探索嵌入式 ARM 平台與 SoC

Part I – ARM 架構瀏覽 .SoC 平台 . 關鍵概念

Jim Huang (**jserv**)
from **0xlab**

June 20, 2010

Rights to copy

Corrections, suggestions, contributions and translations are welcome!

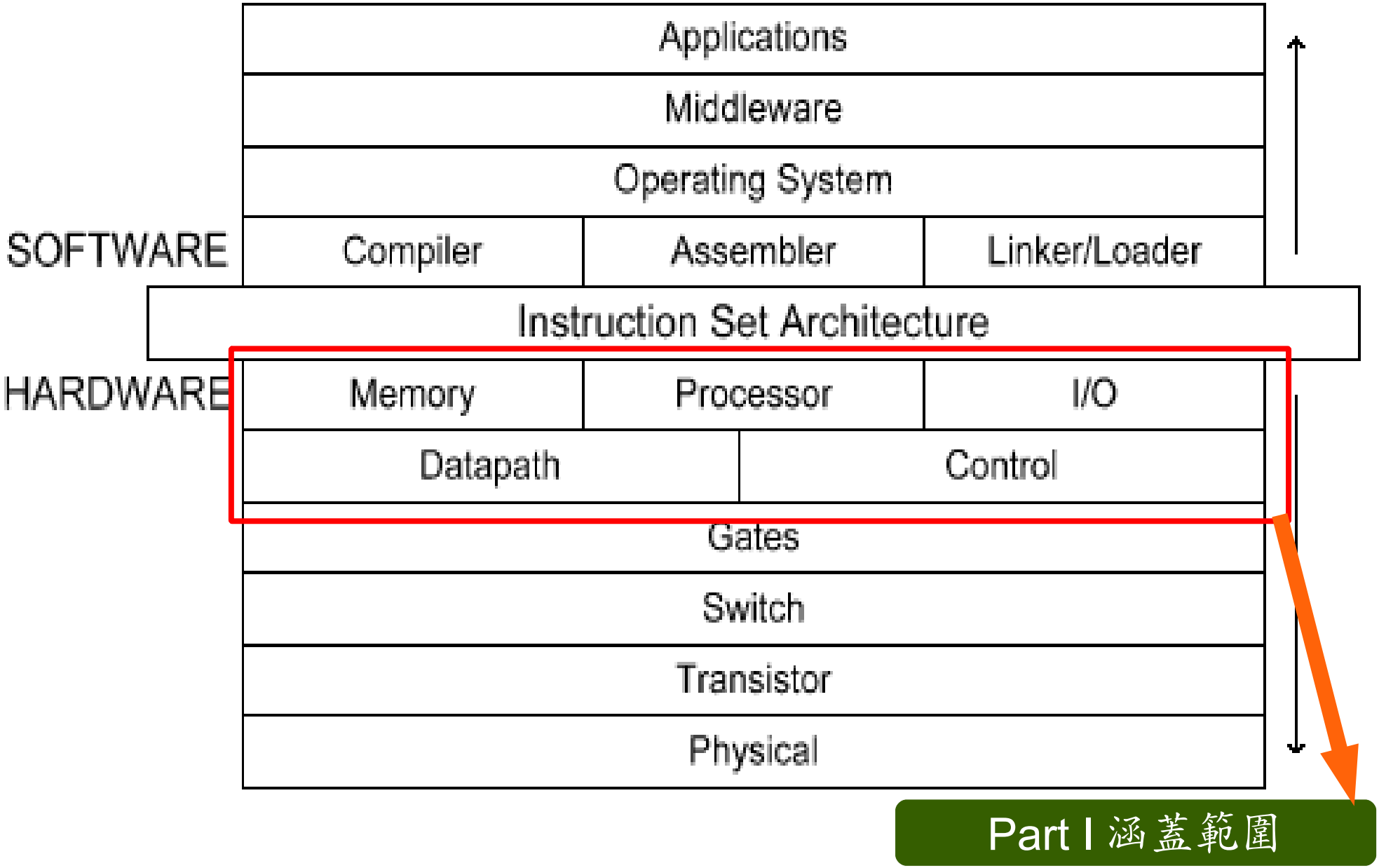
切入點

- 從軟體開發者的角度檢視 ARM 架構
- 善用開放原始碼軟體
  - GNU Toolchain – CodeSourcery 2009q1
  - QEMU 0.12.3
  - CuRT v1 (土製 ARM 即時作業系統)
- 參考 ARM SoC 平台： Marvell/Intel PXA255
- 作中學：觀察、系統模擬、驗證，動手

Agenda

- ARM 架構快速瀏覽
- ARM SoC 平台
- 關鍵概念：
  - 工作模式、暫存器組、系統狀態、指令集、例外處理

- ARM 架構快速瀏覽
  - ARM 歷史背景
  - ARM 的「家族」
- ARM SoC 平台
- 關鍵概念：
  - 工作模式、暫存器組、系統狀態、指令集、例外處理


Applications
Middleware
Operating System
SOFTWARE
Compiler
Assembler
Linker/Loader
Instruction Set Architecture
HARDWARE
Memory
Processor
I/O
Datapath
Control
Gates
Switch
Transistor
Physical

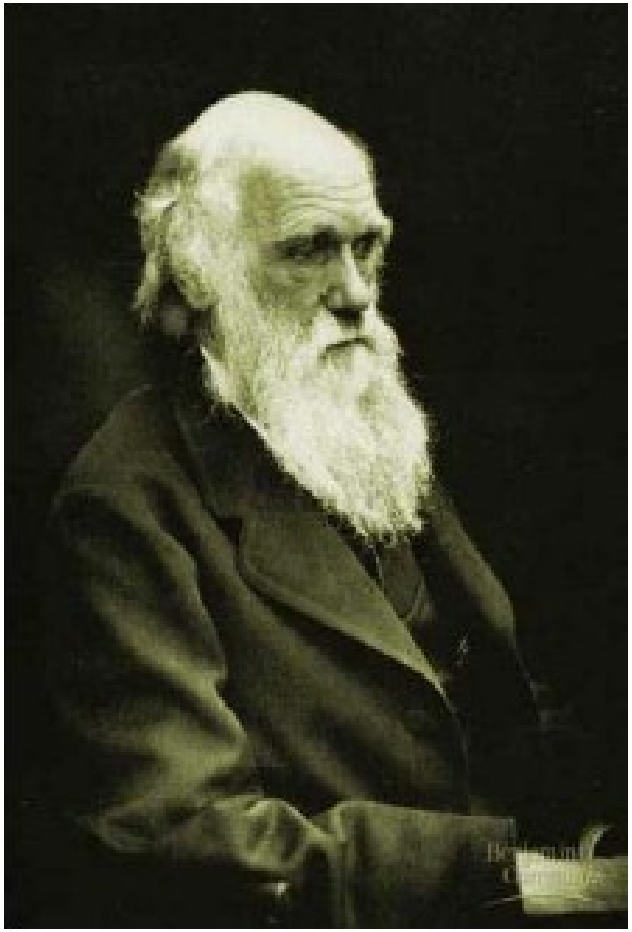
Part I 涵蓋範圍

無所不在的 ARM

Most of the above devides are powered by ARM.

Darwin's Concept

適者生存



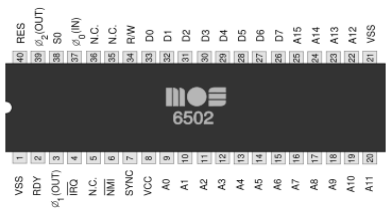
Not the strongest, But the fittest will survive

在 ARM 之前

- BBC Micro (BBC Microcomputer System) 在 BBC (Britsh Broadcasting Corporation) 主導的 BBC Computer Literacy Project 中，設計一系列的電腦與週邊裝置。於 1980 年代已有顯著的技術突破
  - 《 The Computer Programme 》 , 1982
  - programming, graphics, sound and music, Teletext, controlling external hardware and artificial intelligence

- （對 BBC Micro 的）電腦供應商 Acorn Computers 升級其 Atom microcomputer 設計，是爲 Proton， 提供更佳的圖形處理與更快的 2 MHz MOS Technology 6502 CPU
- 1982 年，競爭對手 Sinclair 推出 ZX Spectrum ，嚴重威脅 Acrom 獲利

ARM 就此展開



- 1983-1985 年間， Acorn Computers 推出第一款 ARM chip ，即 Acorn RISC Machine 。爾後該設計團隊於 1990 年代分後，公司命名爲 “ Advanced RISC Machine” (ARM)
- 成功取代 6502 processor 市場

- ARM1 – 由 VLSI Technology, Inc. 製造晶圓生產 (1985)
- ARM2 – 量產 (1986)
  - 32-bit data bus, 26-bit address space (top 6 bits used as status flag)
  - 16 32-bit registers (1 PC), 30,000 transistors
  - No microcode, cache, Low power
- 國家半導體 NS 32016 的出現，威脅 Acorn 的獲利模式，隨後引發財務危機， 1985 年，義大利設備商 Olivetti Ing et Cie 入主
- ARM3 – 引入 4KB Cache, 25MHz (1989-1990)

ARM1 Architect

Steve Furber

Roger Wilson

Robert Heaton

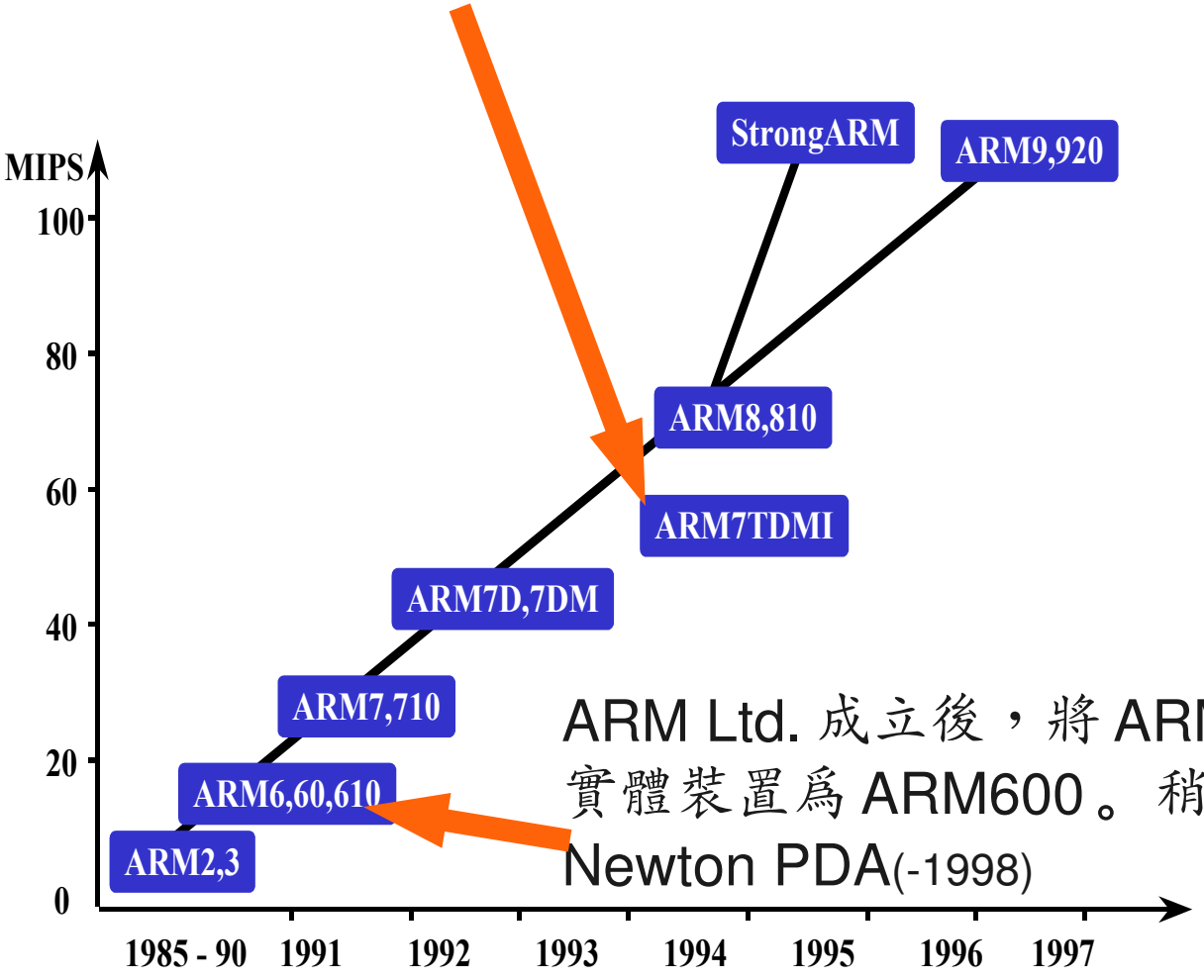
Steve Furber
sfurber@cs.man.ac.uk
Father of ARM

ARM Ltd. 出現

- 1990 年代， Acorn 的財務狀況趨於穩定，亟欲轉型。該年底，於英國劍橋，由 Apple, Acorn, VLSI 三家公司共同成立 ARM Ltd.
- ARM Ltd. 成立後，將 ARM3 的下一代命名爲 ARM6 ，實體裝置爲 ARM600
- 稍候的 ARM610 用於 Apple Newton PDA(-1998)

- 1992 年底， 3DO 公司（主要產品：遊戲機）自 ARM Ltd. 取得 ARM 授權並成功量產電子設備，自此， ARM 在出售晶片設計技術授權，獲得成功
- 目前採用 ARM 技術知識產權 (IP) 核心的微處理器，即我們常所說的 ARM 微處理器
- ARM 遍及工業控制、消費性電子產品、通信網路系統等領域
- ARM7TDMI 是早期成功的 ARM core ，出貨量達數億單位

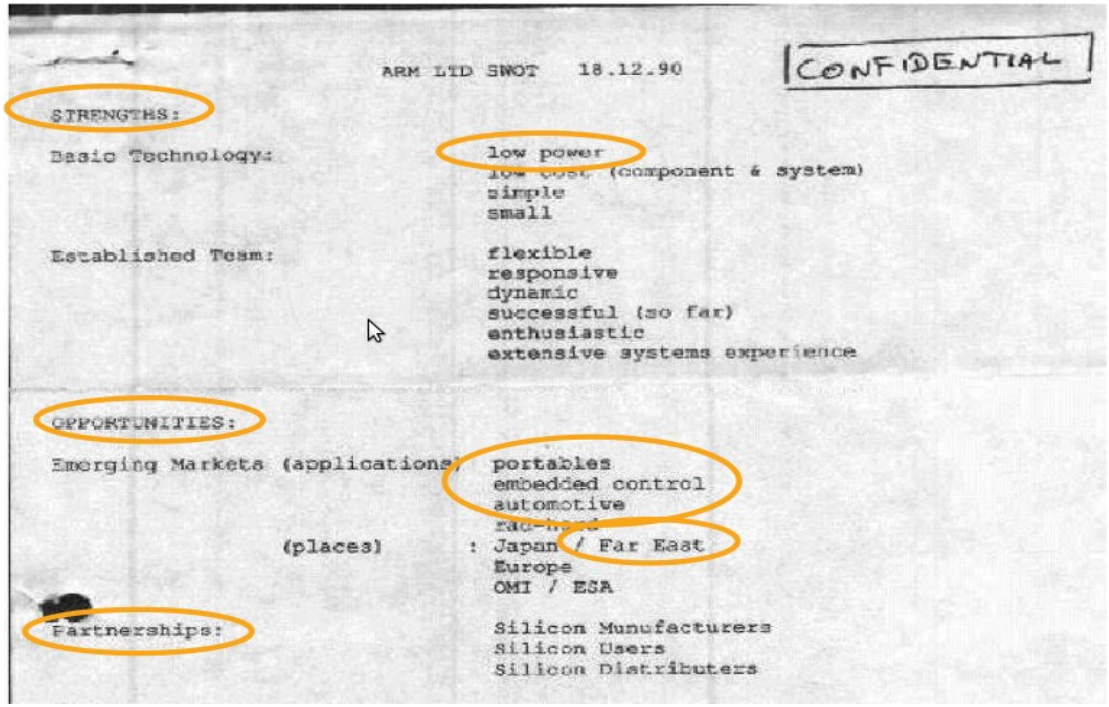
ARM7TDMI 是早期成功的 ARM core ，出貨量達數億單位



ARM Ltd. 成立後，將 ARM3 的下一代命名爲 ARM6 ，實體裝置爲 ARM600 。稍候的 ARM610 用於 Apple Newton PDA(-1998)

