrl, Tracer, Millisecond [20000 cycles_per_millisecond],
Semihost
ARM RDI 1.5.1 -> ASYNC RDI Protocol Converter ADS v1.1 [Build number 709]. Copyright (c
ARM7T_1 - Console
For Help, press F1
Line 87, Col 0 ARMUL ARM7T_1 My_Poeject.axf

ARM Debug Architecture (1/2)

- Two basic approaches to debug
  - from the outside, use a logic analyzer
  - from the inside, tools supporting single stepping, breakpoint setting
- **Breakpoint**: replacing an instruction with a call to the debugger
- **Watchpoint**: a memory address which halts execution if it is accessed as a data transfer address
- **Debug request**: through ICEBreaker programming or by DBGRQ pin asynchronously

ARM Debug Architecture (2/2)

- In debug state, the core's internal state and the system's external state may be examined. Once examination is completed, the core and system state may be restored and program execution is resumed.
- The internal state is examined via a JTAG-style serial interface, which allows instructions to be serially inserted into the core's pipeline without using the external data bus.
- When in debug state, a store-multiple (STM) could be inserted into the instruction pipeline and this would dump the contents of ARM's registers.

In Circuit Emulator (ICE)

- The processor in the target system is removed and replaced by a connection to an emulator
- The emulator may be based around the same processor chip, or a variant with more pins, but it will also incorporate buffers to copy the bus activity to a “trace buffer” and various hardware resources which can watch for particular events, such as execution passing through a breakpoint

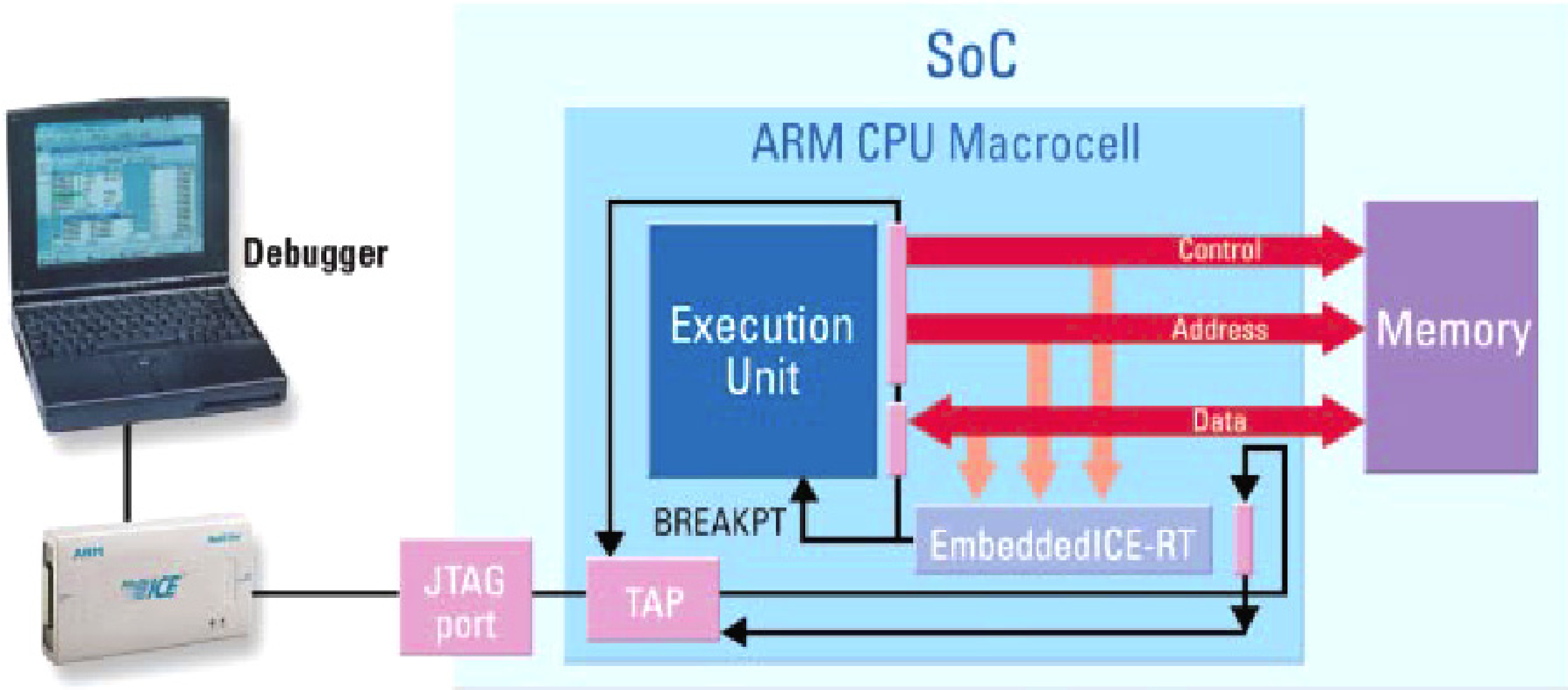
Multi-ICE and Embedded ICE

- Multi-ICE and embedded ICE are JTAG-based debugging systems for ARM processors
- They provide the interface between a debugger and an ARM core embedded within an ASIC
  - real time address-dependent and data-dependent breakpoints
  - single stepping
  - full access to, and control of the ARM core
  - full access to the ASIC system
  - full memory access (read and write)
  - full I/O system access (read and write)

Basic Debug Requirements

- Control of program execution
  - set watchpoints on interesting data accesses
  - set breakpoints on interesting instructions
  - single step through code
- Examine and change processor state
  - read and write register values
- Examine and change system state
  - access to system memory
    - download initial code

