Outline

- ARM-based SoC and Development Tools
- SoC Labs
- Available Lab modules in NCTU
- Summary
SoC Example

Institute of Electronics, National Chiao Tung University

ARM-Based SoC Design Laboratory Course

SoC Example

- Memory and Expansion Control Module incl. PCMCIA
- HW Acceleration
- VGA Controller
- LCD Controller
- APB Bridge DMA Contr
- UART#1 (16C550)
- UART#2 (16C550)
- SPI
- MMC
- Joystick Contr
- Digitizer
- Pen
- RS232
- JTAG Debug PLL & OSC
- SDRAM ROM FLASH PCcard CF
- RTC Timer GPIO IRQ Contr Power Man
- Write buffer
- ARM Core
- Cache Contr
- Cache
- MMU
- On chip Memory SRAM, ROM FLASH EEROM
- Memory X
- Memory Y
- DSP Core
- 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
- ASB

RGB Monitor
LCD, Mono/Colour (STN/TFT)
ARM-based System Development

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

### Embedded Cores

<table>
<thead>
<tr>
<th>Core</th>
<th>Cache Size (Inst/Data)</th>
<th>Tightly Coupled Memory</th>
<th>Memory Management</th>
<th>AHB Bus Interface</th>
<th>Thumb</th>
<th>DSP</th>
<th>Jazelle</th>
<th>Clock MHz **</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM7TDMI</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes*</td>
<td>No</td>
<td>No</td>
<td>133</td>
</tr>
<tr>
<td>ARM7TDMI-S</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>100-133</td>
</tr>
<tr>
<td>ARM7EJ-S</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>100-133</td>
</tr>
<tr>
<td>ARM966E-S</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes*</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>230-250</td>
</tr>
<tr>
<td>ARM940T</td>
<td>4K/4K</td>
<td>No</td>
<td>MPU</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>250</td>
</tr>
<tr>
<td>ARM946E-S</td>
<td>Variable</td>
<td>Yes</td>
<td>MPU</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>180-210</td>
</tr>
<tr>
<td>ARM1026EJ-S</td>
<td>Variable</td>
<td>Yes</td>
<td>MMU+MPU</td>
<td>dual AHB</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>266-325</td>
</tr>
</tbody>
</table>

### Platform Cores

<table>
<thead>
<tr>
<th>Core</th>
<th>Cache Size</th>
<th>Tightly Coupled Memory</th>
<th>Memory Management</th>
<th>AHB Bus Interface</th>
<th>Thumb</th>
<th>DSP</th>
<th>Jazelle</th>
<th>Clock MHz **</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM720T</td>
<td>8K unified</td>
<td>No</td>
<td>MMU</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>100</td>
</tr>
<tr>
<td>ARM920T</td>
<td>16K/16K</td>
<td>No</td>
<td>MMU</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>250</td>
</tr>
<tr>
<td>ARM922T</td>
<td>8K/8K</td>
<td>No</td>
<td>MMU</td>
<td>Yes*</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>250</td>
</tr>
<tr>
<td>ARM926EJ-S</td>
<td>Variable</td>
<td>Yes</td>
<td>MMU</td>
<td>dual AHB</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>220-250</td>
</tr>
<tr>
<td>ARM1020E</td>
<td>32K/32K</td>
<td>No</td>
<td>MMU</td>
<td>dual AHB</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>325</td>
</tr>
<tr>
<td>ARM1022E</td>
<td>16K/16K</td>
<td>No</td>
<td>MMU</td>
<td>dual AHB</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>325</td>
</tr>
</tbody>
</table>

### Secure Applications

<table>
<thead>
<tr>
<th>Core</th>
<th>Cache Size</th>
<th>Tightly Coupled Memory</th>
<th>Memory Management</th>
<th>AHB Bus Interface</th>
<th>Thumb</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC100</td>
<td>No</td>
<td>No</td>
<td>MPU</td>
<td>No</td>
<td>Yes</td>
<td>~500+</td>
</tr>
<tr>
<td>SC110</td>
<td>No</td>
<td>No</td>
<td>MPU</td>
<td>No</td>
<td>Yes</td>
<td>0</td>
</tr>
<tr>
<td>SC200</td>
<td>Optional</td>
<td>No</td>
<td>MPU</td>
<td>No</td>
<td>Yes</td>
<td>120K</td>
</tr>
<tr>
<td>SC210</td>
<td>Optional</td>
<td>No</td>
<td>MPU</td>
<td>No</td>
<td>Yes</td>
<td>120K</td>
</tr>
</tbody>
</table>

### Intel ARM-based Processors

<table>
<thead>
<tr>
<th>Core</th>
<th>Cache Size</th>
<th>Tightly Coupled Memory</th>
<th>Memory Management</th>
<th>AHB Bus Interface</th>
<th>Thumb</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrongARM</td>
<td>16K/8K</td>
<td>No</td>
<td>MMU</td>
<td>N/A</td>
<td>No</td>
<td>&lt;0.4</td>
</tr>
<tr>
<td>Intel XScale</td>
<td>32K/32K</td>
<td>No</td>
<td>MMU</td>
<td>N/A</td>
<td>Yes</td>
<td>400</td>
</tr>
</tbody>
</table>

---

**Note:** Frequency range for soft (synthesizable) cores is dependent upon cell library and synthesis flow.

**UPDATE:** 2002.5.5

---

**Frequency information:**
- 350 - 500+ MHz (0.13µ w/c)
- 0.4 mW/MHz (0.13µ 1.2v)
- ~120K gates equivalent (integer core)
- 400 - 570+ DMIPS (0.13µ w/c)