ARM 的 SWOT 分析

- 由 1990 年 12 月 18 日（時值 ARM Ltd. 成立 21 日）的內部文件，評估企業的 SWOT
  - Strengths: 優勢
  - Weaknesses: 劣勢
  - Opportunities: 競爭市場的機會
  - Threats: 威脅



ARM LTD SWOT 18.12.90

CONFIDENTIAL

STRENGTHS:

Basic Technology: low power
low cost (component & system)
simple
small

Established Team: flexible
responsive
dynamic
successful (so far)
enthusiastic
extensive systems experience

OPPORTUNITIES:

Emerging Markets (applications) portables
embedded control
automotive
rad-hard

(places) : Japan / Far East
Europe
OMI / ESA

Partnerships: Silicon Manufacturers
Silicon Users
Silicon Distributers

ARM LTD SWOT 18.12.90

CONFIDENTIAL

STRENGTHS:

Basic Technology: low power
low cost (component & system)
simple
small

Established Team: flexible
responsive
dynamic
successful (so far)
enthusiastic
extensive systems experience

OPPORTUNITIES:

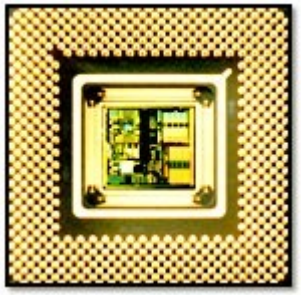
Emerging Markets (applications) portables
embedded control
automotive
rad-hard

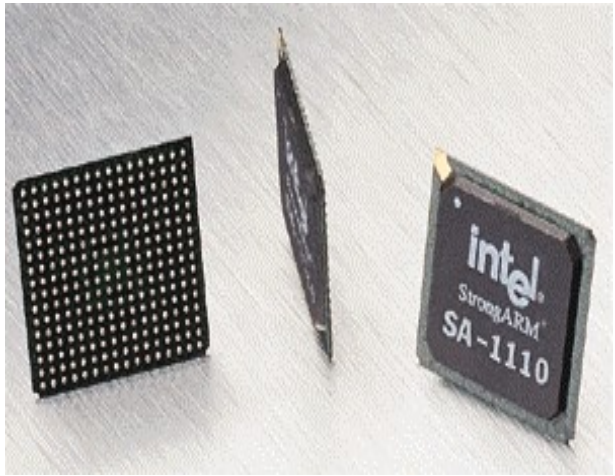
(places) : Japan / Far East
Europe
OMI / ESA

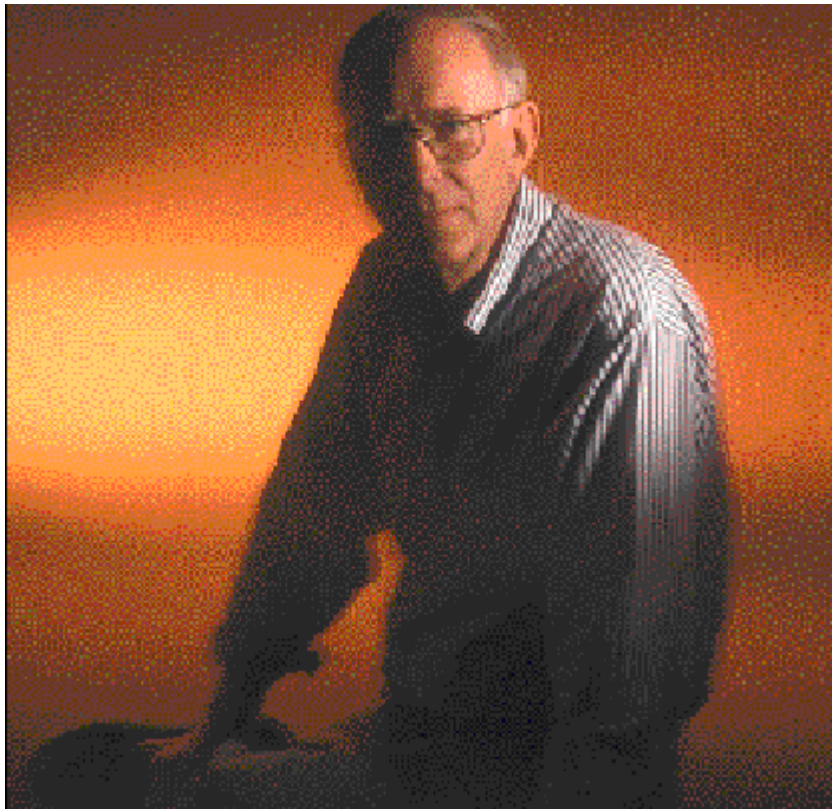
Partnerships: Silicon Manufacturers
Silicon Users
Silicon Distributers

ARM 的時代

- DEC 自 ARM Ltd. 取得 ARM6 的授權，依據此基礎研發出時脈可達 233MHz 的 StrongARM (SA-1x 系列)
  - 使用於 PDA 裝置，如 HP iPAQ
  - 500 mWatt of power @160MHz @.35μm
- DEC 後來爲了與 Inte1 控訴和解，將技術移轉到 Intel (1997)
- Intel 依據 StrongARM 的基礎，發展出 Xscale
- StrongARM 關鍵的成員出走 Intel 後，成立 P.A. Semi (Palo Alto Semiconducto)(2003) ，爾後被 Apple 併購 (2008)
- Intel 出售 XScale 產品線予 Marvell (2006)







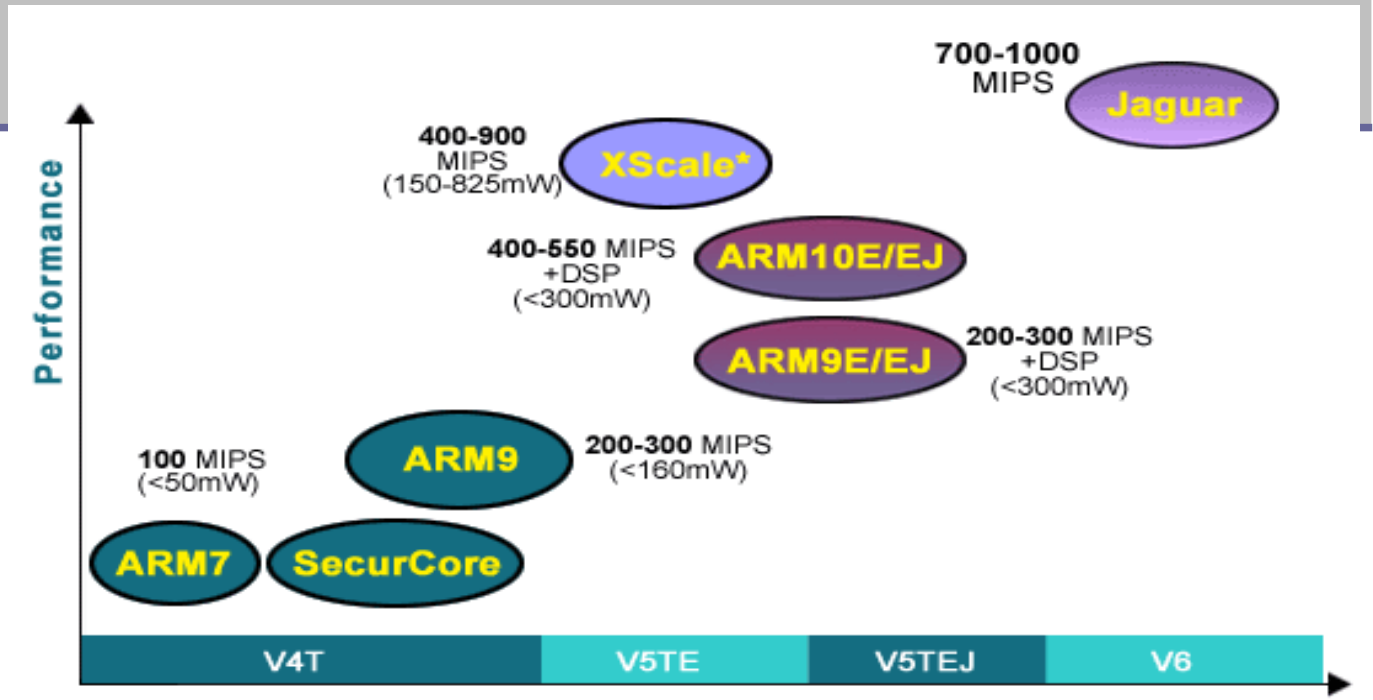
Dan Dobberpuhl

Father of Alpha, StrongARM

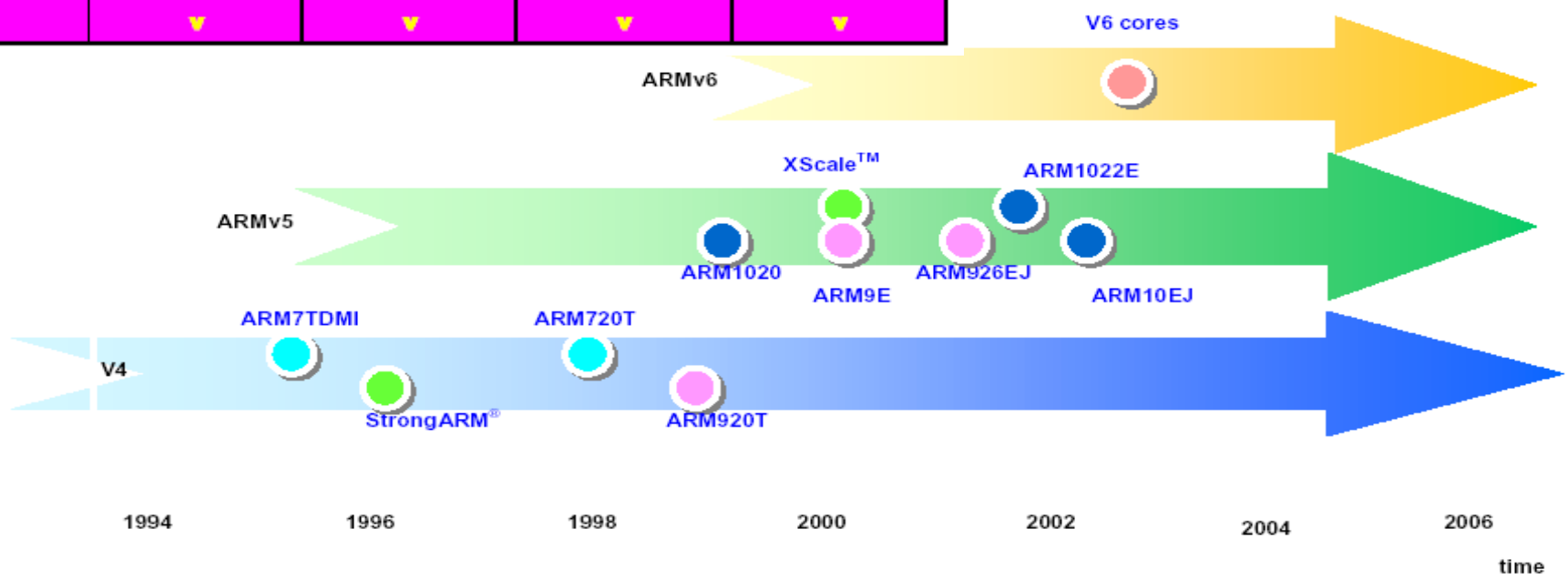
ARM 的「家族」

- Architecture Versions
  - ARM V3, V4, V5, V6, V7
  - 使用 “ architecture” 一詞標注硬體設計架構
    - 更具體來說，指的是 ISA (Instruction Set Architecture)

- Implementations（涉及不同的製程）
  - ARM6 (1991), ARM7 (1995), ARM9 (1997)
  - ARM10 (1999), ARM11 (2003)
  - 使用 “ cores” 一詞標注其世代演進



Architecture	Feature Set			
	Thumb®	DSP	Jazelle™	Media
V4T	✓			
V5TE	✓	✓		
V5TEJ	✓	✓	✓	
V6	✓	✓	✓	✓



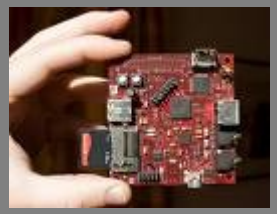
```
0x41 A (ARM Ltd)
0x44 D (DEC)
0x69 I (Intel Corporation)
```

Architecture version

```
/ # cat /proc/cpuinfo
Processor        : ARMv7 Processor rev 3 (v7l)
BogoMIPS         : 471.61
Features         : swp half thumb fastmult vfp edsp thumbee neon
CPU implementer  : 0x41
CPU architecture: 7
CPU variant      : 0x1
CPU part         : 0xc08
CPU revision     : 3

Hardware         : OMAP3 Beagle Board
Revision         : 0020
Serial           : 0000000000000000
```

ARM ISA feature



採用 ARM 技術知識產權 (IP) 核心的微處理器，即我們常所說的 ARM 微處理器，實際的組態變化相當多元

ARMv1

- Implementation: 僅在原型機 ARM1 出現
- 指令集：
  - 基本的資料處理指令 (ALU) ，不包含乘法指令
  - Byte, half-word, word 的 Load/Store 指令
  - 跳躍 (jump/branch) 指令，包括 procedure call
  - 軟體中斷 (SWI) 指令
  - 定址空間： 64MB

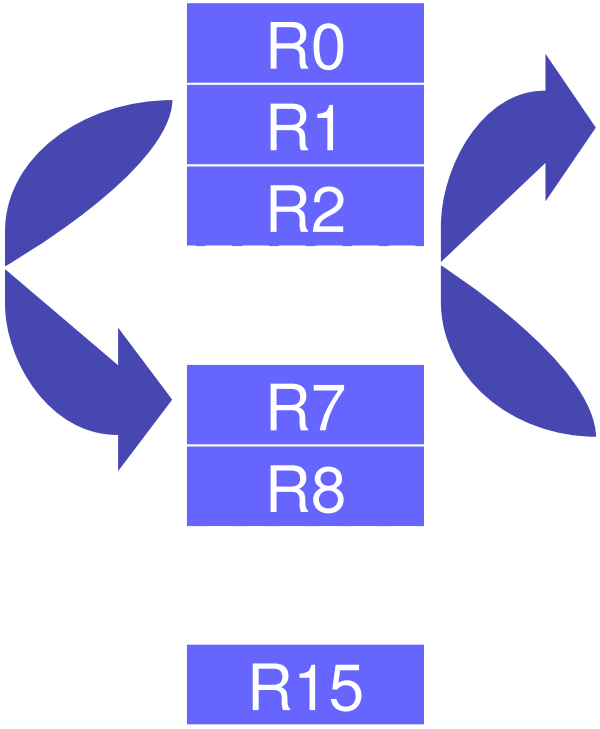
ARMv2

- Implementation: ARM2 與 ARM3 (ARMv2a) 等硬體
- 指令集：
  - 增加乘法和乘加指令
  - 支援輔助運算器操作指令
  - 快速中斷模式 (FIQ)
  - SWP/SWPB 的最基本記憶體與暫存器交換指令
  - 定址空間： 64MB (32-bit data bus, 26-bit address space (top 6 bits used as status flag)