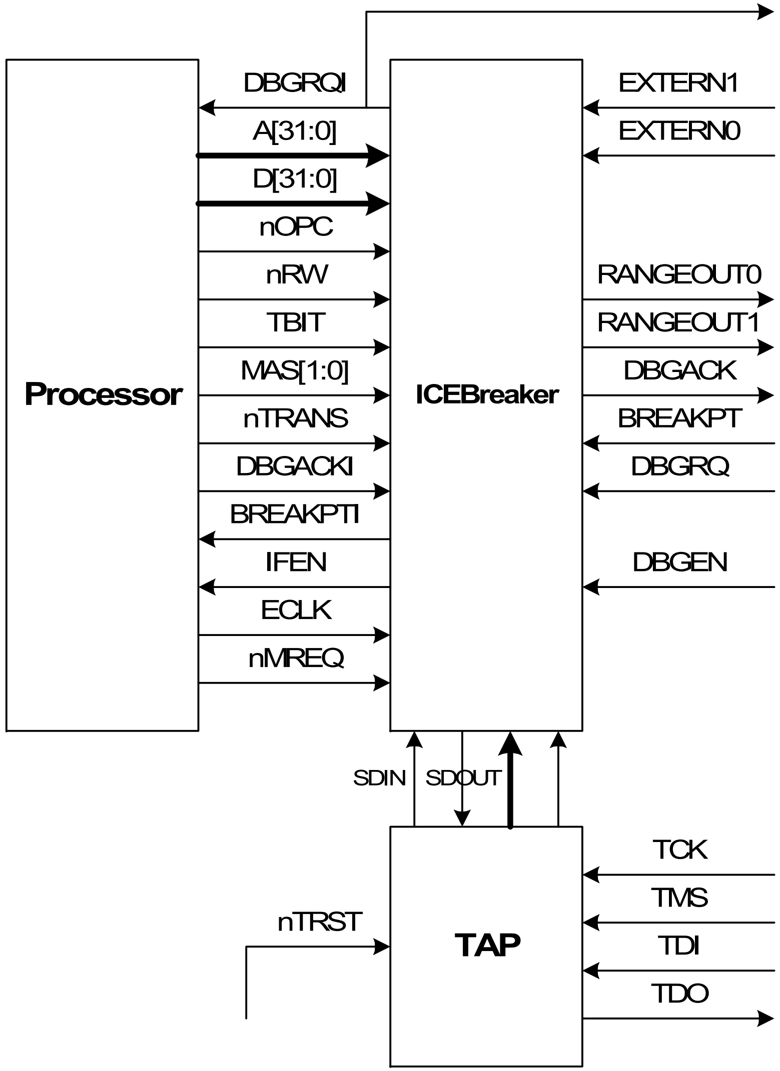
Debugging with Multi-ICE



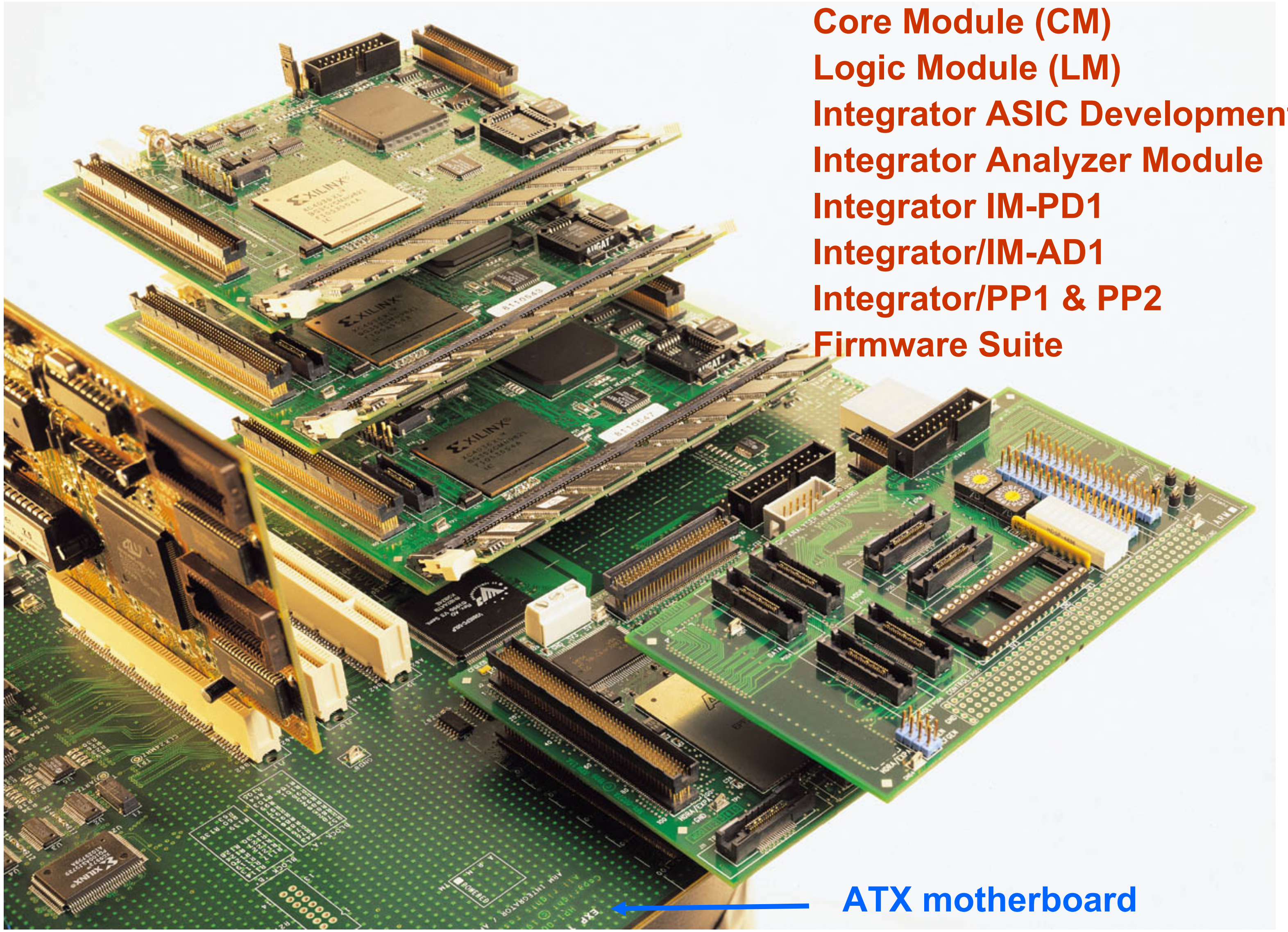
The system being debugged may be the final system