**CPI information:**
- CPI maintained from ARM9™ family

---

**Intel ARM-based processors:**
- Intel process

---

**SecurCore family:**
- 0.18µm silicon foundry generic process, worst case: 1.62V, 125C, slow silicon

---

**Intel ARM-based processors:**
- Intel process

---

**Update information:**
- 2002.5.5

---

**Notes on ARM Cores:**
- Frequency 1.4x previous cores
- 500+ MHz (0.13µ w/c)
- Power
- < 0.4 mW/MHz (0.13µ 1.2v)
- Area < 1.5x previous cores
- ~120K gates equivalent
- CPI maintained from ARM9™ family
# ARM Architecture Version

<table>
<thead>
<tr>
<th>Core</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM1</td>
<td>v1</td>
</tr>
<tr>
<td>ARM2, ARM2as, ARM3</td>
<td>v2</td>
</tr>
<tr>
<td>ARM6, ARM60, ARM610, ARM7, ARM710, ARM7D ARM7DI</td>
<td>v3</td>
</tr>
<tr>
<td>ARM7TDMI, ARM710T, ARM720T, ARM740T</td>
<td>v4T</td>
</tr>
<tr>
<td>StrongARM, ARM8, ARM810</td>
<td>v4</td>
</tr>
<tr>
<td>ARM9TDMI, ARM920T, ARM940T</td>
<td>V4T</td>
</tr>
<tr>
<td>ARM9E-S, ARM10TDMI, ARM1020E</td>
<td>v5TE</td>
</tr>
<tr>
<td>ARM7EJ-S, ARM926EJ-S, ARM1026EJ-S</td>
<td>v5TEJ</td>
</tr>
<tr>
<td>ARM11</td>
<td>v6</td>
</tr>
</tbody>
</table>

[Diagram showing the timeline and core evolution from 1994 to 2006.]
ARM Coprocessors

- **VFP**
  - Optional part of microarchitecture
    - No overhead for markets that do not need floating point
  - A tightly-integrated coprocessor
    - Enables maximum advantage of separate load/store and execution pipelines
  - 8-Stage FMAC pipeline

- **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

Mixed implementation of AHB and AHB-Lite in a multi-layer system.
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 PrimeXsys Wireless Platform

Ingredients

- Hardware building block
- OS-Ports
- Tool Support and Validation Methodology
Example: GPRS Phone
Example: Videophone
## Size and Speed

<table>
<thead>
<tr>
<th>Building Block</th>
<th>Gates</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM926EJ-S *</td>
<td>230k</td>
</tr>
<tr>
<td>DMA Controller</td>
<td>82k</td>
</tr>
<tr>
<td>SDRAM Controller</td>
<td>55k</td>
</tr>
<tr>
<td>Medium ETM</td>
<td>37k</td>
</tr>
<tr>
<td>CLCD **</td>
<td>24k</td>
</tr>
<tr>
<td>MOVE ***</td>
<td>13k</td>
</tr>
<tr>
<td>Static Memory Interface</td>
<td>13k</td>
</tr>
<tr>
<td>VIC</td>
<td>13k</td>
</tr>
<tr>
<td>SIM Card</td>
<td>12k</td>
</tr>
<tr>
<td>UART</td>
<td>9k</td>
</tr>
<tr>
<td>Multi-layer AHB</td>
<td>8k</td>
</tr>
<tr>
<td>SSP</td>
<td>8k</td>
</tr>
<tr>
<td>GPIO (1.6k x 4)</td>
<td>6k</td>
</tr>
<tr>
<td>Timers</td>
<td>6k</td>
</tr>
<tr>
<td>RTC</td>
<td>3k</td>
</tr>
<tr>
<td>Watchdog</td>
<td>2k</td>
</tr>
<tr>
<td>System Controller</td>
<td>2k</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>523k</td>
</tr>
</tbody>
</table>

Assuming

\[ 60k \text{ gates} \approx 1\text{mm}^2 \]

@0.18um TSMC, Libra VISA cell library

\[ 523k \text{ gates} \approx 8.7\text{mm}^2 \]

@0.18um process

ARM926EJ-S & MOVE @ 200+ MHz

The bus and peripherals @ 100+MHz.

* Excludes caches and TCM which are variable.
** Excludes palette RAM and FIFO implemented as D-types
*** Includes 64 bytes of working RAM (implemented as registers)
Tools Support and Validation Methodology

AMBA Compliance Testbench (ACT™)
Integration Validation Testbench
- Testing with a Bus Functional Model
- Testing with a Design Simulation Model
- Testing Device Driver Initialisation
Subsystem Validation Testbench
Software Development Model
IP Development Board

Test
- Register maps
- Address map
- All blocks are correctly connected onto the buses
- Interblock connections are correct
- All peripherals are correctly connected to the outside world
AMBA Compliance Testbench (ACT)

- Each module are tested separately to ensure the portability to any other AMBA designs.
- Six types of DUT
  - AHB Master, AHB Slave, APB Master, APB Slave, Arbiter, Decoder
Integration Validation Testbench

Testing with a Design Simulation Model

• Prove that the core has been successful integrated and can be booted on the extended system.

• Is validated through the boot sequence. Additional code running in conjunction with the boundary scan test blocks and trace monitor components verify correct integration of ICE and ETM.

• The vector sequences which were run on the BFM models are regenerated as C function calls to ensure consistency.
Testing Device Driver Initialization

<table>
<thead>
<tr>
<th></th>
<th>Integration TestBench with DSM</th>
<th>Software Development Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot Code</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Register Integrity</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IRQ Foldback</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Peripheral Connect</td>
<td>✓</td>
<td>X (see note 2)</td>
</tr>
<tr>
<td>Driver Initialisation</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Device Driver Functionality</td>
<td>X (see note 1)</td>
<td>✓</td>
</tr>
</tbody>
</table>

1. It is too slow (less than 100 cycles/sec) to efficiently test the full device driver functionality. The SDM is a better tool for doing this as it runs at 1-10MHz.
2. Peripheral connection software has no meaning in the abstracted SDM. Future versions of the SDM will allow this testing.