ARMv2 :: SWP

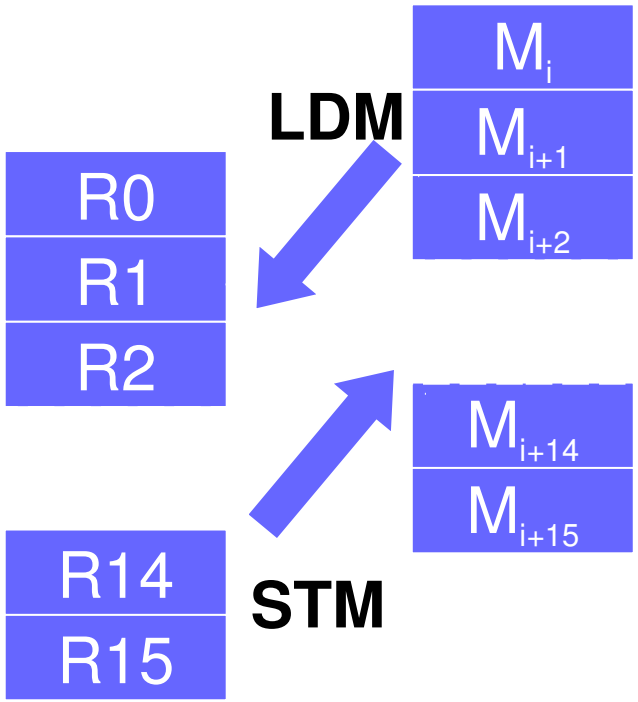
- 在 ARM 暫存器組交換一個 word
  - Two cycles

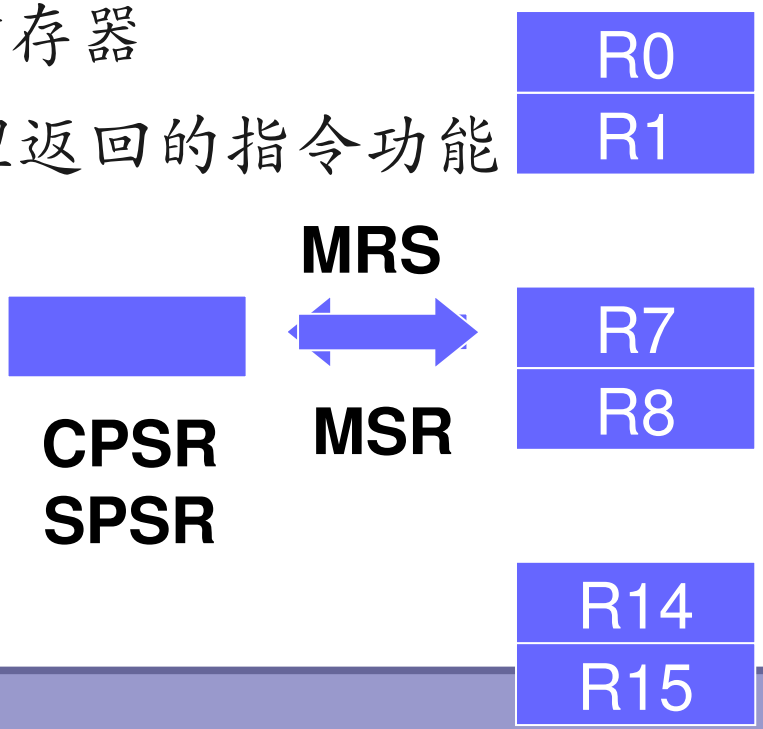
  並確保
- single atomic action
  - Support for RT semaphores



ARMv3

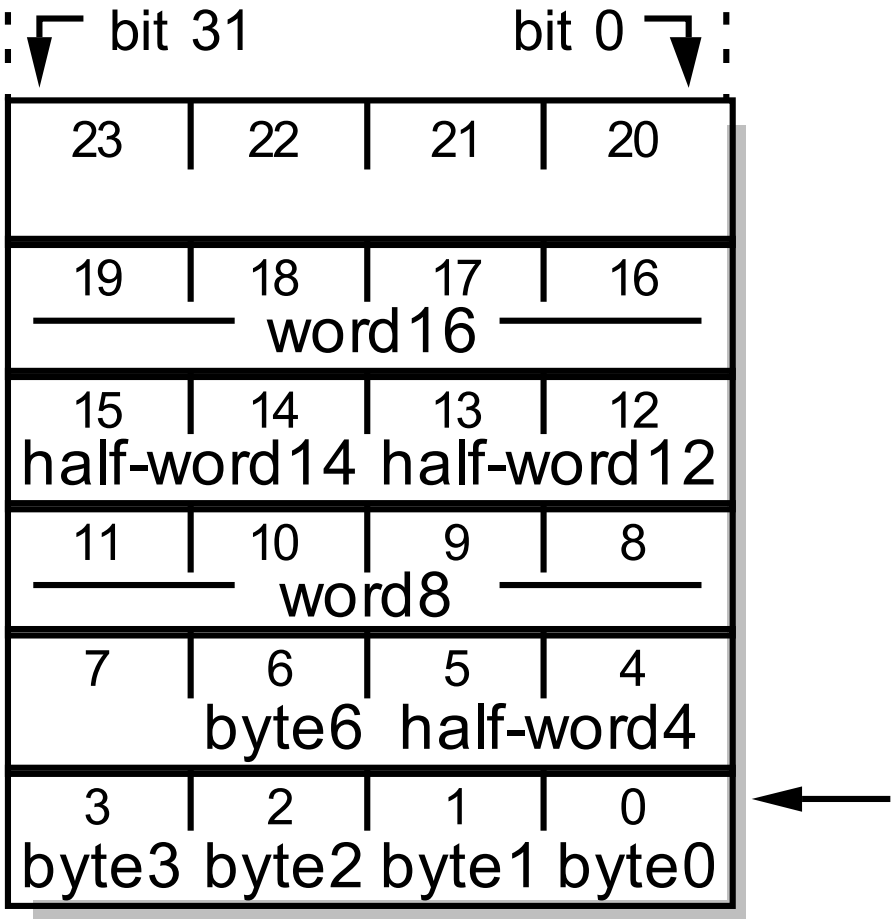
- 較大的改動，將定址空間增至 32-bit (4GB) 。追加 CPSR 和 SPSR， 以利異常處理。增加中止和未定義等兩種處理器模式
- Implementation: ARM6
- 裝置範例： Apple Newton PDA
- 指令集：
  - 增加 MRS/MSR 指令，存取新增的 CPSR/SPSR 暫存器
  - 增加從異常處理返回的指令功能



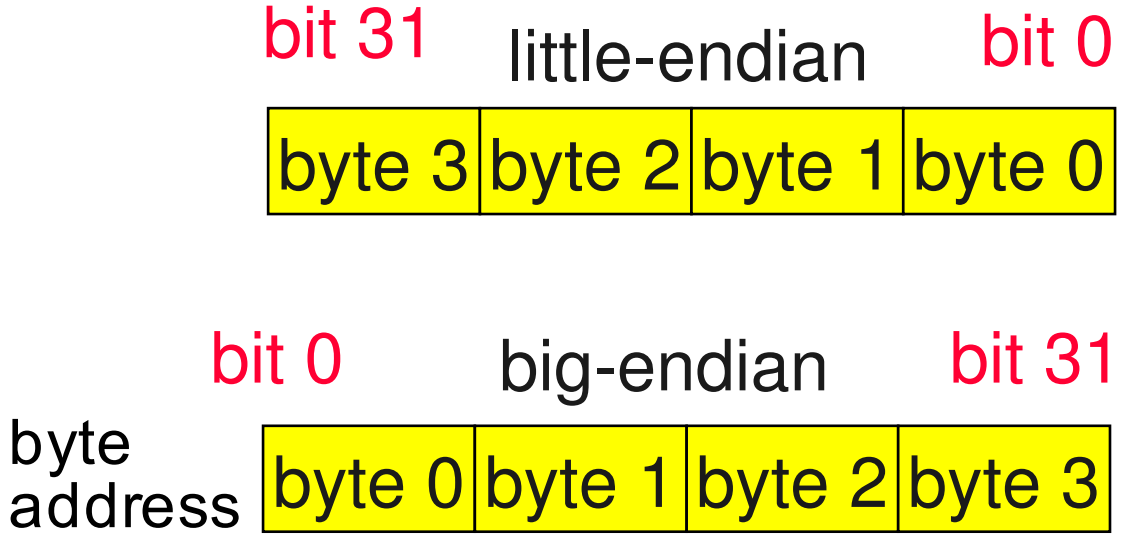


ARMv3 :: Memory Addressing

alignment 相當重要，一些隱性的陷阱可參考拙作〈我是軟體，那些處理器教我的事〉(COSCUP 2008)



- Byte : 8 bits
- Halfword : 16 bits
  - must be aligned to 2-byte boundaries
- Word : 32 bits
  - must be aligned to 4-byte boundaries
- ARM address can be 32 bits long.
- Address refers to byte.
  - Address 4 starts at byte 4.
- 可在系統啓動時，調整爲 little- or big-endian mode

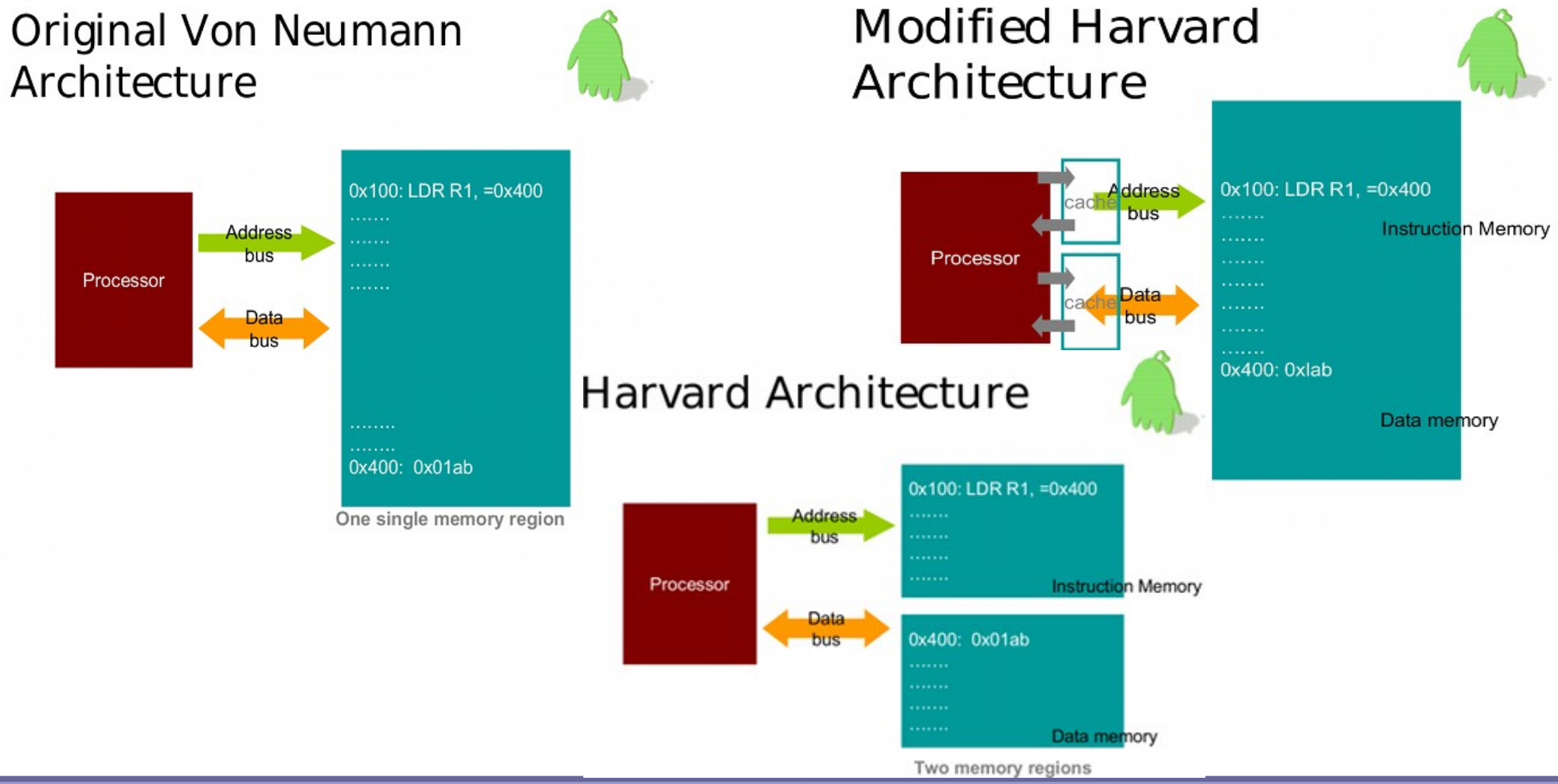


ARMv4

- 廣泛應用的 ARM 架構，對 ARMv3 作進一步擴充，引入引進了 16-bit Thumb 指令集（可選擇）
- Implementation: ARM7, ARM9, StrongARM
- 裝置範例： GameBoy Advance, Nintendo DS, iPod, LEGO NXT, Openmoko GTA01/GTA02
- 指令集：
  - 增加 16-bit Thumb 指令集
  - 加強軟體中斷 (SWI) 指令的功能
  - 處理器系統模式引進特權方式 (SVC) 時，以 user 暫存器操作
  - 將未使用的指令空間，捕捉爲未定義指令

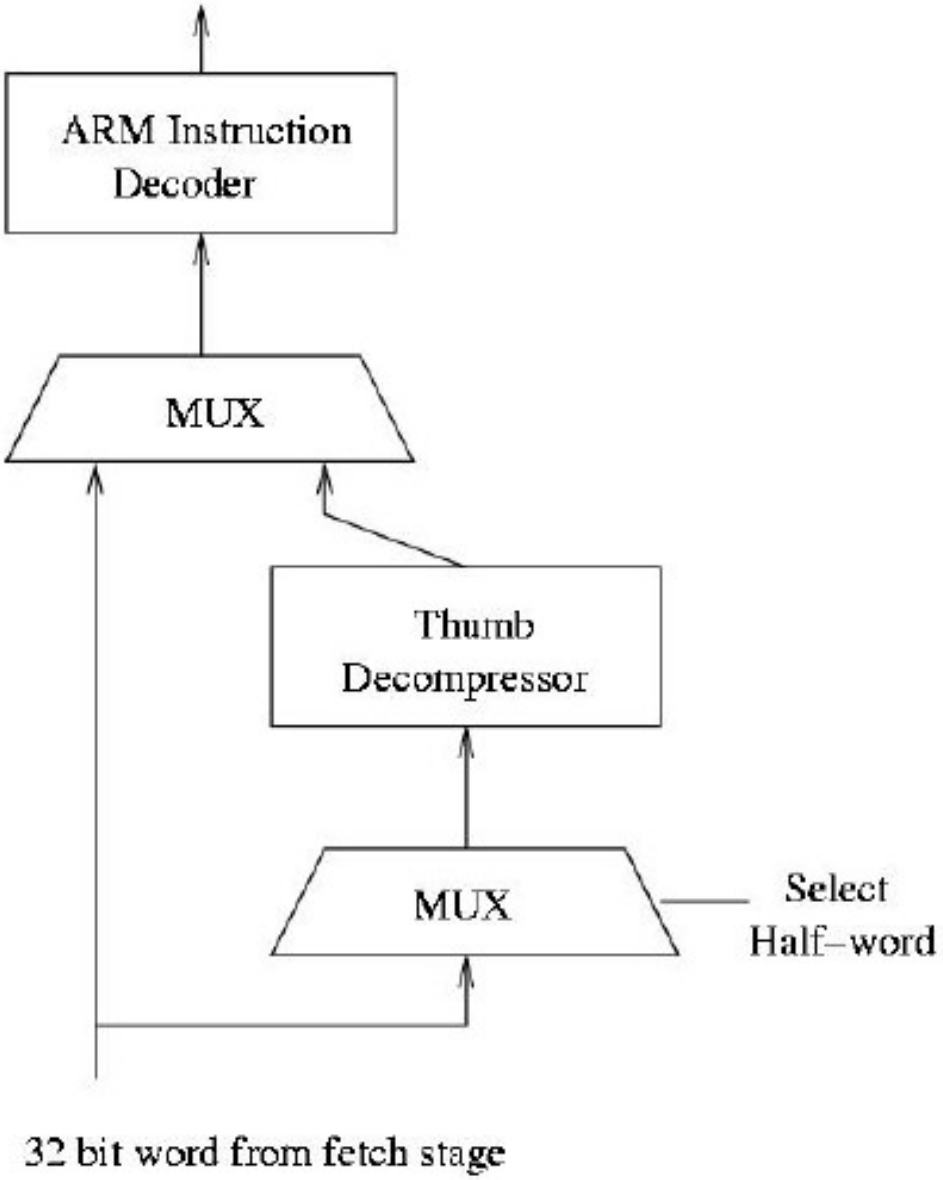
ARMv4

- 從 bus-structure 的角度來看 ARMv4 的 Implementation
  - ARM7 : von Neumann architecture
  - ARM9 : modified Harvard architecture



ARMv4 :: ARM/Thumb

- 32-bit ARM Instruction Sets
- 16-bit Thumb Instruction Set
  - 更高的指令集密度
  - 對 cache 處理更有效率
- All instructions are executed as ARM
- Decompressor converts Thumb to equivalent ARM instruction
- Decompressor present in the decode stage of the pipeline.