ICEBreaker (EmbeddedICE Macrocell)

- ICEBreaker is programmed in a serial fashion using the TAP controller
- It consists of 2 real-time watchpoint units, together with a control and status register
- Either watchpoint unit can be configured to be a watchpoint or a breakpoint



Integrate All The Modules in The Integrator



- Core Module (CM)
- Logic Module (LM)
- Integrator ASIC Development Platform
- Integrator Analyzer Module
- Integrator IM-PD1
- Integrator/IM-AD1
- Integrator/PP1 & PP2
- Firmware Suite

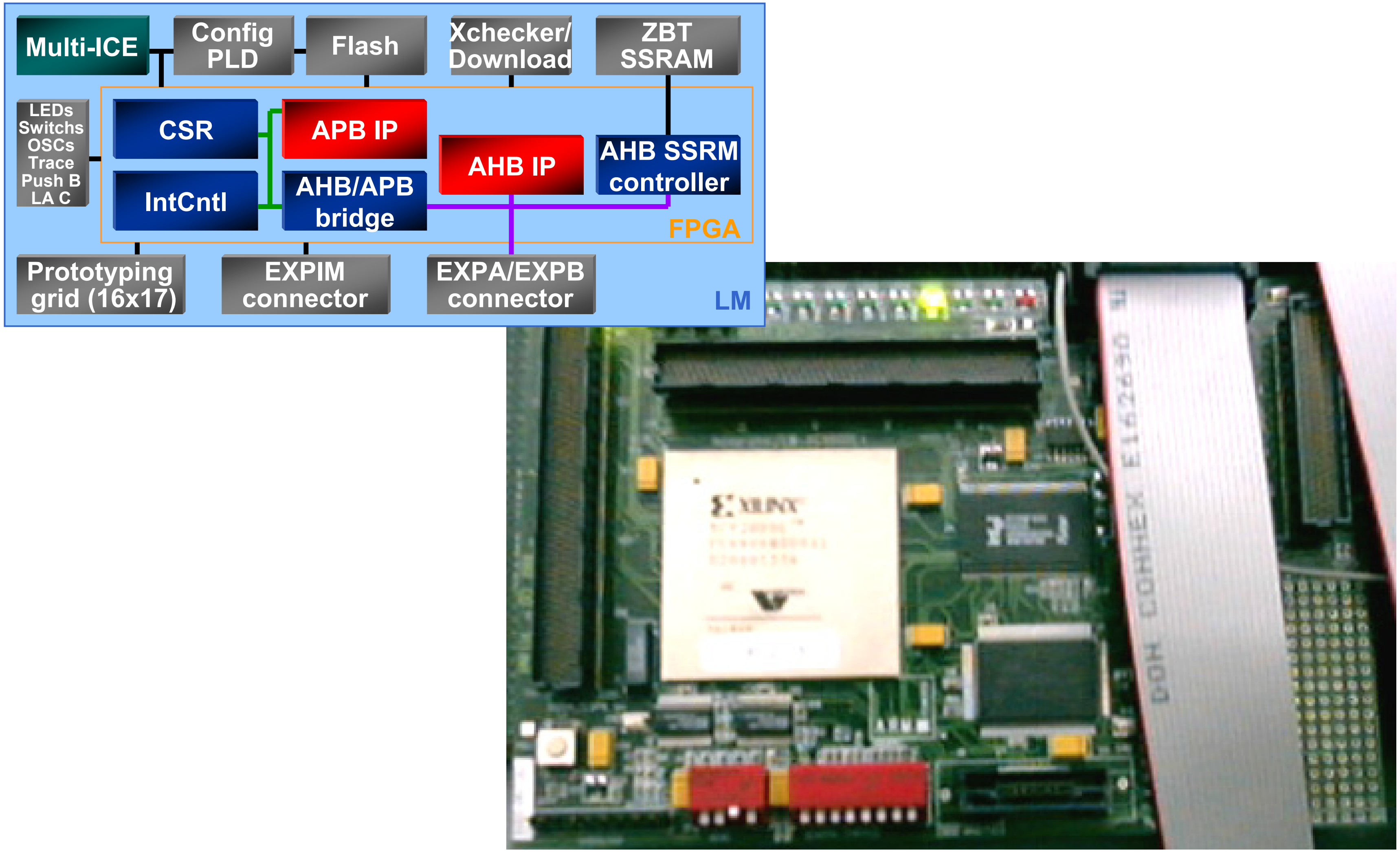
ARM Integrator within an ATX PC Case

Provided by NCTU

Inside the Case

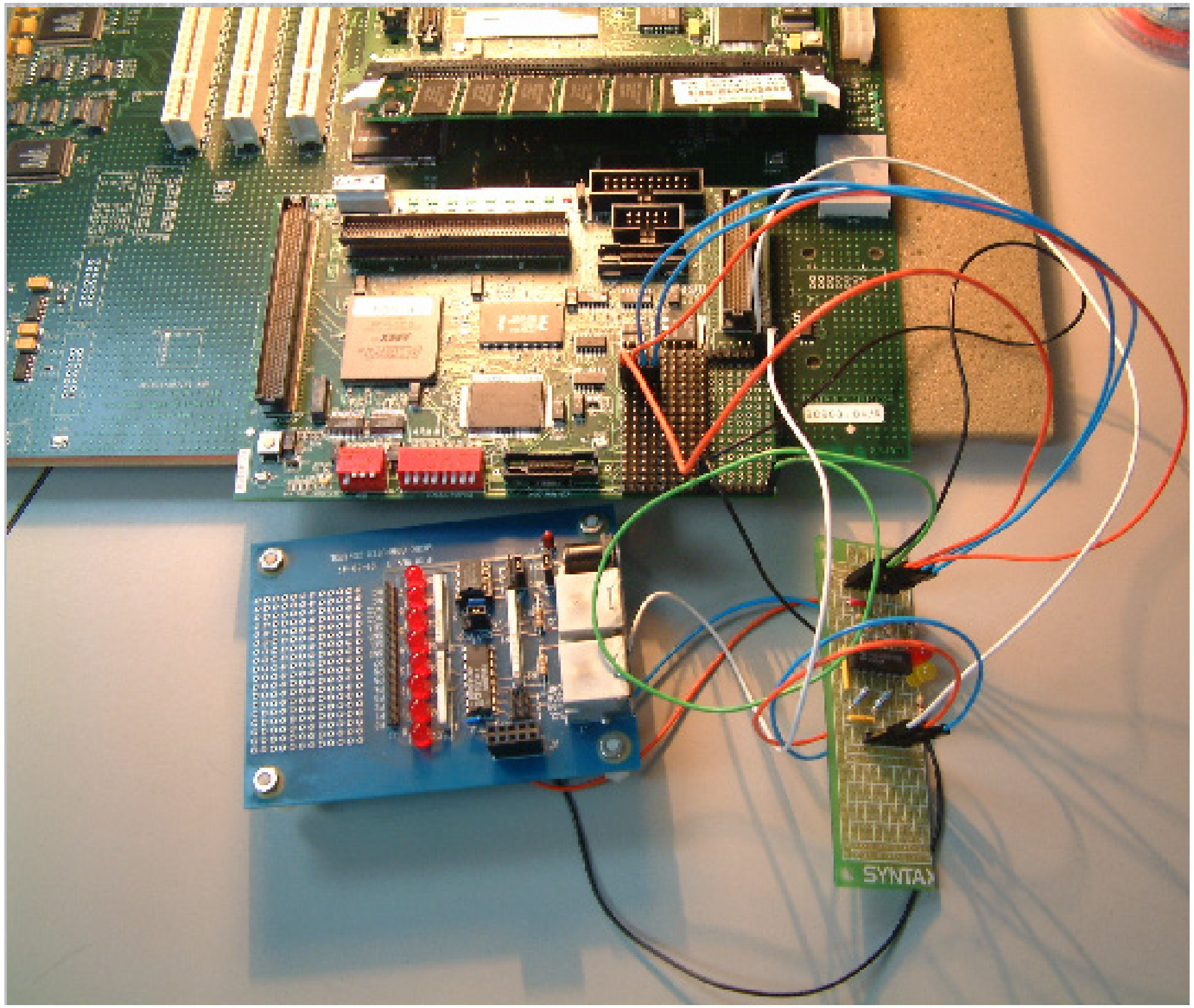
Provided by NCTU

Logic Module



Provided by NCTU

Extension with Prototyping Grid



You can use the prototyping grid to:

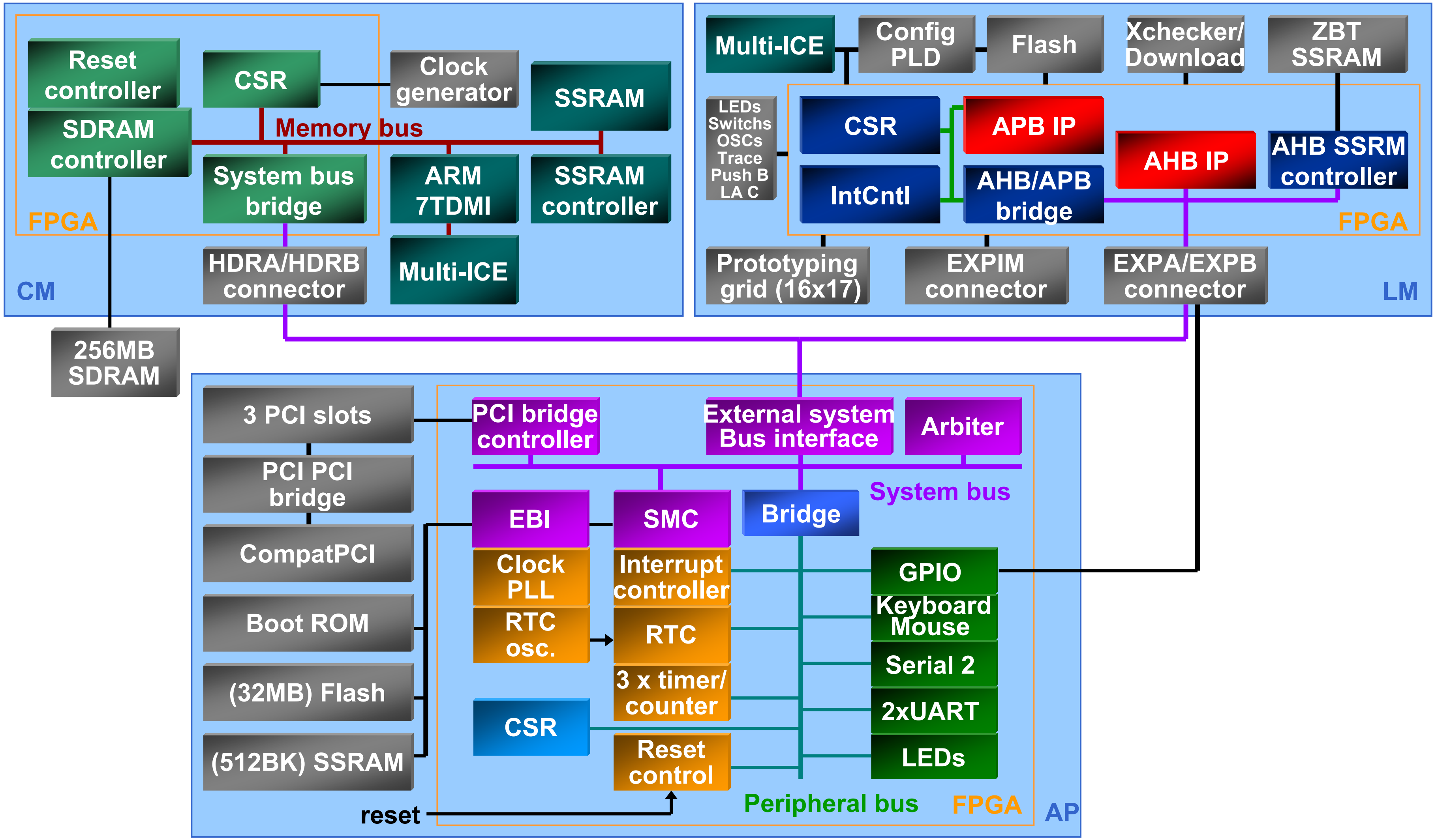
– wire to off-board circuitry

– mount connectors

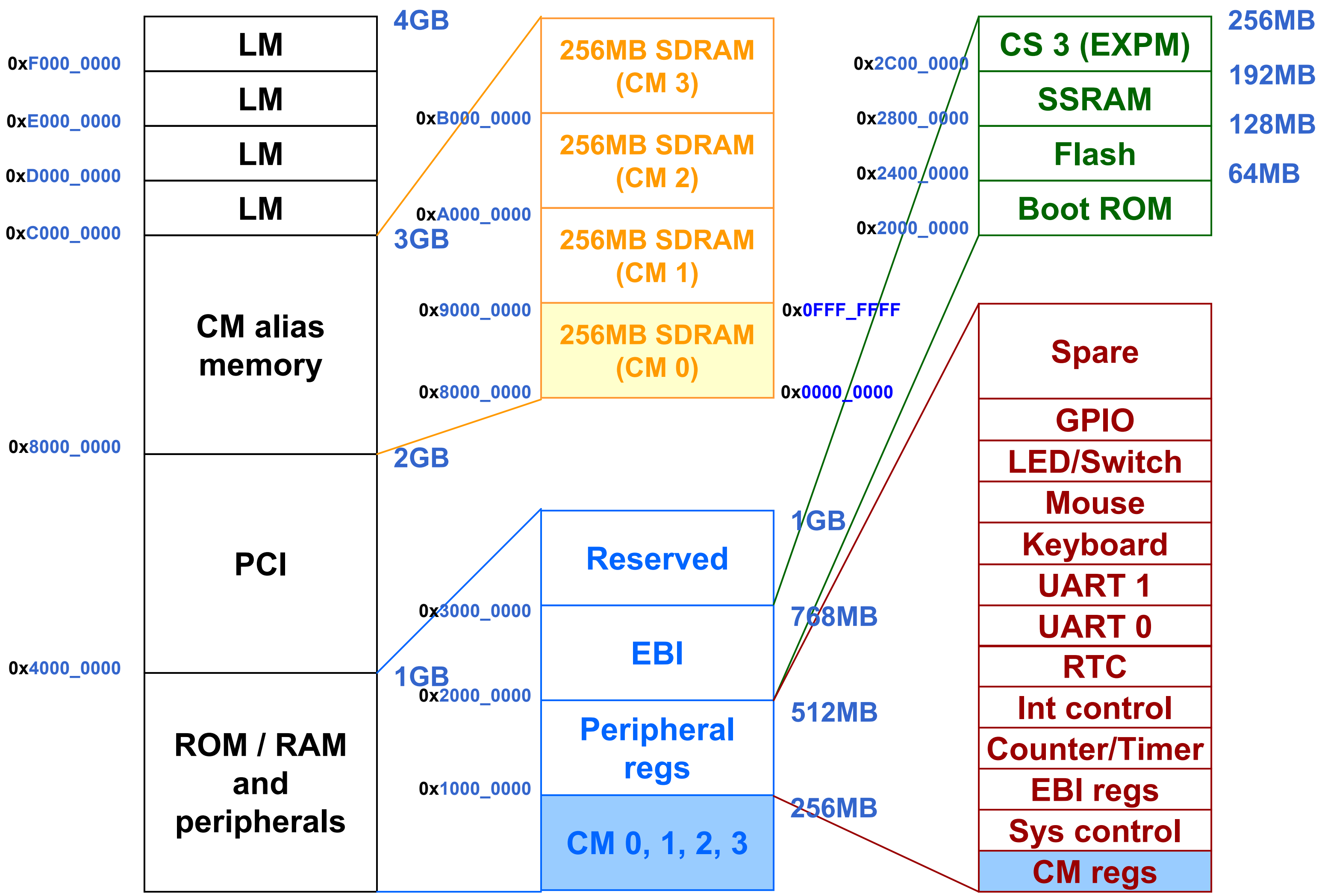
– mount small components

Provided by NCTU

ARM Integrator – One Configuration



System Memory Map



Outline

- ❑ Introduction to SoC
- ❑ ARM-based SoC and Development Tools
- ❑ ***SoC Labs***
- ❑ Available Lab modules in this course
- ❑ Summary

SoC Labs

- Code Development
- Debugging and Evaluation
- Core Peripherals and Standard I/O
- OCB/VCI
- Virtual Prototyping
- Memory Controller
- ASIC logic
- Real-Time OS (RTOS)
- Case Study

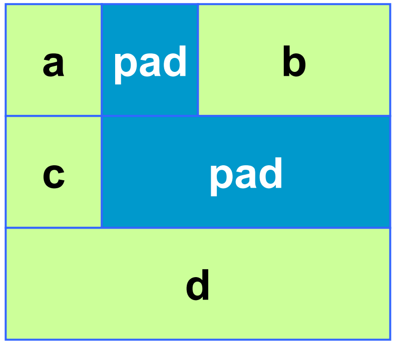
Code Development

- General/Machine-dependent guideline
  - Compiler optimization:
    - Space or speed (e.g, **-Ospace** or **-Otime**)
    - Debug or release version (e.g., **-O0** ,**-O1** or **-O2**)
    - Instruction scheduling
  - Coding style
    - Parameter passing
    - Loop termination
    - Division operation and modulo arithmetic
    - Variable type and size

Data Layout

Default

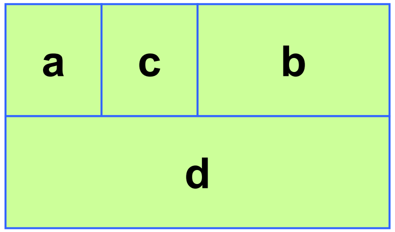
char a;
short b;
char c;
int d;



occupies 12 bytes, with 4 bytes of padding

Optimized

char a;
char c;
short b;
int d;



occupies 8 bytes, without any padding

Group variables of the same type together. This is the best way to ensure that as little padding data as possible is added by the compiler.