ARM recommends that developers designing time-critical device drivers replace the DSM with a Design Development Model (DDM) of the ARM926EJ-S. These provide faster cycle accurate simulations necessary for the purpose of validating critical driver performance.
Subsystem Validation Testbench

xVC: not only simulates ‘good’ behavior but also emulates fault conditions

Test coverage monitor
Subsystem coverage monitor
subsystem annotation block
AMBA protocol block

Verisity Specman
Software Development Model

All of the peripherals including AHB masters and slaves and the APB peripherals have been modeled in C.
The multi-layer fabric and the DMA will be modeled at a cycle count level: there will be a small difference (10%) in the cycle count.

The SDM will be used for:
- Writing device drivers
- Porting the OS (Symbian OS, Windows CE and Linux)
- Porting and developing application software, including Java applications
- Characterizing the performance of the software on the ARM926EJ-S & the PWP platform. This is especially useful when trying to determine how much cache, TCM and on chip memory is required.
ARM's Extending IP - Application Software

- Application Software
- Jazelle Technology Enabling Kit (JTEK™)
- MOVE Technology
- Audio Software
- PrimeCell® Peripherals

Symbian OS Quartz DFRD
- Agenda
- Calculator
- Connect
- Contacts
- Jotter
- To do
- Web browser

Windows CE Pocket PC
- Internet Explorer
- Pocket Excel
- Pocket Outlook
- Pocket Word
- Reader
- Windows Media Player

<table>
<thead>
<tr>
<th>Description</th>
<th>AMBA bus type</th>
<th>Description</th>
<th>AMBA bus type</th>
</tr>
</thead>
<tbody>
<tr>
<td>UART</td>
<td>APB</td>
<td>Advanced Audio Codec Interface</td>
<td>APB</td>
</tr>
<tr>
<td>Similar to 16C550</td>
<td></td>
<td>Includes software device driver</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supporting AC97</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Includes software device driver</td>
<td></td>
</tr>
<tr>
<td>SD-Card Interface</td>
<td>APB</td>
<td>Multimedia Card Interface</td>
<td>APB</td>
</tr>
<tr>
<td>Includes physical layer device</td>
<td></td>
<td>Security software stack also</td>
<td></td>
</tr>
<tr>
<td>driver</td>
<td></td>
<td>available</td>
<td></td>
</tr>
<tr>
<td>G.723.1</td>
<td>TBA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Codec Code</th>
<th>Availability</th>
<th>Clock Freq in MHz</th>
<th>Memory Footprint, no DRM, (kB)</th>
<th>SNR cf to Ref Codec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ARM926</td>
<td>ROM</td>
<td>RAM</td>
</tr>
<tr>
<td>MP3 Decode XP</td>
<td>NOW</td>
<td>16</td>
<td>29</td>
<td>22</td>
</tr>
<tr>
<td>MPEG-AAC (TNS)</td>
<td>NOW</td>
<td>15 (16)</td>
<td>39</td>
<td>22</td>
</tr>
<tr>
<td>MS-WMA</td>
<td>NOW via MS</td>
<td>23</td>
<td>75</td>
<td>26</td>
</tr>
<tr>
<td>MP3 Encode</td>
<td>NOW</td>
<td>32</td>
<td>63</td>
<td>38</td>
</tr>
<tr>
<td>AAC Encode (TNS)</td>
<td>4Q01</td>
<td>&lt;33 (&lt; 50)</td>
<td>~64</td>
<td>~27</td>
</tr>
<tr>
<td>ADPCM</td>
<td>NOW</td>
<td>&lt;5</td>
<td>&lt;1</td>
<td>3.5</td>
</tr>
<tr>
<td>G.723.1</td>
<td>TBA</td>
<td>45</td>
<td>123</td>
<td>20</td>
</tr>
</tbody>
</table>
## PrimeCell Library

<table>
<thead>
<tr>
<th>Part #</th>
<th>Description</th>
<th>Approx. gate count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Main system (AHB) peripherals</strong></td>
<td></td>
</tr>
<tr>
<td>PL090</td>
<td>Static Memory Controller: SRAM, Flash &amp; ROM</td>
<td>10.5 k</td>
</tr>
<tr>
<td>PL110</td>
<td>Color LCD Controller: Color and Mono with gray scale, Supports TFT and STN, Single and Dual Panel</td>
<td>26 k</td>
</tr>
<tr>
<td>PL170</td>
<td>SDRAM controller</td>
<td>-</td>
</tr>
<tr>
<td>PL170</td>
<td>External Bus Interface (EBI)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td><strong>Ancillary (APB) peripherals</strong></td>
<td></td>
</tr>
<tr>
<td>PL010</td>
<td>UART w/ IrDA SIR: Similar to 16C550, 16Byte FIFO, up to 115K2 bits/s</td>
<td>7.7 k</td>
</tr>
<tr>
<td>PL021</td>
<td>Synchronous Serial I/F (Master &amp; Slave): Supports Motorola SPI, TI SSI, Microwire</td>
<td>8.0 k</td>
</tr>
<tr>
<td>PL030</td>
<td>Real Time Clock: 32 Bit Counter, Match Reg, Requires 1Hz Ck</td>
<td>2.8 k</td>
</tr>
<tr>
<td>PL050</td>
<td>Keyboard and Mouse Interface: PS/2 Compatible</td>
<td>2.0 k</td>
</tr>
<tr>
<td>PL060</td>
<td>General Purpose IO: 2x8bit</td>
<td>0.8 k</td>
</tr>
<tr>
<td>PL160</td>
<td>DC to DC Converter Interface: 1.8MHz, 900, 225, 96 kHz Prog O/P</td>
<td>1.4 k</td>
</tr>
<tr>
<td>PL130</td>
<td>Smartcard Interface: Compliant with the EMV Standard and ISO 7816-3</td>
<td>12.3 k</td>
</tr>
</tbody>
</table>