ARMv5

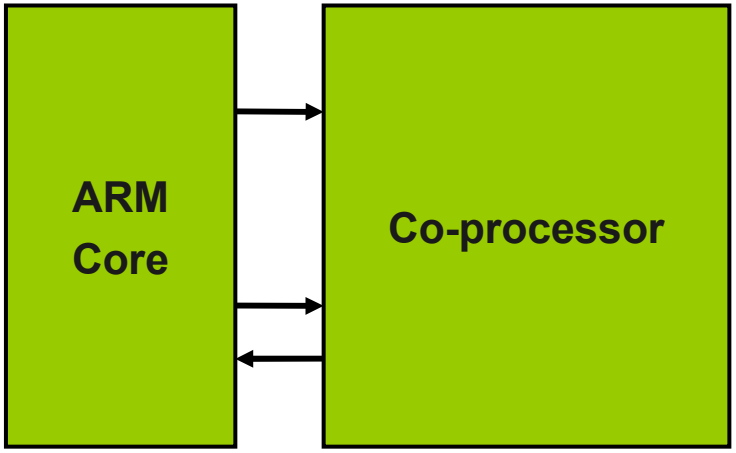
- Implementation: ARM10, Xscale
- 裝置範例： HTC Dream (Google G1 Phone)
- 指令集：
  - BLX 指令 (Improved ARM and Thumb interworking)
  - CLZ 指令 (count leading-zeroes)
  - BRK 中斷指令
  - 增加 DSP 強化指令 (v5TE)
    - E – enhanced DSP instructions including saturated arithmetic operations and 16-bit multiply operations
  - 增加 Jazelle 指令集
    - J – support for new Java state, offering hardware and optimized software acceleration of bytecode execution.
  - 爲 co-processor 增加更多可選擇的指令

ARMv5 :: DSP Extension (E)

Instruction	Operation	Purpose
SMLAxy{cond}	16 x 16 + 32 → 32	Signed MAC
SMLAWy{cond}	32 x 16 + 32 → 32	Signed MAC wide
SMLALxy{cond}	16 x 16 + 64 → 64	Signed MAC long
SMULxy{cond}	16 x 16 → 32	Signed multiply
SMULWy{cond}	16 x 32 → 32	Signed multiply long
QADD Rd, Rm, Rs	SAT(Rm + Rd)	Saturating add
QDADD Rd, Rm, Rs	SAT(Rm + SAT(Rs x 2))	Saturating add double
QSUB Rd, Rm, Rs	SAT(Rm – Rd)	Saturating subtract
QDSUB Rd, Rm, Rs	SAT(Rm – SAT(Rs x 2))	Saturating subtract double
CLZ{cond} Rd, Rm	COUNTZ(Rm)	Count leading zeros

ARMv5 :: co-processor

- Marvell/Intel Xscale Wireless MMX co-processor
  - 概念：將 x86 行之有年的 MMX/SSE 指令集設計概念，應用於 Xscale 處理器，以 co-processor 的形式存在
- ARM 的架構最多可支援 16 個 co-processor ，以 Xscale 爲例：
  - co-processor 0
  - co-processor 1
- Xscale co-processor instructions
  - Coprocessor data transfers
  - Coprocessor data operations
  - Coprocessor register load and store



ARMv6

- Implementation: ARM11
- 裝置範例： Apple iPhone (1st/2nd)
- 指令集 / 特性：
  - 引入 60 個以上新的 SIMD(Single Instruction Multiple Data) ，使多媒體處理速度快 1.75 倍
  - 提出新的 atomic operations(LDREX/STREX)
    - 避免 swp 指令的限制
  - 改進記憶體管理，使系統性能提高 30%
  - 強化 Mixed-Endian 支援 (Little-Endian OS + Bit-Endian Data for TCP/IP)
  - 改進 unaligned data 支援，進一步提昇字串操作的效能

ARMv6 :: SIMD Extensions

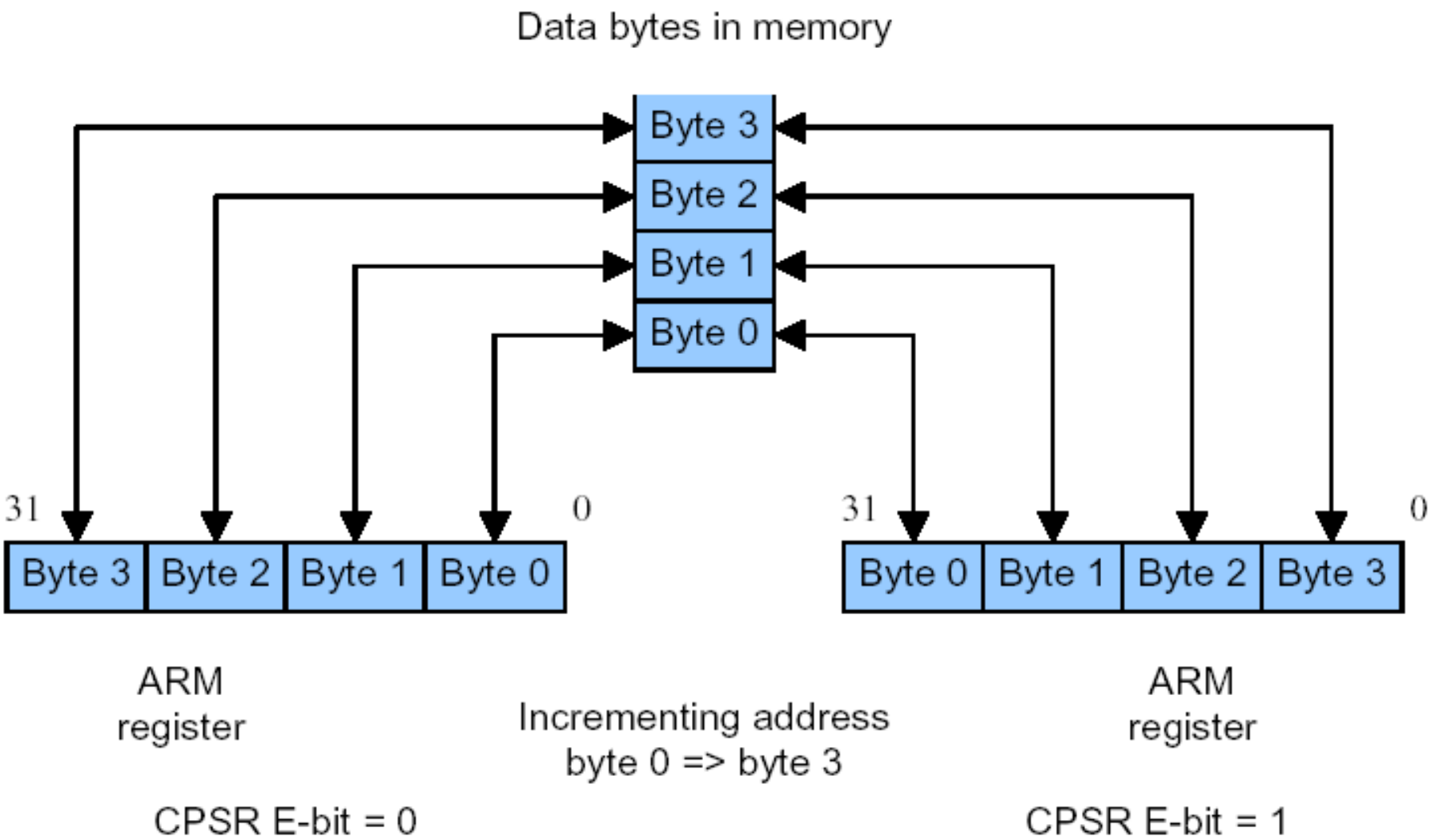
ARMv5TE: 5 cycles in a single-cycle implementation

```
SMULTT Real,Ra,Rb ;Real = Ra.real*Rb.real
SMULBB Temp,Ra,Rb ;Temp = Ra.imag*Rb.imag
SUB Real,Real,Temp ;Real = Ra.real*Rb.real - Ra.imag*Rb.imag
SMULTB Imag,Ra,Rb ;Imag = Ra.real*Rb.imag
SMLABT Imag,Ra,Rb ;Imag = Ra.real*Rb.imag + Ra.imag*Rb.real
```

ARMv6: 2 cycles in a single-cycle implementation

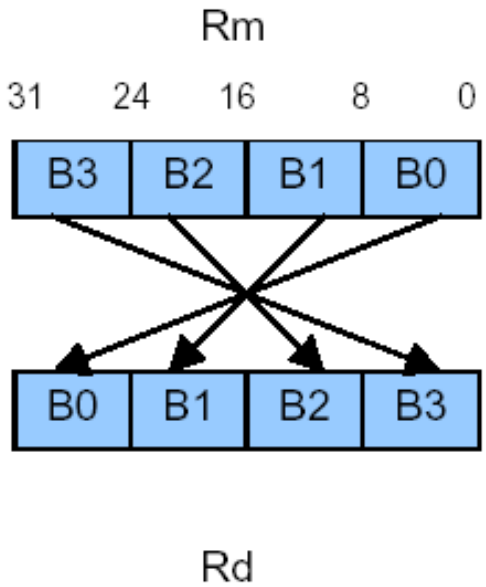
```
SMUSD Real,Ra,Rb ;Real = Ra.real*Rb.real - Ra.imag*Rb.imag
SMUADX Imag,Ra,Rb ;Imag = Ra.real*Rb.imag + Ra.imag*Rb.real
```

ARMv6 :: Endianness Support (E bit)

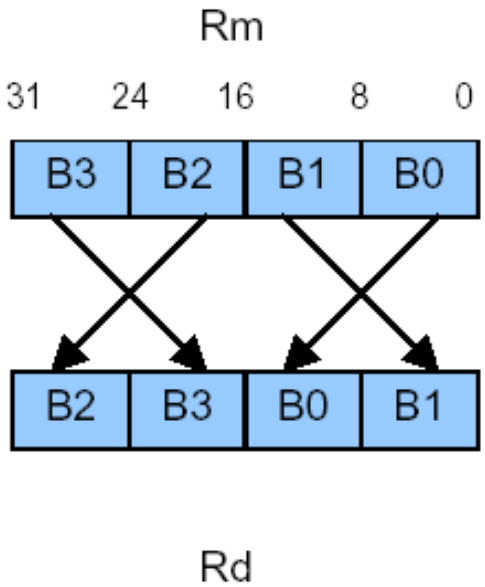


ARMv6 :: Byte Reverse instruction

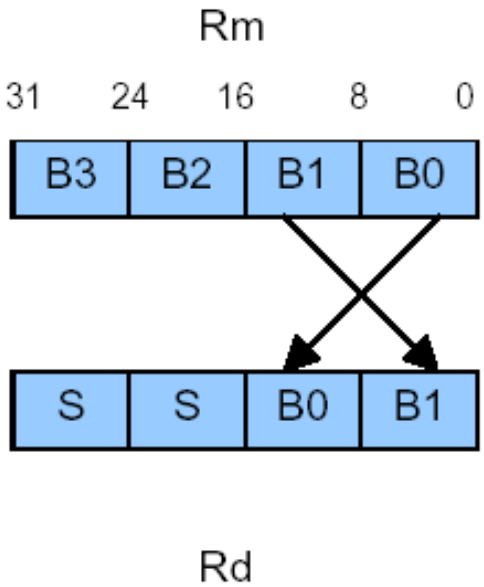
REV{<cond>} Rd, Rm



REV16{<cond>} Rd, Rm



REVSH{<cond>} Rd, Rm



ARMv6 :: unaligned access

Aligned access

0xC	0xD	0xE	0xF
0x8	0x9	0xA	0xB
0x4	0x5	0x6	0x7
0x0	0x1	0x2	0x3

```
MOV r0, #4
LDR r0, [r0]
```

Unaligned access

0xC	0xD	0xE	0xF
0x8	0x9	0xA	0xB
0x4	0x5	0x6	0x7
0x0	0x1	0x2	0x3

```
MOV r0, #5
LDR r0, [r0]
```

- Test on BeagleBoard 500 MHz + 0xdroid (0xlab's Android distribution)

```
mmemcpy_use_unaligned :  (16 bytes copy) =  167.4 MB/s /  368.7 MB/s
memcpy :                 (16 bytes copy) =  137.0 MB/s /  352.5 MB/s

mmemcpy_use_unaligned :  (24 bytes copy) =  231.3 MB/s /  552.7 MB/s
memcpy_neon :            (24 bytes copy) =  199.3 MB/s /  350.5 MB/s

mmemcpy_use_unaligned :  (31 bytes copy) =  315.6 MB/s /  714.2 MB/s
memcpy :                 (31 bytes copy) =  267.9 MB/s /  375.6 MB/s
```

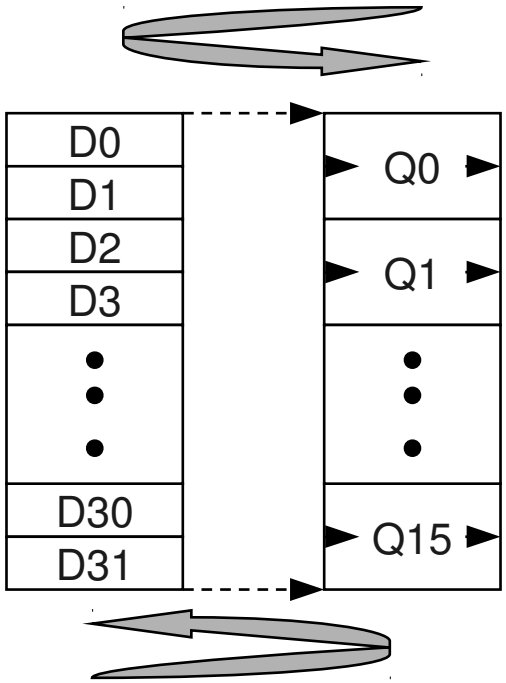
ARMv7

- Implementation: Cortex-A8
  - 第一個 ARMv7 ISA 的完整 implementation ，包含 Advanced SIMD Media Extension (NEON)
  - 自 ARMv7 起， core 命名改以 Cortex 開頭

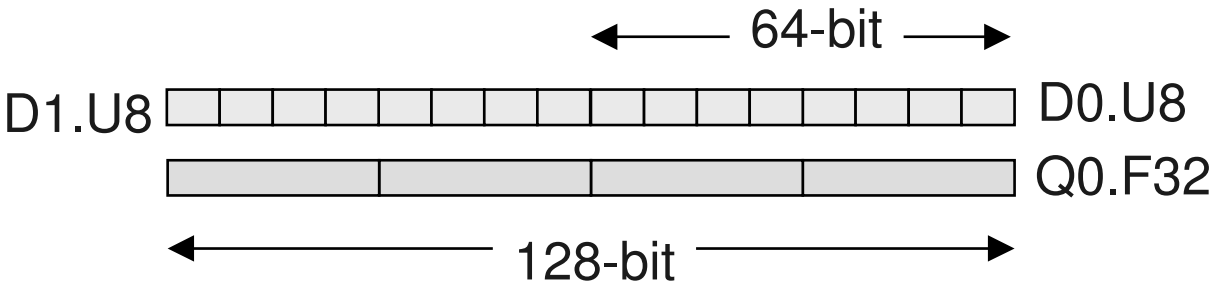
- In-order, dual-issue superscalar core
  - 13-stage integer pipeline
  - 10-stage NEON media pipeline
  - Dedicated L2 with 9-cycle latency
  - Branch predictor based on global history
  - NEON: 64/128-bit SIMD, 2x-4x ↑ over prior ARMv6 SIMD
- ARMv7 關鍵特性
  - 引入 Thumb-2, Thumb-EE (for dynamic/JIT compiler)
- 對應的 Implementation 時脈達 1 GHz
  - 功耗小於 300mW
  - < 4mm^2 at 65nm (不含 NEON, L2 cache, ETM)

ARMv7 :: NEON

- 64/128-bit 混合式 SIMD 架構
- Two aliased register files
  - 32 x 64-bit (D0-D31)
  - 16 x 128-bit (Q0-Q15)
  - Shared with VFP
- Integer and SP FP processing
  - 8, 16, 32, 64-bit integers
- Encoded in ARM and Thumb-2