Stack Usage

- C/C++ code uses the stack intensively. The stack is used to hold:
  - Return addresses for subroutines
  - Local arrays & structures
- To minimize stack usage:
  - Keep functions small (few variables, less spills) minimize the number of ‘live’ variables (i.e., those which contain useful data at each point in the function)
  - Avoid using large local structures or arrays (use malloc/free instead)
  - Avoid recursion

Software Quality Measurement

- Memory requirement
  - Data type: volatile (RAM), non-volatile (ROM)
  - Memory performance: access speed, data width, size, and range
- Performance benchmarking
  - Harvard core
    - D-cycles, ID-cycles, I-cycles
  - Von Newman cores
    - N-cycles, S-cycles, I-cycles, C-cycles
  - Clock rate
    - Processor, external bus
  - Cache efficiency
    - Average memory access time = hit time + miss rate x miss penalty
    - Cache efficiency = core-cycles / total bus cycles

Global Data Issues

- When declaring global variables in source code to be compiled with ARM software, three things are affected by the way you structure your code:
  - How much **space the variables occupy at run time.** This determines the **size of RAM** required for a program to run. The ARM compilers may insert padding bytes between variables, to ensure that they are properly aligned.
  - How much **space the variables occupy in the image**. This is one of the factors determining the **size of ROM** needed to hold a program. Some global variables which are not explicitly initialized in your program may nevertheless have their initial value (of zero, as defined by the C standard) stored in the image.
  - The **size of the code needed to access the variables**. Some data organizations require more code to access the data. As an extreme example, the smallest data size would be achieved if all variables were stored in suitably sized bit fields, but the code required to access them would be much larger.

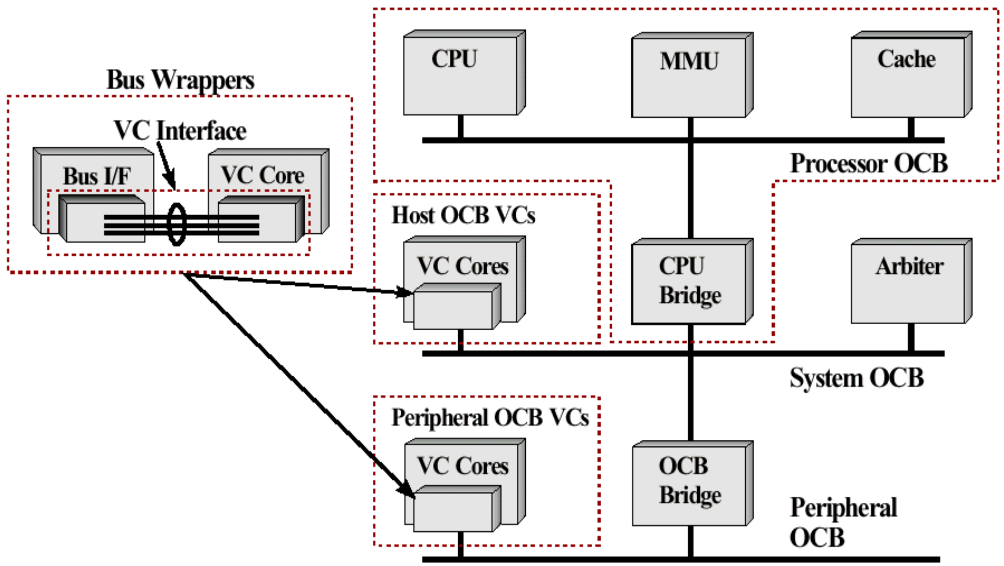
Debugger

- Functionality
  - Execution trace
  - Exam/modify program states
    - Memory
    - Registers (including PC)
  - Control of program execution
    - Run/Halt/Continue/Goto/Stepin
    - Break point: conditional, repeat count
- Issue: debug optimized code in source

Concept of the Bus

- A group of lines shared for interconnection of the functional modules by a standard interface
  - E.g., ARM AMBA and IBM CoreConnect
- Interconnection structure
  - Point-to-point
  - On-chip bus
  - On-chip network
    - Network on Silicon
    - Network on Chip