*Gate count, NAND-2 equivalent without scan. Target Cell Library: Avant! Corporation Passport CB25 v2.1 (0.25µm)*
Third Party Partnership Program

- **Hardware IP Criteria:**
  - AMBA bus-compatible
  - Suitable for inclusion into a PWP platform-based ASIC
  - **Device driver meets the software IP criteria**
  - Can be successfully run on the PWP IP Development Board
  - Has been measured for a required set of characteristics:
    - Bus bandwidth
    - Power consumption
    - Gate count

- **Software IP Criteria:**
  - Is ported to at least one of the OS that come with the PWP platform
  - Must work on the IP Development Board (and therefore the PWP platform-based ASIC)
  - Has been measured for a required set of characteristics:
    - Performance (MHz)
    - RAM/ROM size
    - Bus bandwidth
    - Power consumption

› ARM will help developers to ensure these criteria.
## OS-Ports

<table>
<thead>
<tr>
<th>OS</th>
<th>Version</th>
<th>Available on</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>SymbianOS</td>
<td>6.1</td>
<td>PWP Q3-2001</td>
<td>The customer must first be a member of Symbian’s Semiconductor Program.</td>
</tr>
<tr>
<td>WinCE</td>
<td>4.0</td>
<td>Q4-2001</td>
<td>The customer must first join the Windows Consortium.</td>
</tr>
<tr>
<td>Linux</td>
<td>2.4x</td>
<td>Q4-2001</td>
<td>Each provider may have a different business model.</td>
</tr>
<tr>
<td>PalmOS</td>
<td>TBA</td>
<td>TBA</td>
<td></td>
</tr>
</tbody>
</table>

As long as the customer continues to pay support and maintenance, ARM will keep the OS-ports updated.
ARM-based System Development

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

- 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
- 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

- C/C++ source
- C libraries
- asm source
- C compiler
- assembler
- ELF object file
- With DWARF2 debug tables
- linker
- .axf
- ELF/DWARF2 image
- ARMulator
- System models
- Librarian
- object libraries
- ARMsd
- development board
- Dwarf: Debug With Arbitrary Record Format
- ELF: Executable and Linking Format
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
  - processor core models (ISS) which can emulate ARM cores
  - a memory interface (cache, physical memory) which allows the characteristics of the target memory system to be modeled and customized
  - a coprocessor interface that supports custom coprocessor models
  - an OS interface that allows individual system calls to be handled
  - Simulation of exceptions, such as interrupts and aborts, and MMU
  - User extensible to add support for custom peripherals

- Can be operates at various levels of accuracy
  - instruction accurate
  - cycle accurate
  - timing accurate

- Benchmarking before hardware is available
  - instruction count or number of cycles can be measured for a program
  - performance analysis
  - Once the design is OK,
    - hardware ➔ design or synthesis by CAD
    - software ➔ still use ARMulator model, but instruction accurate
ARM Emulator: ARMulator (2/2)

- 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
  - Access shielding
  - Register shielding
  - Functional shielding

- 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

Diagram:

- User application
- AFS utilities
- C and C++ libraries
- AFS board-specific µHAL routines
- AFS support routines
- Development board

General

Specific
µHAL Examples

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

• Examples
  
  // Install new trap handlers and soft vectors
  uHALr_InitIntInterrupts();
  
  // initialise the timers
  uHALr_InitTimers();
  
  // initialise the tick count
  OSTick = 0;
  uHALr_printf("Timer init\n");
  
  if (uHALr_RequestSystemTimer(TickTimer, (const unsigned char*)"test") <= 0)
    uHALr_printf("Timer/IRQ busy\n");
  
  // Start system timer & enable the interrupt.
  uHALr_InstallSystemTimer();
Reference Peripheral Specification

- The reference peripheral specification (RPS) defines a basic set of components, providing a framework within which an operating system can run but leaving full scope for application-specific system
  - a memory map
    - The base address (e.g., ICBase for the interrupt controller)
    - All the address of the registers are relative to one of the base addresses
  - an interrupt control
    - Enabling, disabling (by mask) and examining the status of up to 32 level-sensitive IRQ sources and one FIQ source
  - a counter timer
    - Two 16-bit counter-timers, controlled by registers. The counters operate from the system clock with selectable pre-scaling
  - Reset and pause controller includes some registers
    - The readable registers give identification and reset status information
    - The writable registers can set or clear the reset status, clear the reset map and put the system into pause mode where it uses minimal power until an interrupt wakes it up again
ARM Symbolic Debugger (ARMsd)

- ARMsd: ARM and Thumb symbolic debugger
  - can single-step through C or assembly language sources,
  - set break-points and watch-points, 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)

- 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
To debug your application you must choose:

- a **debugging system**, that can be either:
  
  • hardware-based on an ARM core
  
  • software that simulates an ARM core.

- a **debugger**, such as AXD, ADW, ADU, or armd.