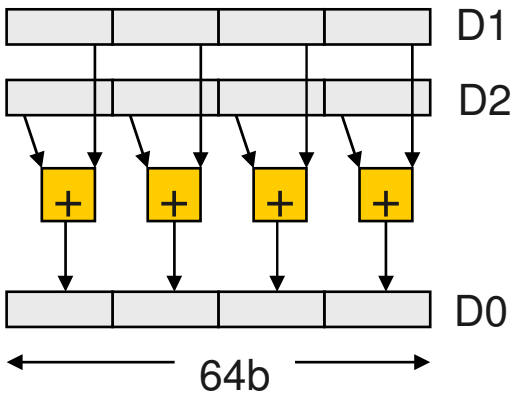


Alias, same physical structure

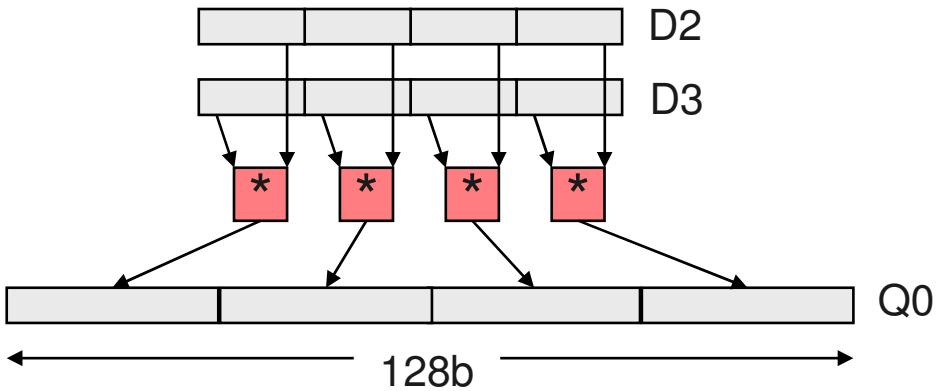


Source: Cristinel Ivan of ARM

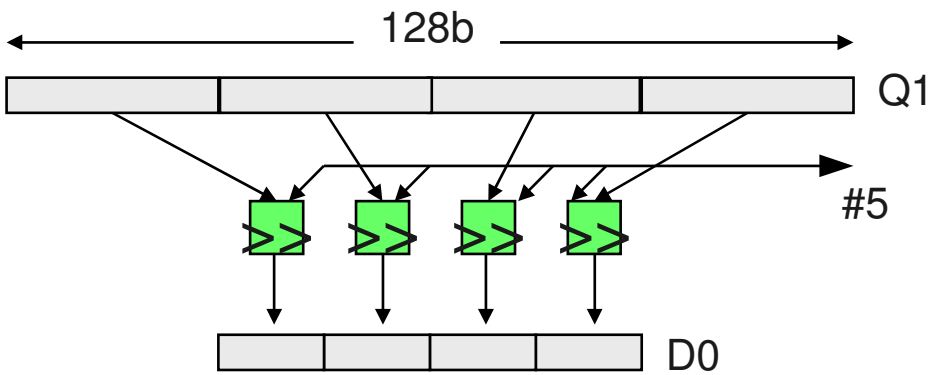
```
vadd.I16 D0, D1, D2
```




```
vmul.I32.S16 Q0, D2, D3
```

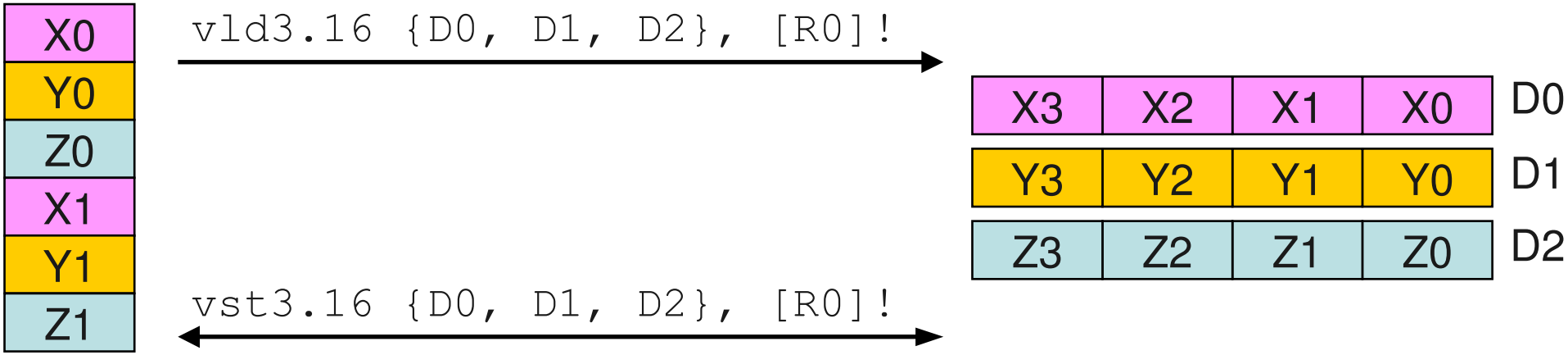



```
vshr.I16.I32 D0, Q1, #5
```



Source: Grisenthwaite of ARM

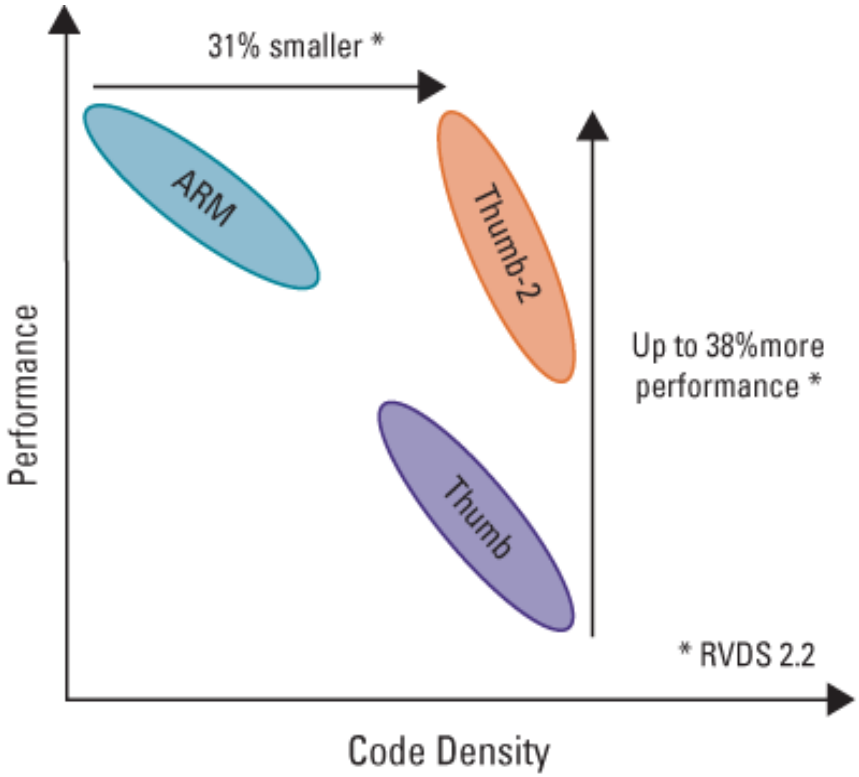
- NEON Load or store 1-element or 2, 3, 4-elelment structure
- Handle complex number, coordinates, etc.
- Easy AoS to SoA



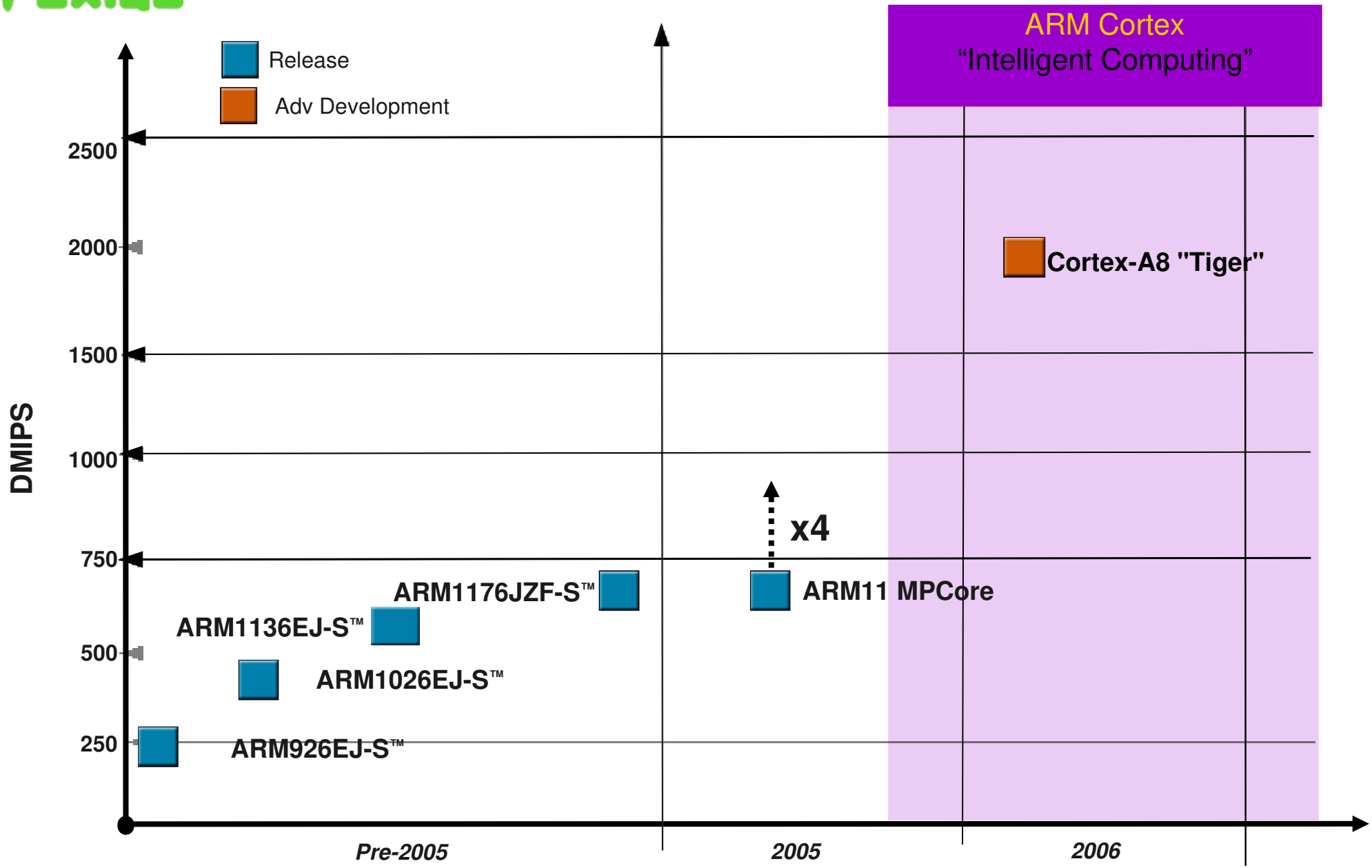
- No data swizzling needed!
- Fewer instruction, higher performance

ARMv7 :: Thumb-2

- 大幅改進 Thumb 指令集的限制
  - Thumb 的空間使用
  - ARM 的效能
- 引入混合 16-/32-bit 指令集的並存模式
- 不再需要繁瑣的狀態轉換
  - 兼顧效能與程式碼密度



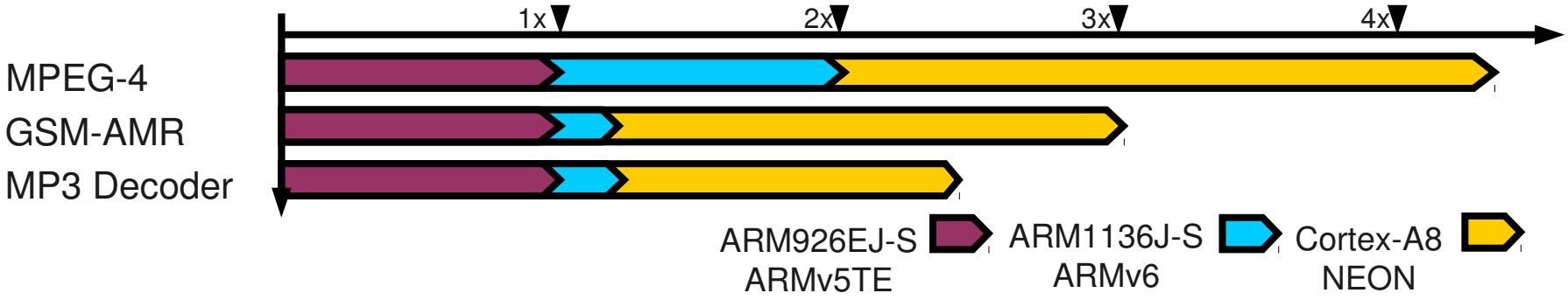
ARM core performance roadmap



Source: Williamson of ARM at Fall processor Forum 05

Cortex-A8 NEON Performance

- Cortex-A8 NEON performance vs. ARMv5 and ARMv6 implementations



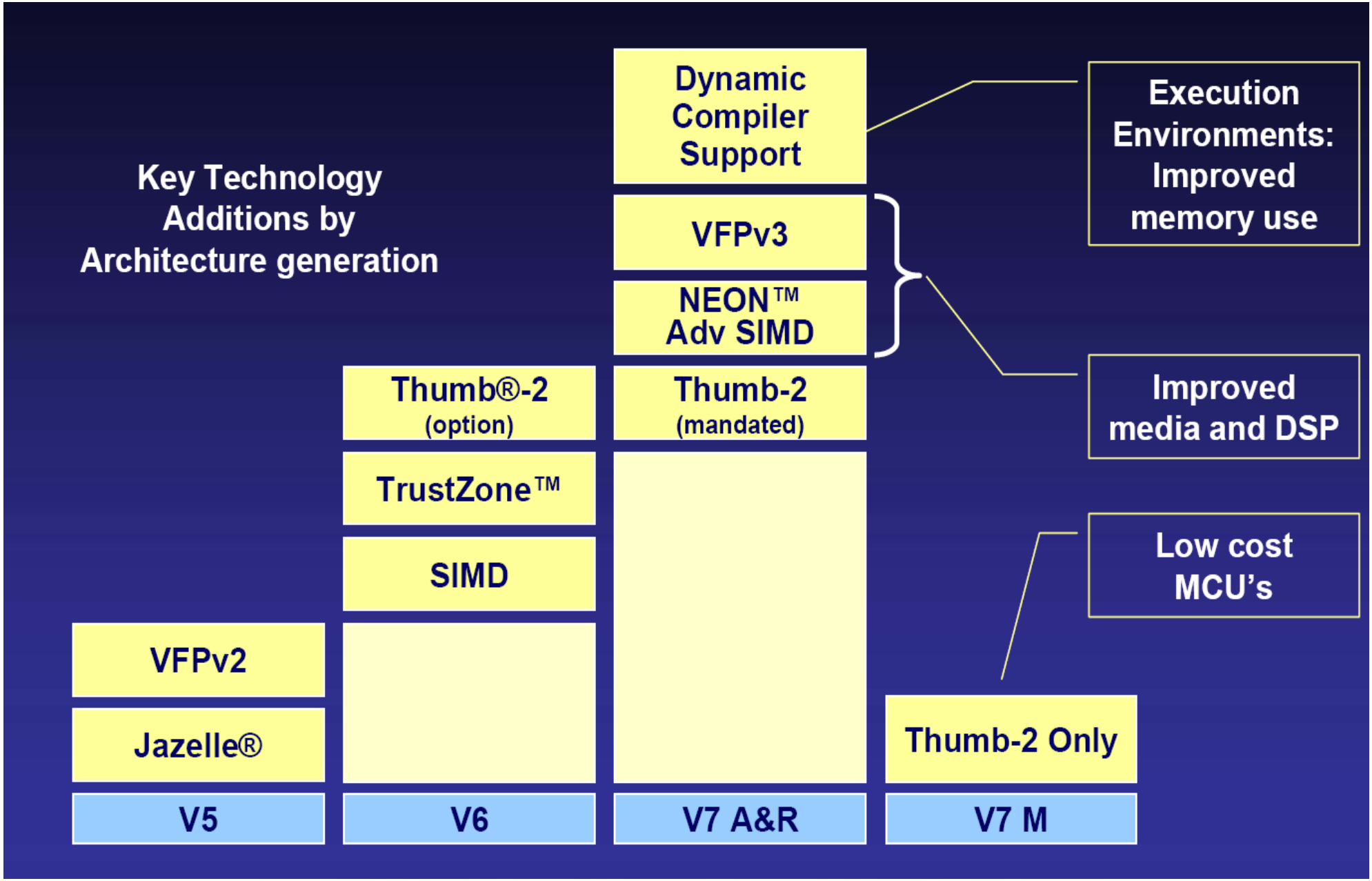
- CPU bandwidth required for various applications:
  - MPEG4 VGA decode[1] 275 MHz
  - MP3 decode, 320kbps 48kHz, worst case[2] 9.8 MHz
  - “Quake 2-like” application, CIF resolution[3] 300 MHz
  - H.264 (estimated) 350 MHz