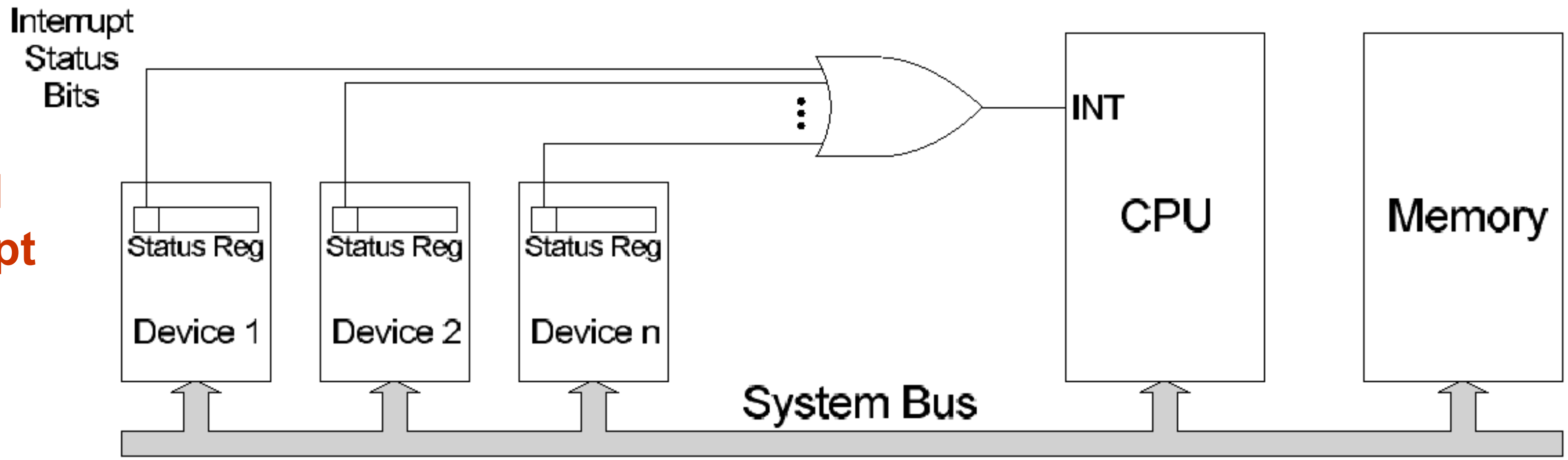
Bus Hierarchy



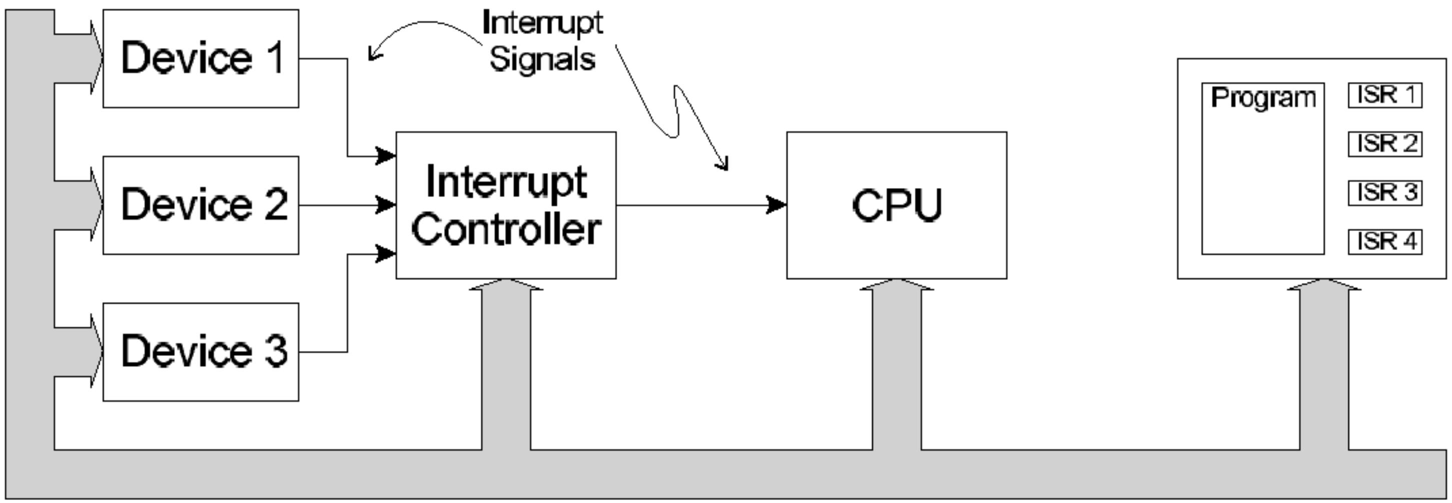
- The structure of multiple buses within a system, organized by bandwidth.
- Local processor bus
  - High processor-specific
  - Processor, cache, MMU, coprocessor
- System bus (backbone)
  - RISC processor, DSP, DMA (masters)
  - Memory, high resolution LCD peripheral
- Peripheral bus
  - Components with other design considerations (power, gate count, etc.)
  - Bridge is the only bus master

Core Peripherals: Interrupt Schemes

Polled Interrupt



Vectored Interrupt



Real Time OS

- A RTOS is an abstraction from hardware and software programming
  - Shorter development time
  - Less porting efforts
  - Better reusability
- Choosing a RTOS is important
  - High efforts when porting to a different OS
  - The chosen OS may have a high impact on the amount of resources needed

RTOS: Functionalities

- ❑ Interrupt service
- ❑ Process (task) management
  - – Scheduler
  - – Synchronization mechanism
    - Inter-process communication (IPC)
    - Semaphores
- ❑ Memory management
- ❑ Service routine
- ❑ Device driver
- ❑ Protection

Characteristics of a RTOS

- Multitasking
  - Non-preemptive vs. preemptive
  - Priority scheduling
- Real-time
  - Soft and hard real time requirements
- Speedy interrupt response
- Frequent Interrupts

Memory Controller

- The International Technology Roadmap for Semiconductors (ITRS) shows memory already accounting for over 50% of a typical SoC, growing to 94% by the year 2014.
- Memory design
  - Size, ports, device number, and memory hierarchy
  - Application-specific behavior
- Memory power management

Outline

- ❑ Introduction to SoC
- ❑ ARM-based SoC and Development Tools
- ❑ SoC Labs
- ❑ ***Available Lab modules in this course***
- ❑ Summary

Lab 1: Code Development

- Goal
  - How to create an application using ARM Developer Suite (ADS)
  - How to change between ARM state and Thumb state when writing code for different instruction sets
- Principles
  - Processor's organization
  - ARM/Thumb Procedure Call Standard (ATPCS)
- Guidance
  - Flow diagram of this Lab
  - Preconfigured project stationery files
- Steps
  - Basic software development (tool chain) flow
  - ARM/Thumb Interworking
- Requirements and Exercises
  - See next slide
- Discussion
  - The advantages and disadvantages of ARM and Thumb instruction sets.

Lab 1: Code Development (cont')

- ARM/Thumb Interworking
  - Exercise 1: C/C++ for "Hello" program
    - Caller: Thumb
    - Callee: ARM
  - Exercise 2: Assembly for "SWAP" program, w/wo veneers
    - Caller: Thumb
    - Callee: ARM
  - Exercise 3: Mixed language for "SWAP" program, ATPCS for parameters passing
    - Caller: Thumb in Assembly
    - Callee: ARM in C/C++

Lab 2: Debugging and Evaluation

- Goal
  - A variety of debugging tasks and software quality evaluation
    - Debugging skills
      - Set breakpoints and watchpoints
      - Locate, examine and change the contents of variables, registers and memory
    - Skills to evaluate software quality
      - Memory requirement of the program
      - Profiling: Build up a picture of the percentage of time spent in each procedure.
      - Evaluate software performance prior to implement on hardware
  - Thought in this Lab the debugger target is ARMulator, but the skills can be applied to Multi-ICE/Angel with the ARM development board(s).

Lab 2: Debugging and Evaluation (cont')

- Principles
  - The Dhrystone Benchmark
  - CPU's organization
- Guidance
  - Steps only
- Steps
  - Debugging skills
  - Memory requirement and Profiling
  - Virtual prototyping
  - Efficient C programming
- Requirements and Exercises
  - Optimize 8x8 inverse discrete cosine transform (IDCT) [1] according to ARM's architecture.
  - Deliverables
- Discussion
  - Explain the approaches you apply to minimize the code size and enhance the performance of the lotto program according to ARM's architecture.
  - Select or modify the algorithms of the code segments used in your program to fit to ARM's architecture.