![Diagram showing Debugger-Target Interface]

**ARM Debugger**

- AXD

Remote Debug Interface (RDI)

<table>
<thead>
<tr>
<th>Target (software)</th>
<th>Target (hardware)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARMulator</td>
<td>Multi-ICE</td>
</tr>
<tr>
<td>RDI</td>
<td>RDI</td>
</tr>
<tr>
<td>Target emulated in software</td>
<td>ARM development board</td>
</tr>
<tr>
<td></td>
<td>Remote_A</td>
</tr>
<tr>
<td></td>
<td>ARM development board</td>
</tr>
</tbody>
</table>
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
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 Desktop

Menu

Toolbar

Control System view

Variable processor view

Watch processor view

Watch system view

Disassembly processor view

Source processor view

Console processor view

Status bar
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
In debug state, the core’s internal state and the system’s external state may be examined. Once examination is complete, 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 watch-point units, together with a control and status register
- Either watch-point unit can be configured to be a watch-point or a breakpoint
Available Design Examples and Tools

- **Design Examples**
  - ARM Micropack
  - AMBA Design Kit (ADK)
  - Green AMBA

- **Verification Tools**
  - ARM AMBA Compliance Testbench (ACT)
  - Verisity AMBA eVC
  - Cadence AMBA Transaction Verification Modules
  - Synopsys AMBA Verification IP
EASY Environment

TB Easy: Normal operation, as in real design

TB Tic: Test mode operation, test individual blocks
AMBA Design Kit (ADK)

FRBM-based simple EASY

- Synthesis script & timing scripts
- Example C/Assembler software, e.g., initializes the system and tests modules

ARM7TDMI core-based single master EASY

ARM922T core-based multi-layer EASY
ARM Modeling

System model
Instruction set simulators (ISS)
Co-verification model
Bus Interface model
Behavioral/RTL model
Design signoff models

Hardware modeling
Gate Level netlist model

Concept

Silicon

Efficiency

Accuracy
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

ATX motherboard
ARM Integrator within a ATX PC Case
Inside the Case
Logic Module

- Multi-ICE
- Config PLD
- Flash
- Xchecker/Download
- ZBT SSRAM
- CSR
- APB IP
- IntCntl
- AHB/APB bridge
- AHB IP
- AHB SSRM controller
- Prototyping grid (16x17)
- EXPI/M connector
- EXPA/EXPB connector
- FPGA
- LEDs
- Switchs
- Oscs
- Trace
- Push B
- LA C
- LM
Extension with Prototyping Grid

You can use the prototyping grid to:
- wire to off-board circuitry
- mount connectors
- mount small components
## System Memory Map

### ROM / RAM and peripherals
- **LM**
  - **CM alias memory**
  - **PCI**
  - **ROM / RAM and peripherals**
  - **Reserved**
  - **EBI**
  - **Peripheral regs**
  - **CM 0, 1, 2, 3**

### Memory Map
- **ROM**
  - Address: 0x0000_0000
  - Size: 256MB
  - **CM alias memory**
    - Address: 0x0C00_0000
    - Size: 1GB
  - **PCI**
    - Address: 0x0400_0000
    - Size: 1GB
  - **ROM / RAM and peripherals**
    - Address: 0x0800_0000
    - Size: 2GB
    - **Reserved**
      - Address: 0x0A00_0000
      - Size: 3GB
    - **EBI**
      - Address: 0x0C00_0000
      - Size: 4GB
    - **Peripheral regs**
      - Address: 0x0E00_0000
      - Size: 1GB
    - **CM 0, 1, 2, 3**
      - Address: 0x1000_0000
      - Size: 1GB

### RAM
- **SDRAM**
  - **CM 0**
    - Address: 0x0F00_0000
    - Size: 256MB
  - **CM 1**
    - Address: 0x1000_0000
    - Size: 256MB
  - **CM 2**
    - Address: 0x1100_0000
    - Size: 256MB
  - **CM 3**
    - Address: 0x1200_0000
    - Size: 256MB

### RAM Size
- **256MB**
- **512MB**
- **768MB**
- **1GB**

### External Memory
- **0x0FFF_FFFF**
- **0x0000_0000**
- **0x1000_0000**
- **0x2000_0000**
- **0x3000_0000**

### Other Resources
- **GPIO**
- **LED/Switch**
- **Mouse**
- **Keyboard**
- **UART 0**
- **UART 1**
- **RTC**
- **Int control**
- **Counter/Timer**
- **EBI regs**
- **Sys control**
- **CM regs**

### Boot ROM
- **SSRAM**
- **Flash**

### External Memory
- **Boot ROM**
- **0x0C00_0000**
- **0xD000_0000**
- **0xE000_0000**
- **0xF000_0000**

### External Memory Size
- **64MB**
- **128MB**
- **192MB**
- **256MB**

### External Memory Addresses
- **0x0000_0000**
- **0x1000_0000**
- **0x2000_0000**
- **0x3000_0000**

### Other Resources
- **CS 3 (EXPM)**
- **Spare**
- **SSRAM**
- **Flash**
- **Boot ROM**
Outline

• ARM-based SoC and Development Tools
• SoC Labs
• Available Lab modules in NCTU
• Summary
SoC Challenges

• Bigger circuit size (Size does matter)
  – Design data management, CAD capability
  – Forced to go for high-level abstraction

• Smaller device geometries, new processing (e.g., SOI)
  – Short channel effect, sensitivity, reliability
  – Very different, complicated device model

• Higher density integration
  – Shorter distance between devices and wires: cross-talk coupling

• Low Power requirement
  – Standby leakage power is more significant, lower noise margin

• Higher frequencies
  – Inductance effect, cross talk coupling noise
SoC Challenges (cont.)