1) MPEG-4 Simple Profile @ 30fps 512kbps , 133MHz SDRAM 10-1-1-1 memory

2) MP3 Decoder @ 320kbps 48kHz (worst case means cold start on context switch), 133MHz SDRAM 10-1-1-1 memory

3) Quake2-like simulator, full software graphics pipeline, FP implementation 133MHz SDRAM 10-1-1-1 memory

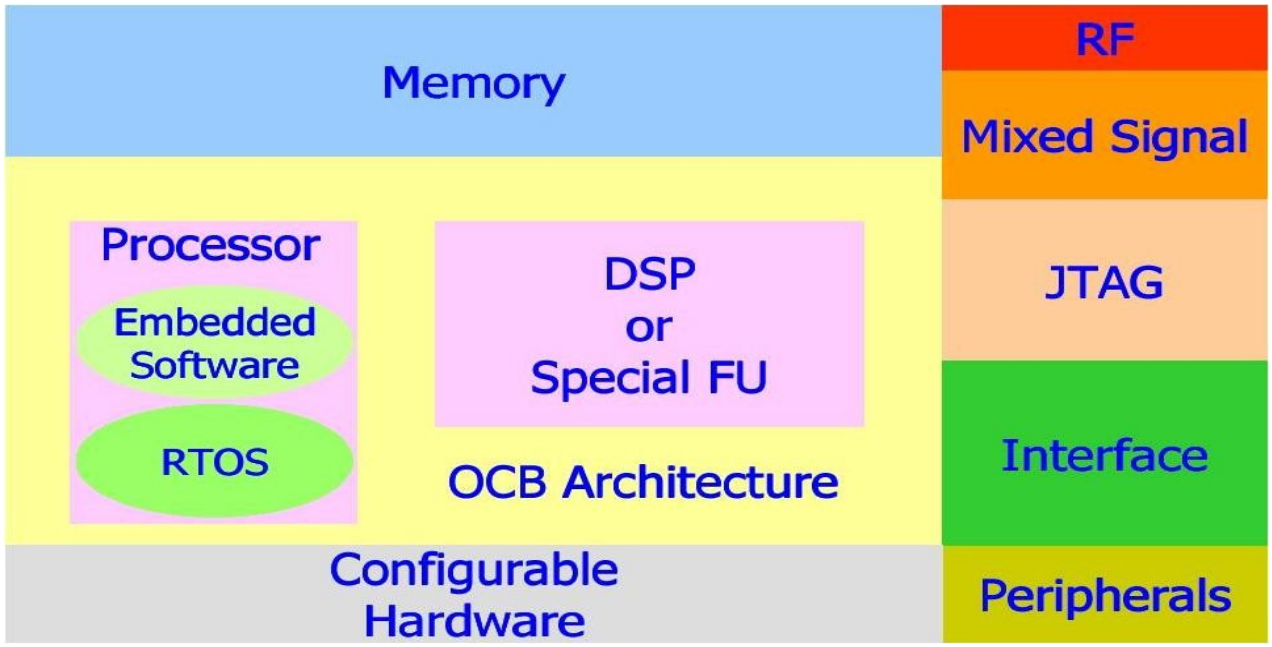
Source: ARM Developer conference


Key Technology Additions by Architecture generation
Dynamic Compiler Support
VFPv3
NEON™ Adv SIMD
Thumb®-2 (option)
Thumb-2 (mandated)
TrustZone™
SIMD
VFPv2
Jazelle®
Thumb-2 Only
V5
V6
V7 A&R
V7 M
Execution Environments: Improved memory use
Improved media and DSP
Low cost MCU's

- ARM 架構快速瀏覽
- ARM SoC 平台
- 關鍵概念：
  - 工作模式、暫存器組、系統狀態、指令集、例外處理

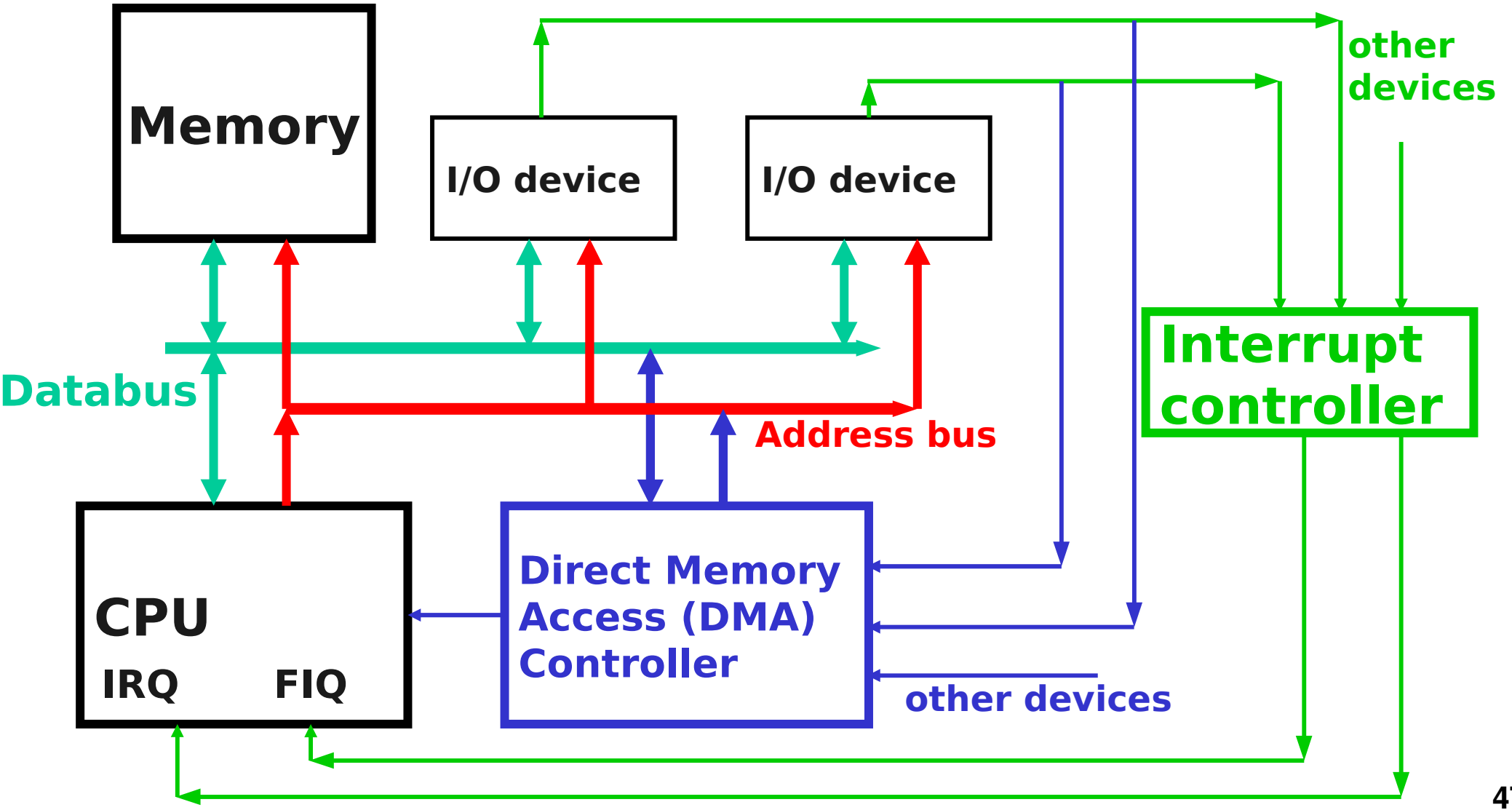
SoC (System-on-Chip)

- 整合多種不同功能的複雜 IC 組合，針對特定的市場或應用需求
- 典型的組成
  - Programmable processor
  - On-chip memory
  - HW accelerating function units (DSP)
  - Peripheral interfaces (GPIO)
  - Embedded software



典型的硬體介面

- I/O 裝置藉由 memory-mapped register 溝通
- DMA 與 interrupt 是選擇性的



CPU 與 I/O

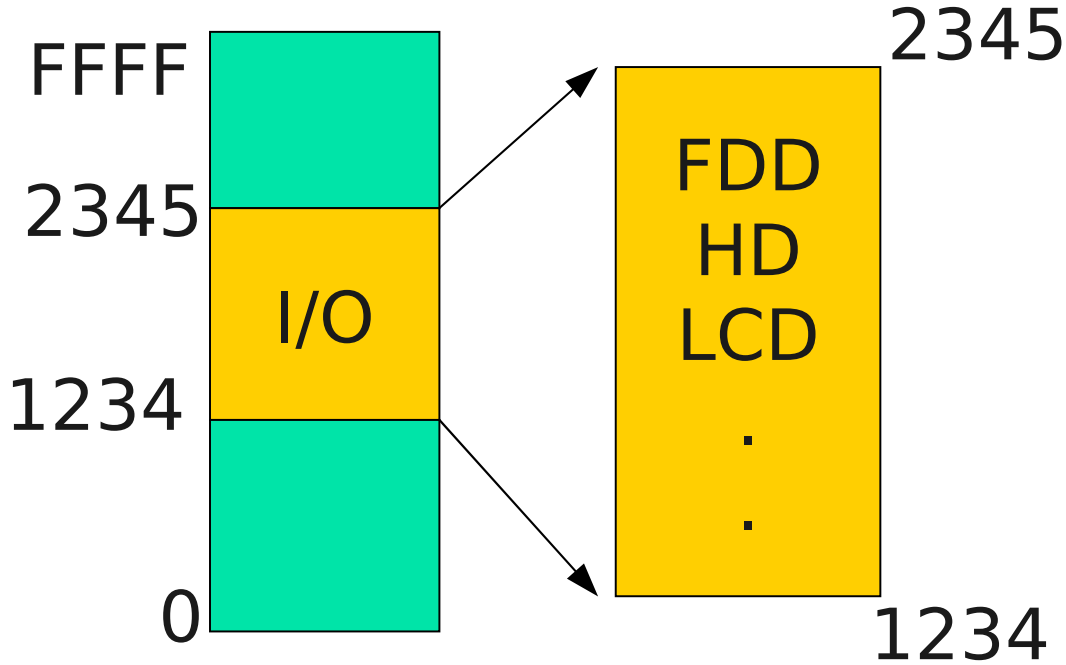
- 典型有兩種對應方式：
  - I/O mapped I/O
    - I/O 與 memory 均擁有自己的記憶體空間
    - 需要特別的指令來處理 I/O
  - Memory mapped I/O
    - I/O 與 memory 共用記憶體空間
    - 不需特別指令來處理 I/O

Memory-Mapped I/O

R1=1234

sta r0,r1 ;; 寫入 mem address 1234

out r0,r1 ;; 寫入 I/O address 1234

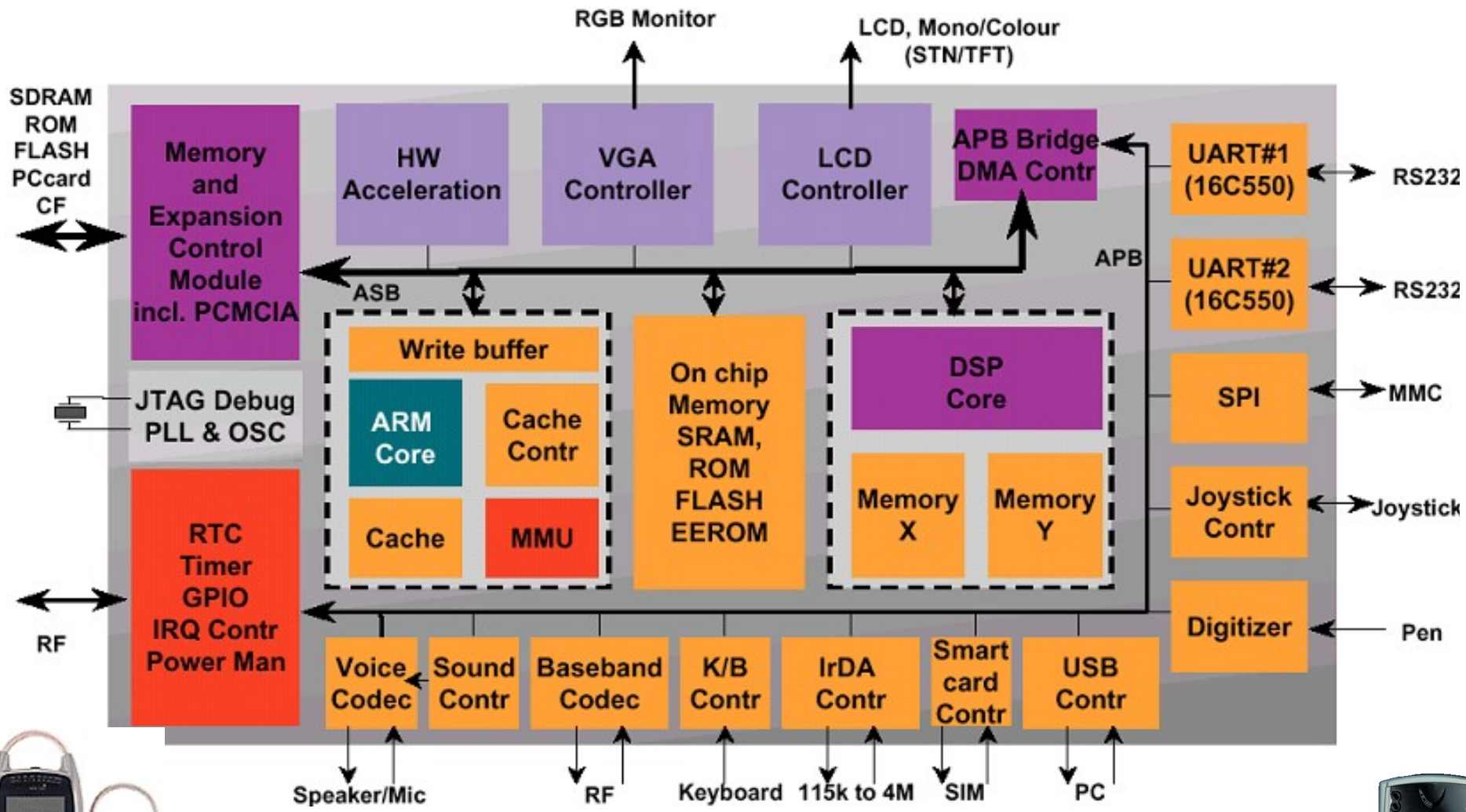


R0='A', R1=1233, R2=1234 (假設 1234 爲 LCD 的 MMIO 位址)

str r0, [r1,#0] ;;addr 1234 ='A'

str r0, [r2,#0] ;;LCD 顯示' A'

SoC 實例


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
UART#2 (16C550)
RS232
APB
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