Lab 3: Core Peripherals and Std. I/O

- Goal
  - Understand the HW/SW coordination
    - Memory-mapped device
    - Operation mechanism of polling and Timer/Interrupt
    - HAL
  - Understand available resource of ARM Integrator
    - semihosting
- Principles
  - Semihosting
  - Interrupt handler
  - Architecture of Timer and Interrupter controller
- Guidance
  - Introduction to Important functions used in interrupt handler
- Steps
  - The same to that of code development
- Requirements and Exercises
  - Use timer to count the total data transfer time of several data references to SSRAM and SDRAM.
- Discussion
  - Compare the performance between using SSRAM and SDRAM.

Lab 4: On-Chip Bus and VCI

- Goal
  - To introduce the interface design conceptually. Study the communication between FPGA on logic module and ARM processor on core module. We will introduce the ARMB in detail.
- Principle
  - Overview of the AMBA specification
  - Introducing the AMBA AHB
  - AMBA AHB signal list
  - The Arm-based system overview
- Guide
  - We use a simple program to lead student understanding the ARMB.
- Requirements and Exercises
  - To trace the hardware code and software code, indicate that software how to communicate with hardware using the ARMB interface.
- Discussion
  - If we want to design an accumulator (1,2,3…) , how could you do to implement it using the scratch code?
  - If we want to design a hardware using FPGA, how could you do to add your code to the scratch code and debugger it ?
  - To study the ARMB bus standard, try to design a simple ARMB interface.

Lab 5: Virtual Prototyping

- Goal
  - To introduce the prototyping using high-level hardware model. Study the communication and synchronization between hardware and software using ARM's ARMulator.
- Principle
  - ARMulator hardware model
  - Synchronization between hardware and software
  - Memory-mapped hardware control interface
- Guide
  - We use a simple example to help student understand how to virtual prototype usingARMulator.
- Requirements and Exercises
  - To trace the C/C++ code of the hardware model and the software code.
  - Understand the synchronization and communication scheme.
  - Being able to write simple drviers.
- Discussion
  - Compare the advantages/disadvantages of the synchronization schemes.

Lab 6: Memory Controller

- Goal
  - Realize the principle of memory map and internal and external memory
- Principles
  - System memory map
  - Core Module Control Register
  - Core Module Memory Map
- Guidance
  - We use a simple program to lead student understanding the memory.
- Requirements and Exercises
  - Modify the memory usage example. Use timer to count the total access time of several data accessing the SSRAM and SDRAM. Compare the performance between using SSRAM and SDRAM.
- Discussion
  - Discuss the following items about Flash, RAM, and ROM.
    - Speed
    - Capacity
    - Internal /External

Lab 7: ASIC Logic

- Goal
  - HW/SW Co-verification using Rapid Prototyping
- Principles
  - Basics and work flow for prototyping with ARM Integrator
  - Target platform: AMBA AHB sub-system
- Guidance
  - Overview of examples used in the Steps
- Steps
  - Understand the files for the example designs and FPGA tool
  - Steps for synthesis with Xilinx Foundation 3.1i
- Requirements and Exercises
  - RGB-to-YUV converting hardware module
- Discussion
  - In example 1, explain the differences between the Flash version and the FPGA one.
  - In example 1, explain how to move data from DRAM to registers in MYIP and how program access these registers.
  - In example2, draw the interconnect among the functional units and explain the relationships of those interconnect and functional units in AHB sub-system
  - Compare the differences of polling and interrupt mechanism

Lab 8: Real-Time OS

- Goal
  - A guide to use RTOS and port programs to it
- Principles
  - Basic concepts and capabilities of RTOS
    - Task, task scheduling & context switch
    - Resource management using Semaphore
    - Inter-process communication using Mailbox
    - Memory management
  - Coding guideline for a program running on the embedded RTOS
  - Setting up the ARMulator
- Guidance
- Steps
  - Building μC/OS-II
  - Building Program with μC/OS-II
  - Porting Program to μC/OS-II
- Requirements and Exercises
  - Write an embedded software for ID checking engine and a front–end interface
- Discussion
  - What are the advantages and disadvantages of using RTOS in SoC design?

Lab 9: Case design

- Goal
  - Study how to use the ARM-based platform to implement JPEG system. In this chapter, we will describe the JPEG algorithm in detail.
- Principle
  - Detail of design method and corresponding algorithm
- Guidance
  - In this section, we will introduce the JPEG software file (.cpp) in detail. We will introduce the hardware module.
- Steps
  - We divide our program into two parts:
    - Hardware
    - Software
- Requirements and Exercises
  - Try to understand the communication between the software part and hardware part. To check the computing result is correct. You can easily check that the output value from the FPGA on LM

Lab 9: Case design (cont')

- Discuss
  - We describe the decoder part algorithm on reference 3, try to implement it on ARM-based platform. You can divide to two parts: software & hardware.

Summary

- To build SoC labs
  - Software tools
    - Code development\debug\evaluation (e.g. ARM Developer Suite)
    - Cell-based design EDA tools
  - Development boards, e.g., ARM Integrator
    - Core Module: 7TDMI, 720T, 920T, etc
    - Logic Module (Xilin XCV2000E, Altera LM-EP20K1000E)
    - ASIC Development Platform (Integrator/AP AHB)
    - Multi-ICE Interface
  - Advanced labs: RTOS (e.g., μC/OS-II) Verification/Validation

Summary

- The ARM has played a leading role in the opening of this era since its very small core size leaves more silicon resources available for the rest of the system functions.
- SoC labs are challenges to universities
  - Various expertise
  - Tight schedule for (M.S.) students

Reference

[1] http://twins.ee.nctu.edu.tw/courses/ip_core_02/index.html

[2] **ARM System-on-Chip Architecture** by S.Furber, Addison Wesley Longman: ISBN 0-201-67519-6.

[3] http://www.arm.com