- Design Complexity
  - μCs, DSPs, HW/SW, SW protocol stacks, RTOS’s, digital/analog IPs, On-chips buses
  - Use a known real entity
    - A pre-designed component (IP reuse) or A platform (architecture reuse)
  - Partition
    - Hardware and software, based on functionality
  - Modeling
    - Consistent and accurate models at different levels

- IP Reuse

- Verification, at different levels
  - HW/SW co-verification
  - Digital/analog/memory circuit verification
  - Timing, power and signal integrity verification

- Time-to-market
Digital IC design expertise may become commodity (?)
Analog IC design is still an art
-- Summary Conclusion

Source: Signal Integrity Verification in SoC Design, Dr. An-Chang Deng, 2002.7.18
SoC Labs

- Code development
- Debugging and evaluation
- Core peripherals
- Real-time OS (RTOS)
- On-chip bus
- Cache and MMU
- Coprocessing
- Memory controller
- ASIC logic
- Standard I/O
- JTAG and ICE
- Case Designs
- Verification/validation
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
Remainders – Modulo Arithmetic

- The remainder operator ‘%’ is commonly used in modulo arithmetic.
  - This will be expensive if the modulo value is not a power of two
  - This can be avoid by rewriting C code to use if () statement heck

```c
unsigned counter1(unsigned counter)
{
    return (++counter % 60);
}

unsigned counter2(unsigned counter)
{
    if (++counter >= 60)
    {
        counter=0;
        return counter
    }
}
```

```asm
counter1
    STMFB sp!, {lr}
    ADD r1, r0, #1
    MOV r0, #0x3C
    BL __rt_udiv
    MOV r0, r1
    LDMIA sp!, {pc}
```

```asm
counter2
    ADD r0, r0, #1
    CMP r0, #0x3C
    MOVCS r0, #0
    MOV pc, lr
```
Variable Types – Size Examples

```c
int wordinc (int a)
{ return a + 1;
}

short shortinc (short a)
{ return a + 1;
}

char charinc (char a)
{ return a + 1;
}
```
Data Layout

Default

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

<table>
<thead>
<tr>
<th>a</th>
<th>pad</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>pad</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>d</td>
</tr>
</tbody>
</table>

occupies 12 bytes, with 4 bytes of padding

Optimized

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

<table>
<thead>
<tr>
<th>a</th>
<th>c</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>d</td>
</tr>
</tbody>
</table>

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 bitfields, but the code required to access them would be much larger.
Dhrystone Result Example

**Target:**  ARM940T, 4kB I-cache, 4kB D-cache, 10.00MHz core clock, (Physical memory, 3.3MHz)

<table>
<thead>
<tr>
<th></th>
<th>Instructions</th>
<th>Core Cycles</th>
<th>S-cycles</th>
<th>N-cycles</th>
<th>I-cycles</th>
<th>C-cycles</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Iteration 1</td>
<td>306</td>
<td>446</td>
<td>377</td>
<td>0</td>
<td>345</td>
<td>0</td>
<td>722</td>
</tr>
<tr>
<td>Iteration n</td>
<td>306</td>
<td>446</td>
<td>7</td>
<td>0</td>
<td>142</td>
<td>0</td>
<td>149</td>
</tr>
</tbody>
</table>

**Iteration 1:**  \(673 \times 1 / 3,333,333 = 216.6\)us
**Iteration n:**  \(149 \times 1 / 3,333,333 = 44.7\)us
\(446/149 = 2.993\)

**Iteration 1~n:**
- **Total Core Cycles:** 27074407
- **Total Bus Cycles:** 9034428
- **Cache Efficiency:** 2.9979 (MCCFG=3)
- **Cache Efficiency %:** 100 x (Cache Efficiency x MCCFG) = 99.93%

Cached with different clock domains
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, 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
  - highly 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
Differences Between Traditional Bus and OCB

• The root: I/O pins are limited and fixed
• The characteristics of a traditional bus
  – Shared I/O
  – Fixed interconnection scheme
  – Fixed timing requirement
  – Dedicated address decoding

• For a OCB
  – Routing resource in target device (e.g. FPGA, ASIC)
  – Bandwidth and latency are more important
Significance of the OCB

• Long wire problems in deep sub-micro design
  – Wire-First methodology (Dally), ICCAD –2000)
  – IR drop
  – Electromigration
  – Signal integrity

• Design Consideration
  – Bus, clock, power, test, I/O

Source: SoC/IP Design Methodology, Michael Chen, 2002.7.9

• Power consumption in FPGA
  – Interconnect (65%), Clock Power (21%), I/O (9%), CLB (5%)

Source: Low-energy embedded FPGA structures, 1998 International Symposium on Low Power Electronics and Design
Core Peripherals

- Example: SA-1100
  - SA-110 StrongARM CPU
  - DMA, Interrupt controller, Timers, etc.
  - Memory controller for EDO DRAM, ROM, flash memory and SRAM
Core Peripherals: Interrupt Schemes

**Polled Interrupt**
- Interrupt Status Bits
- Device 1
- Device 2
- Device n

**Vectored Interrupt**
- Interrupt Signals
- Device 1
- Device 2
- Device 3
- Interrupt Controller
- CPU
- Program:
- ISR 1
- ISR 2
- ISR 3
- ISR 4
Core Peripherals: DMAs

Channel registers in ARM PrimeCell
- Source address
- Destination address
- Linked list address
- Control
- Channel configuration register

Common registers in ARM PrimeCell
- PrimeCell / Peripheral identification register
- Enabled DMA channels
- DMA controller configuration register
- Interrupt status
- Error interrupt status
- Transaction completing (terminal count) interrupt
- Raw status of DMA terminal count
- Raw status of DMA error interrupts
- Clear interrupt status RawTCStatus/TCStatus
- Clear interrupt status RawErrorStatus/ErrorStatus
- Allow single/burst requests to be generated by software
- Allow DMA last single/burst requests to be generated by SW
- Enables synchronization logic for the DMA request signals