Display Driver IC: TFT, OLED

Camera Chipset: CMOS - CCD

Connectivity: WLAN, GPS, Bluetooth

Processor: ARM

Modem: GSM/GPRS, WCDMA

RF/Analog: Rx/Tx, Zero IF

RAM: **Mobile** DRAM, SRAM, UtRAM

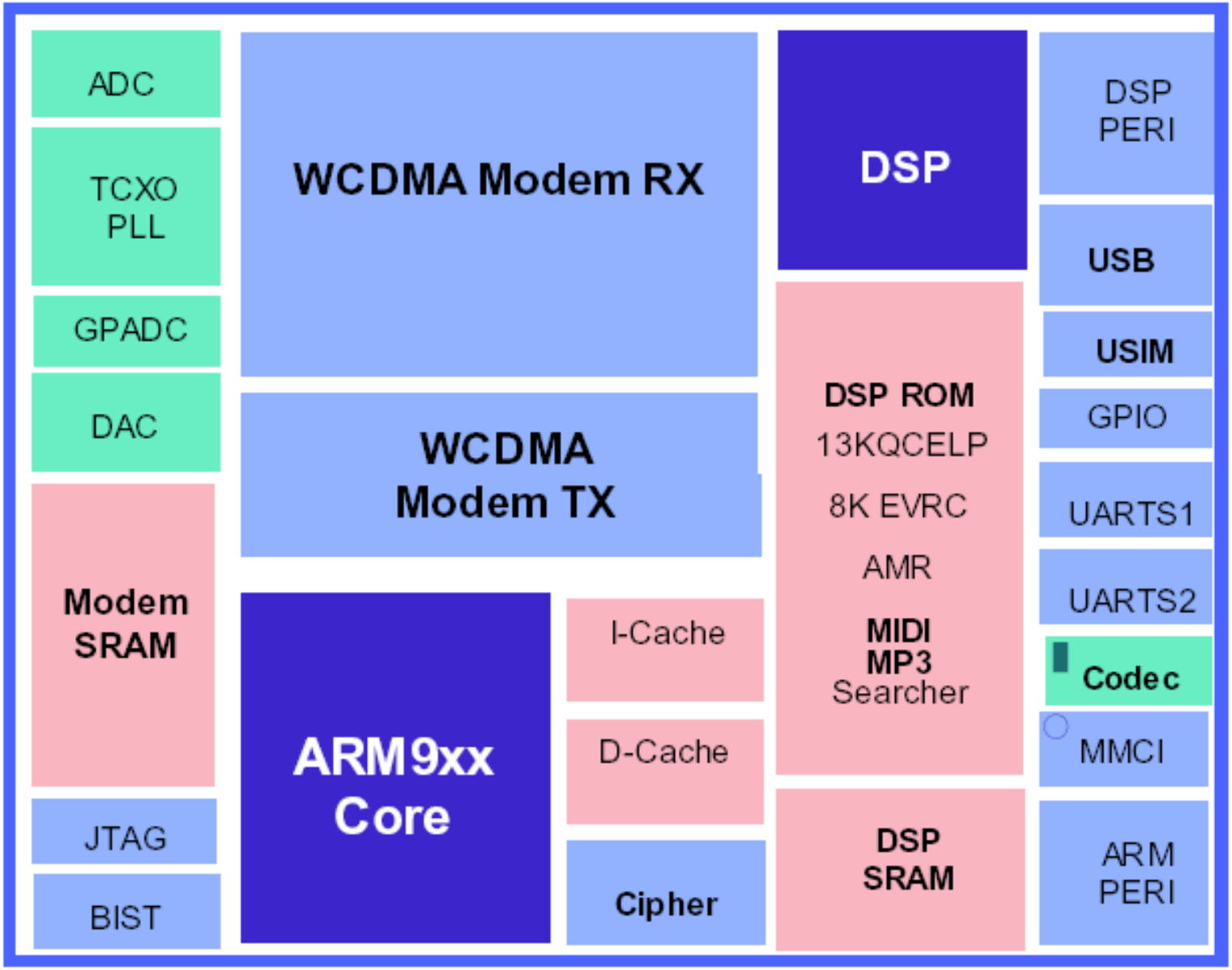
Smart Card: SIM

SIP / MCP

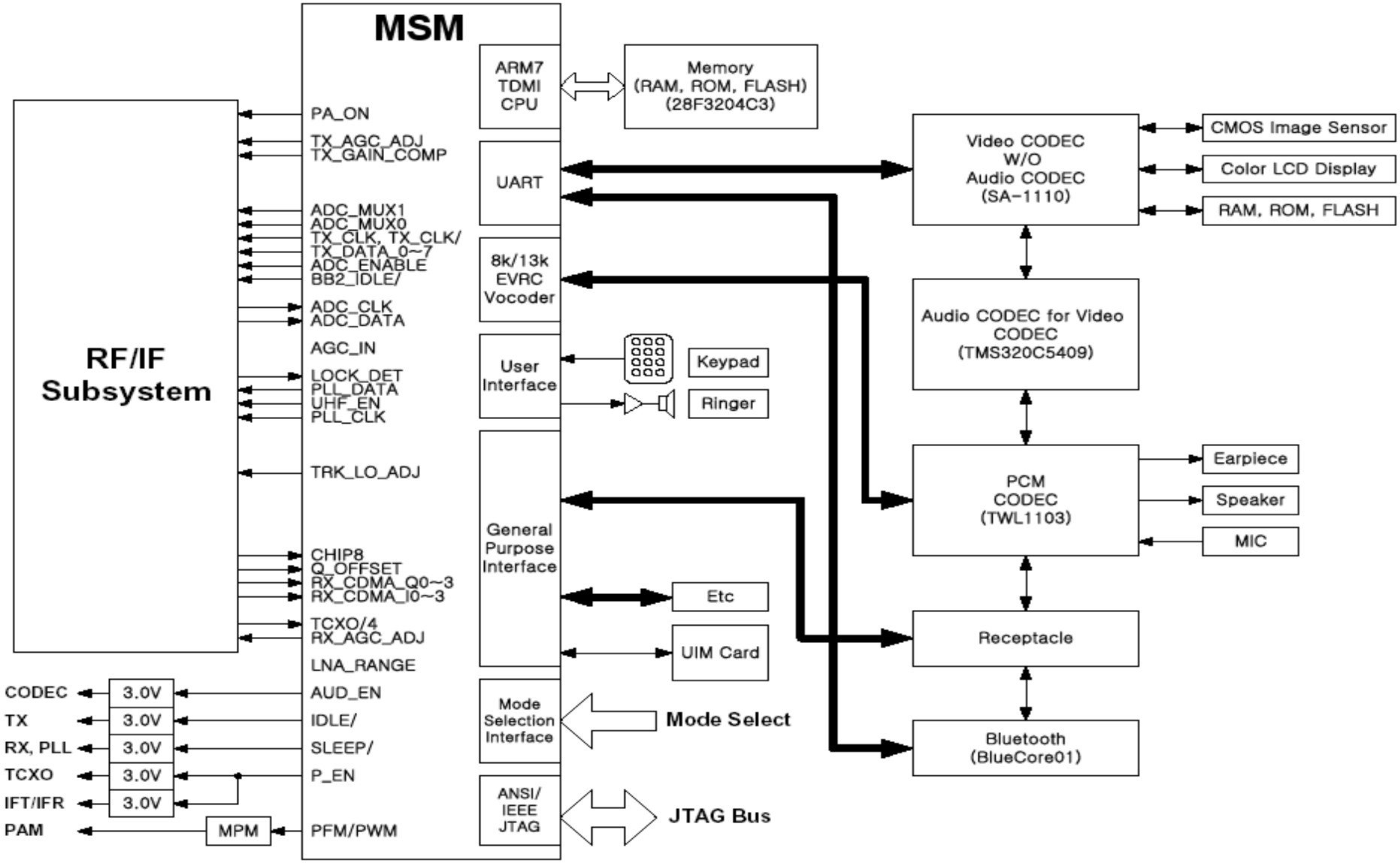
Flash Memory: Code/Data Storage

SoC

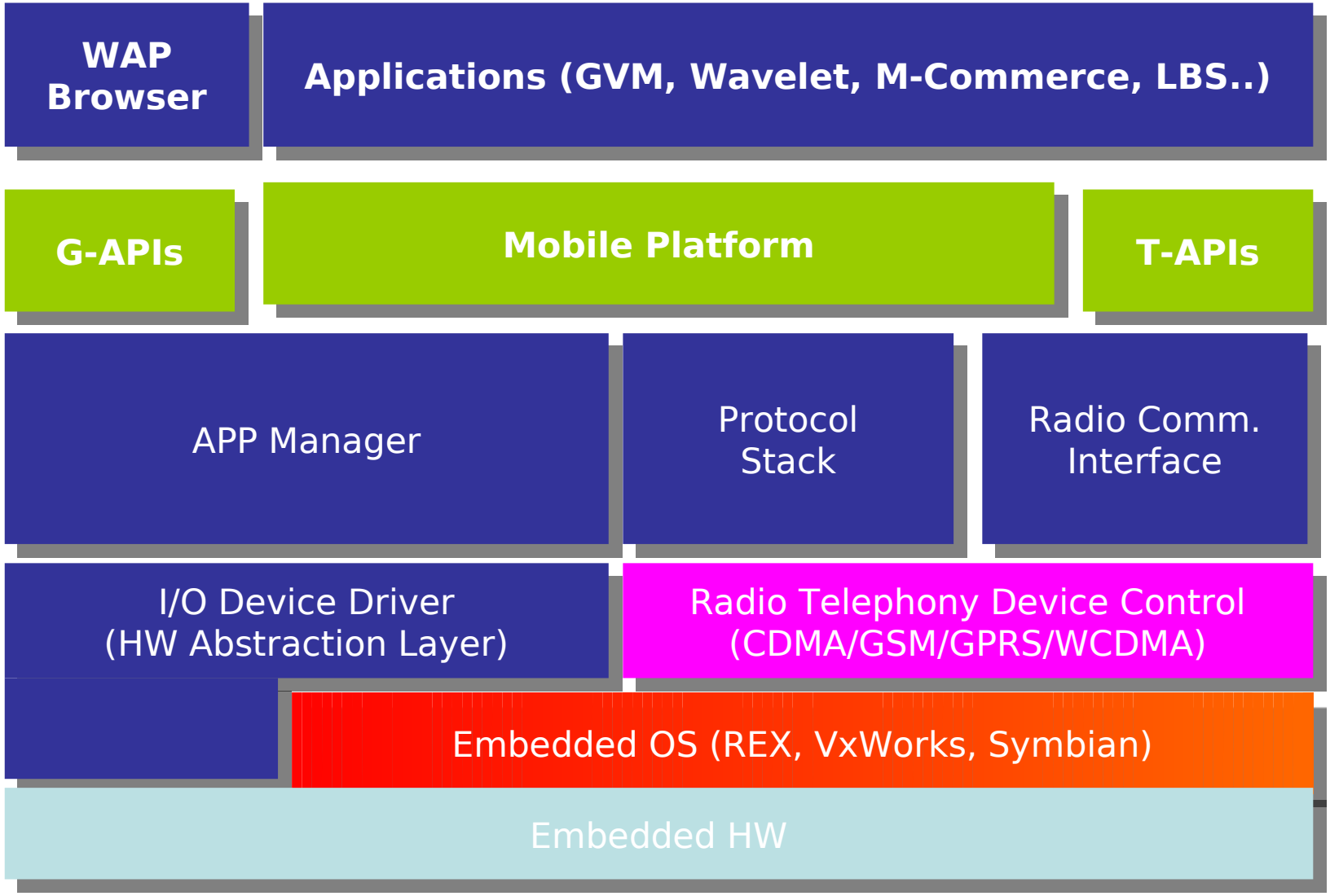
Modem Chip Evolution for Cellular Phone



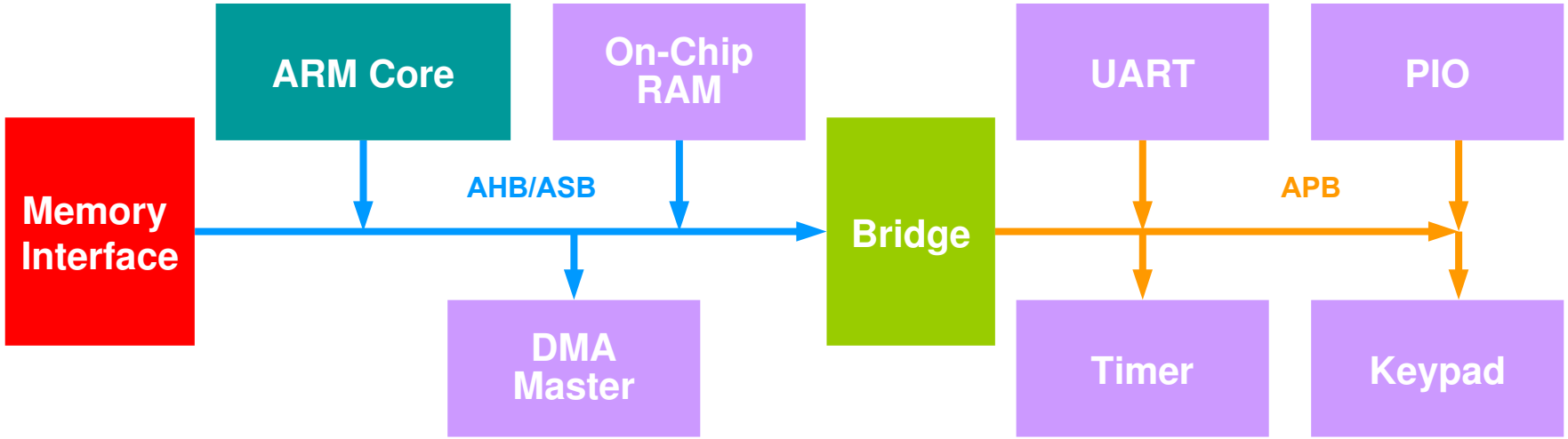
Moble station :: Baseband



Moble station :: Software



ARM On-Chip Bus



A typical AMBA system

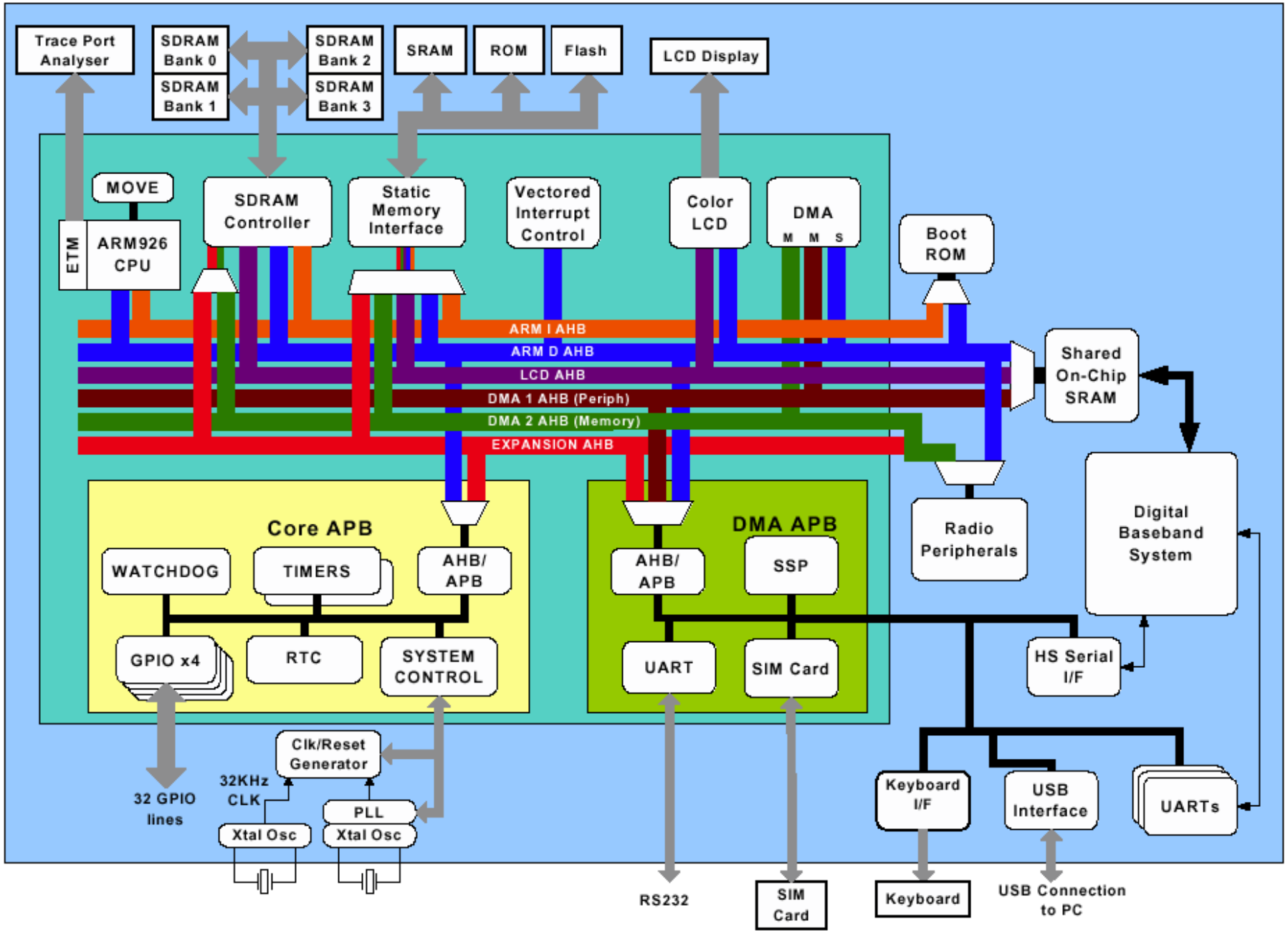
AHB: Advanced High-performance Bus

ASB: Advanced System Bus

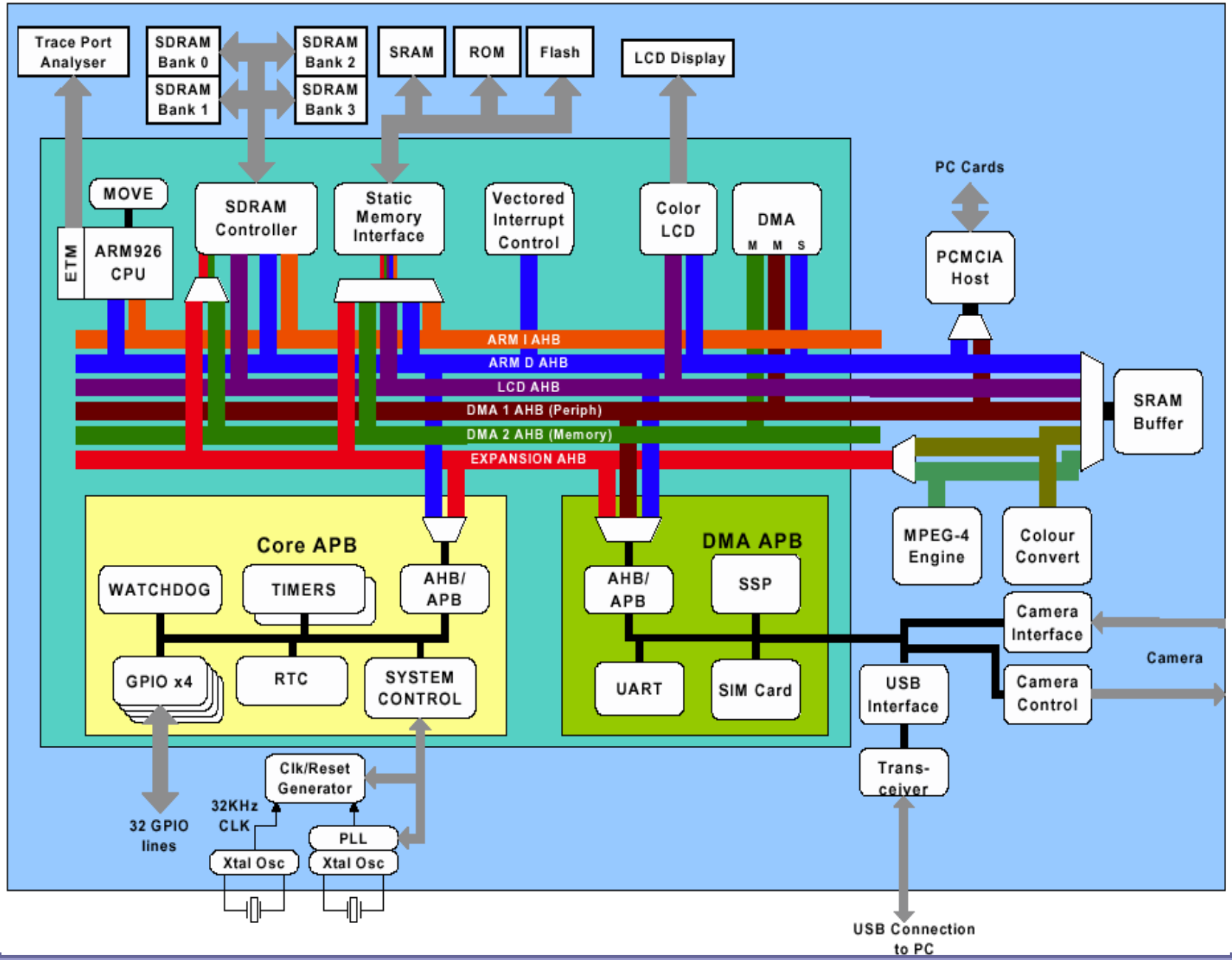
APB: Advanced Peripheral Bus

*** AMBA: Advanced Microcontroller Bus Architecture*

Example: GPRS Phone



Example: Videophone



- ARM 架構快速瀏覽
  - ARM 歷史背景
  - ARM 的「家族」
- ARM SoC 平台
- 關鍵概念：
  - 工作模式、暫存器組、系統狀態、指令集、例外處理
  - 探討 PXA255 SoC 爲例
    - 觀察 CuRT 的運作並驅動 UART 裝置

善用強大的系統模擬器 QEMU

- 不只是「窮人的 ARM 開發板」（中國北京清華大學 Skyeye 開發團隊的故事）
- 對於學習 / 分析 ARM 架構與指令集，提供一個高整合度的互動環境