Channel registers in 8237A
- Base Address Registers
- Base Word Count Registers
- Current Address Registers
- Current Word Count Registers
- Mode Registers

Common registers in 8237A
- Temporary Address Register
- Temporary Word Count Register
- Status Register
- Command Register
- Temporary Register
- Mask Register
- Request Register

8237A
- 4 (request) channel
- 1,115 (CAST, Inc.)
- Total 37 I/O

PL080
- Dual AHB Master
- 8 channel
- 82.0 k
- Total 415 I/O

PL081
- Single AHB Master
- 2 channel
- 24.9 k
- Total 300 I/O
API Platform

Source: Making embedded software reusable for SoCs
By Jack Shandle and Grant Martin

Today's platform is a virtual component for tomorrow's platform!
--VSIA PBD SG
Hardware Platforms are Not Enough!

A Software-Centric View of Platforms

Source: Platform-Based Design: A Tutorial, ISQED 2002, Grant Martin and Henry Chang
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
- Small “food print”
Variants of RTOS

- Pure real time OS

- OS Real-Time Extensions
Nexperia Software Module Architecture

Middleware Adaptation & Platform Reference Implementations
(e.g. OpenTV™, JavaTV™ or WebTV™)

OS dependent Device Drivers
pre-defined by Operating Systems (pSOS, WinCE)

Component Libraries
- Nexperia Streaming Components
- Device Library Components
- BSL Components
- HwAPI Components
- Nexperia HAL

to Bus DevLibs only

Infrastructure Libraries
- Nexperia Streaming Architecture
  - TSSA Infrastructure
  - THI
  - Inter-Processor Communication
    - RPC
    - HostComm
  - Operating System Abstraction
    - OSAL
    - MML
    - System Registry
  - BSL
    - BSL Manager
    - BSL Boards
    - Boot Manager (BTM)
Memory Controller

- The International Technology Roadmap for Semiconductors (ITRS) shows memory already accounting for over **50 percent** of a typical SoC, growing to **94 percent** by the year 2014.

- Memory design
  - Size, ports, device number, memory hierarchy
  - Application-specific behavior

- Memory power management

```plaintext
Predictor
Memory Controller

Addresses

- 000-2FF
- 300-4FF
- 500-7FF
- 800-9FF

DRAM 1
Active
35.7nJ
Zero delay

DRAM 2
Standby
0.83nJ
2 cycles

DRAM 3
NAP
0.32nJ
30 cycles

DRAM 4
PDN
0.005nJ
9000 cycles
```
Cache

• For an uncached processor operating from perfect memory, the number of cycles to execute a particular instruction is predictable.

• For a cached core, factors affecting the time an instruction takes to execute include
  – Is the instruction cached?
  – Is the address contained in r1 cached?
  – If the processor has an MMU, does the instruction fetch cause a TLB miss to occur? Does the data access cause a TLB miss to occur?
  – If a cache eviction occurs, did the old cache line contain dirty data?

• Alternative: Tightly Coupled Memory
  – DEC 21164a (2.0 V_{dd}, 0.35u, 400MHz, 20W max)
    – Caches dissipate 25% of the total chip power
  – DEC SA-110 (2.0 V_{dd}, 0.35u, 233MHz, 1W typ)
    – I$ (D$) dissipate 27% (16%) of the total chip power
Problem of SOC Functional Verification

- Functionality Verification has been over 60% (even 70%) of design cycle for millions gates design. For example, one cycle needs seconds.
- Traditional Simulator is virtual time, sequential. event process simulation, it is too slow. CPU time increases exponentially in big design.
- Verification for
  - Embedded application software (SW debug tools, ISS, RTOS, Apps, BFM)
  - Analog IP (mixed-signal) simulation
  - Different types of modeling for IPs (e.g, C/C++, HDL)
  - Stable IPs in the emulator
Verification Technology Overview

- Simulation Technology
  - Event-based
  - Cycle-based
  - Transaction-based
  - Code coverage
  - HW/SW co-verification
  - Emulation
  - Virtual (Soft) prototyping
  - Rapid prototyping
  - Hardware accelerator

- Static Technology
  - Lint check
  - Static timing

- Formal Technology
  - Theorem proving
  - Formal model check
  - Formal equivalence check
Software Verification – Soft Prototype

• Features
  – Trade-off by modifying system parameters & checking results
  – Develop & test device drivers
  – Test the correctness of compiler generated code
  – Visualize behavior of system and peripherals
  – Test the correctness of application algorithms
Building Soft Prototype

- **Requirement**
  - Processor debugger
  - Instruction Set Simulator (ISS) with a capability to interface C models of the peripherals
  - C models of the peripherals
- **Limitations**
  - Limited capacity
  - Limited speed
  - Accuracy of models
  - Synchronization
Transaction-Based Verification

- Raising the level of abstraction from signals to transactions.
- Transaction is transfer between testbench and DUV

\[
\text{Transactions to signals}
\]

\[
\text{DUV: Design Under Verification}
\]
Transaction Language

Standalone VC Testing

VC Testing in the System With the Same Transaction Vectors
How to Implementation Transactions

• HDL
  – Verilog: tasks are used to implement transactions
  – VHDL: procedures are used to implement transactions.
  – Limitation: data structure, test scenarios, and dynamic test creation

• High-Level Verification Language
  – Synopsys Vera
  – Verisity e Language
  – Cadence C++ class - TestBuilder
Transaction Language Levels

• The lowest level (VCI interface language)
  – A direct mapping to the VC Interface signals, and cell transfers.
  – Two formats
    • Simple function call set of statements
    • File (simulation vector) oriented set of commands

• The highest level (Transaction language)
  – Compatible with a subset of the SL-VCI (as in System Level Interface Documentation Standard) transactions.
High Level Transactions

Normal Packet Model

Advanced Packet Model
Example - Mapped to a 32-Bit AVCI

- channel cmdstruct;
- int trdid1;
- vsi_int8 [32] datavec;
- int err;

/* Open a channel with max packet size 16 bytes, no wrapping, and allocated address range from 0x100 to 0x1FF.*/
trdid1 = vciOpen (cmdstruct, 0, 0, 0, 16, 0, 0x100, 0x1FF, 0, 0);

/* Write 32 bytes starting from address 0x100, in channel trdid1 */
err = vciTWrite (trdid1, 0x100, sizeof(datavec), datavec);

err = vciClose(trdid1);

In a 32-bit AVCI, the transaction results to packet chain of two 16-byte contiguous, non-wrapped packets identified with the packet identifier.

In a BVCI, the identifiers would be ignored.

vciConfig 0 1 0 0 0 16 1 0 1 0// Configure first packet in the chain
vciWrite 0x00000100 0x15 0 0x12345678 0
vciWrite 0x00000104 0x15 0 0x9abcdef0 0
vciWrite 0x00000108 0x15 0 0x12345678 0
vciWrite 0x00000110C 0x15 1 0x9abcdef0 0
vciConfig 0 1 0 0 0 16 0 1 0// Configure the second packet in the chain
vciWrite 0x00000110 0x15 0 0x12345678 1
vciWrite 0x00000114 0x15 0 0x9abcdef0 1
vciWrite 0x00000118 0x15 0 0x12345678 1
vciWrite 0x0000011C 0x15 1 0x9abcdef0 1
Outline

- ARM-based SoC and Development Tools
- SoC Labs
- Available Lab modules in NCTU
- Summary
**Lab Module 1: Software Development**

- Familiarize with ARM software development tools, ADS.
  - Project management
  - Configuring the settings of build targets for your project
- Writing code (driver) for ARM-based platform design
- Software cost (code size) estimation.
  - The cost of a program includes Read Only (RO) data, Read Write (RW) data and Zero-Initialized (ZI) data.
- 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.
- Memory configuration
  - E.g., 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.
- Debug skills to be used to debug both software of processor and memory-mapped hardware design running at the target platform.
Lab Module 2: Resources of Target Platform

• A set of pre-built, well-defined, well-verified hardware and software components.

• Hardware components
  – Performance & constraint of different types of memory (SSRAM, SDRAM, and Flash, and the cache)
  – Data alignment and data layout (in which type of memory)
  – Available data bus and memory bandwidth
  – The use of timer and interrupt controller
Lab Module 2: Resources of Target Platform (cont.)

- Software components
  - The hardware abstraction layer, uHAL library
  - A set of Application Programming Interface (API), including API for Flash memory and PCI
  - A ported OS, uC/OS-2
    - ready for AP but not for ARMulator
    - details are in Lab 5
  - Basic applications, e.g., boot monitor, and example codes to make use of ARM Integrator
  - Hardware/software communication through
    - memory mapped I/O or uHAL library
    - interrupt handler or palling
Lab Module 2: Resources of Target Platform (cont.)

- Extension of the functionality of ARM Integration
  - 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
    - limitation of available semihosting bandwidth
  - User-programmable logic elements
    - in Lab 4
  - General Purpose Input/Output (GPIO).
    - not included in this course
  - User-defined boards
    - not included in this course
Lab Module 3: Virtual Prototyping

• The virtual prototyping allows designers to do the following:
  – Make trade-offs by modifying system parameters and checking the results
  – Perform the initial evaluation of design alternatives before detailed RTL design
  – Test interrupt handlers
  – Develop and test drivers for your IPs
  – Test the correctness of the application algorithms

• Hardware models in this lab
  – Memory-mapped IP: timer, MAC
  – Coprocessor: MAC

• Procedure to add the hardware models to ARMutator
• Verify the hardware models
Lab Module 4: IP Core Design

- IP authoring
  - Algorithm and architecture exploration in IP core design
  - RTL coding guidelines for IP authoring
  - AMBA-compliant IP
  - Hardware/software coordination

- Rapid prototyping
  - Already ready hardware resource in LM
  - Procedure to configure the hardware design to LM and software to CM

- Exercise: RGB to YCrCb Converter
Exercise and Homework

- Each Lab has its corresponding examples, exercises and homework
- The homework is designed to familiarize them with the design flow by using JPEG encoder as an example:
  - Hardware/software partition through profiling JPEG encoder C/C++ program (homework 1),
  - Port the program to target environment, make use of existing hardware resource (homework 2),
  - Hardware IP modeling and software driver authoring for JPEG encoder IP (homework 3), and finally
  - FPGA-proven, AMBA-complaint, JPEG encoder IP module(s) (homework 4).
Lab Module 5: RTOS

- Concept of RTOS
- Features of uC/OS-II
- Model ARMulator as ARM Integrator platform
  - Port uC/OS-II to ARMulator with uHAL
- Port applications to uC/OS-II
  - Partition original program into tasks
  - Necessary coding changes
  - Insert system calls into tasks
  - Create tasks and resources in main function
  - Driver authoring
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 (Xilinx XCV2000E, Altera LM-EP20K1000E)
    • ASIC Development Platform (Integrator/AP AHB )
    • Multi-ICE Interface
  – Advanced labs: RTOS (e.g., \mu C/OS-II) Verification/Validation

• SoC labs are challenges to universities
  – Various expertise
  – Tight schedule for (M.S.) students