  - 內建 instruction-level tracer
  - gdb server 的整合
  - 豐富的週邊硬體模擬
- Android SDK/Emulator 的 goldfish 虛擬硬體平台

後續的實驗部份皆以 QEMU 模擬硬體環境

豐富的硬體模擬環境

- Openmoko GTA01/GTA02 emulated by QEMU

  ARMv4t / Samsung S3C2410/2442

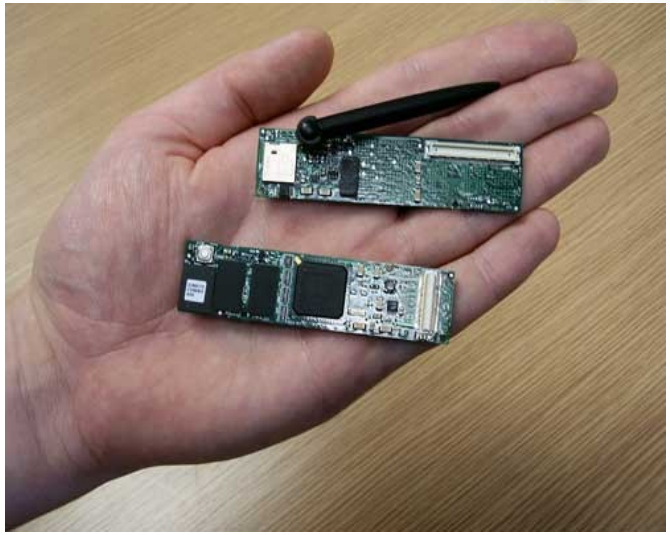
- Sharp PDAs emulated by QEMU
  ARMv5te / Marvell PXA2xx

- ARM Versatile PB board emulated by QEMU
  ARM926ej

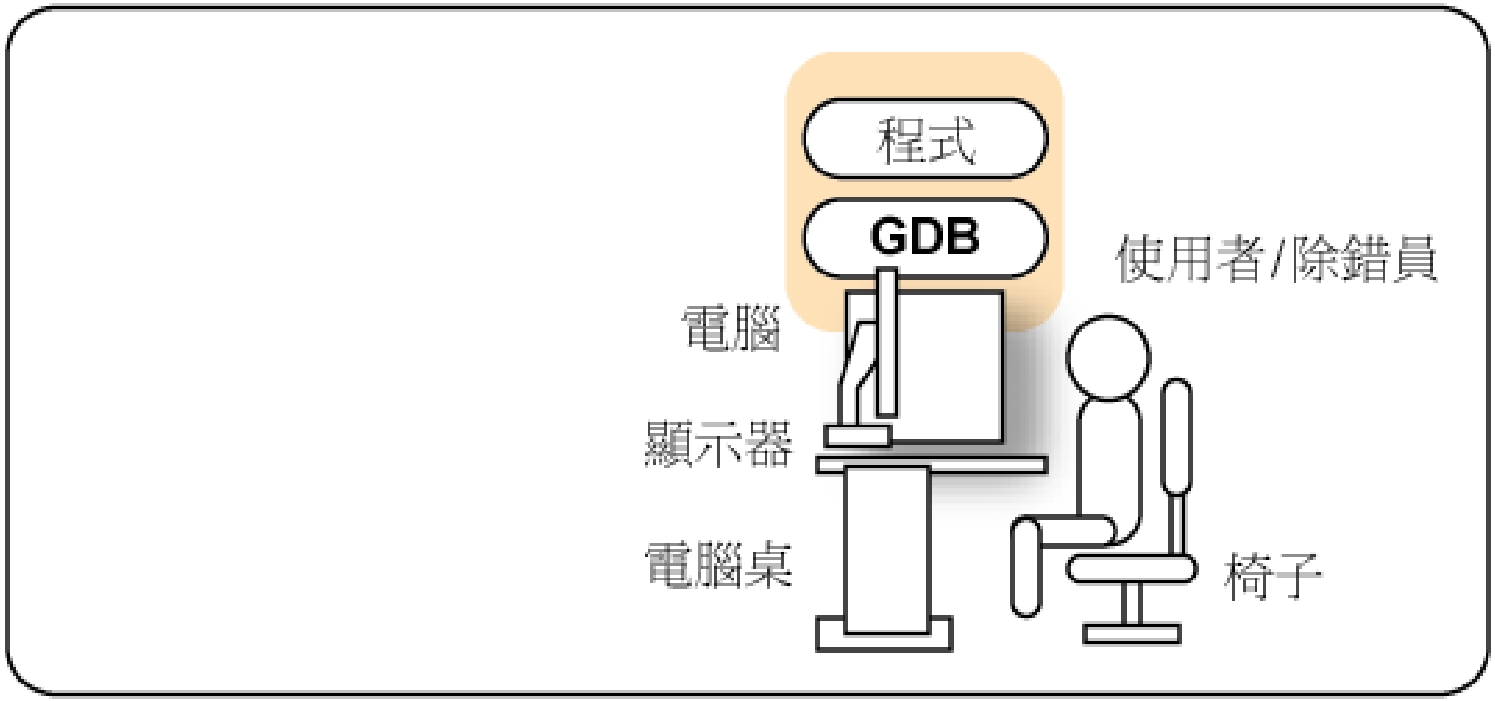
- Gumstix

  ARMv5te / Marvell PXA25x

  (hardware model: connex)

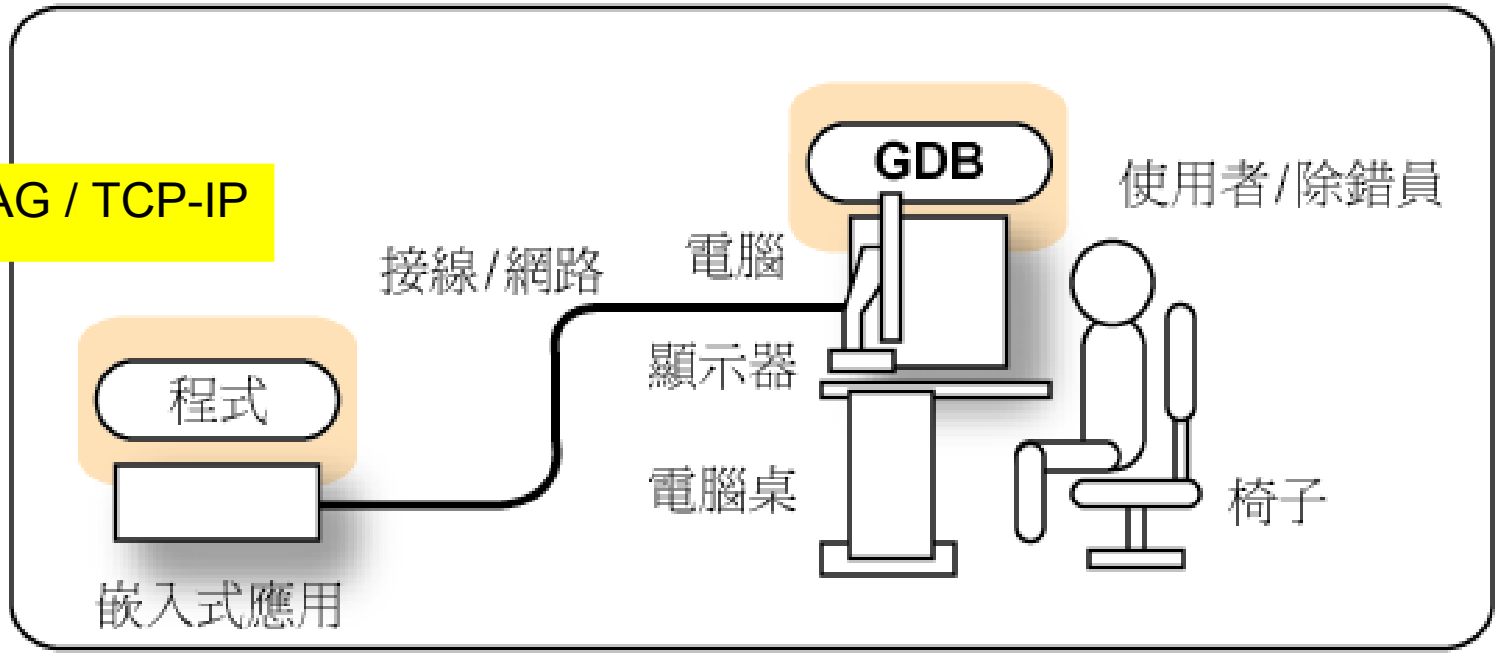


近端/本機型除錯



Emulated by QEMU

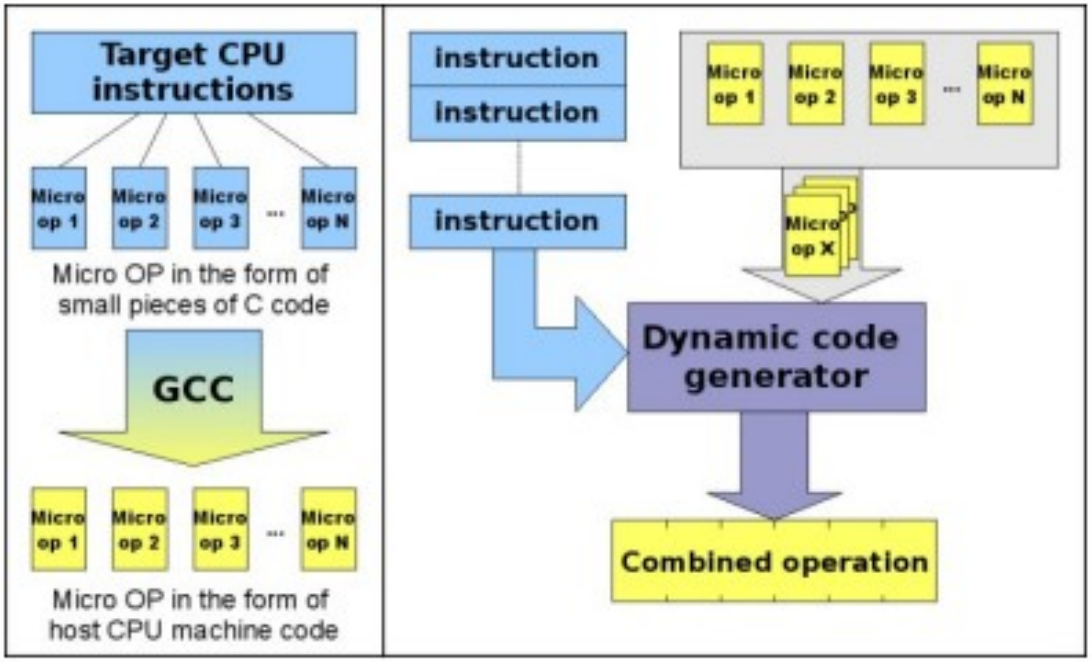
遠端/遙控型除錯



Serial / JTAG / TCP-IP

QEMU 回顧 (1)

- 快速的模擬器： Portable dynamic translator



QEMU 編譯與執行時的流程

- 完整系統模擬
  - instruction set + processor + peripherals
    硬體平台： x86, x86_64, ppc, arm, sparc, mips, nds(台灣心)
  - 指定特定機器： `qemu-system-arm -M ?`
- 兩種模擬模式： User, System

關於 QEMU 的原理，可參考拙作〈 QEMU JIT Code Generator & System Emulation 〉

QEMU 回顧 (2)

兩種執行模式

- user mode emulation ：可執行非原生架構之應用程式
  支援： x86, ppc, arm, sparc, mips
- system emulation
  - qemu linux.img
  - 也可分別指定 kernel image、 initrd，及相關參數
- 以 xscale 爲例：

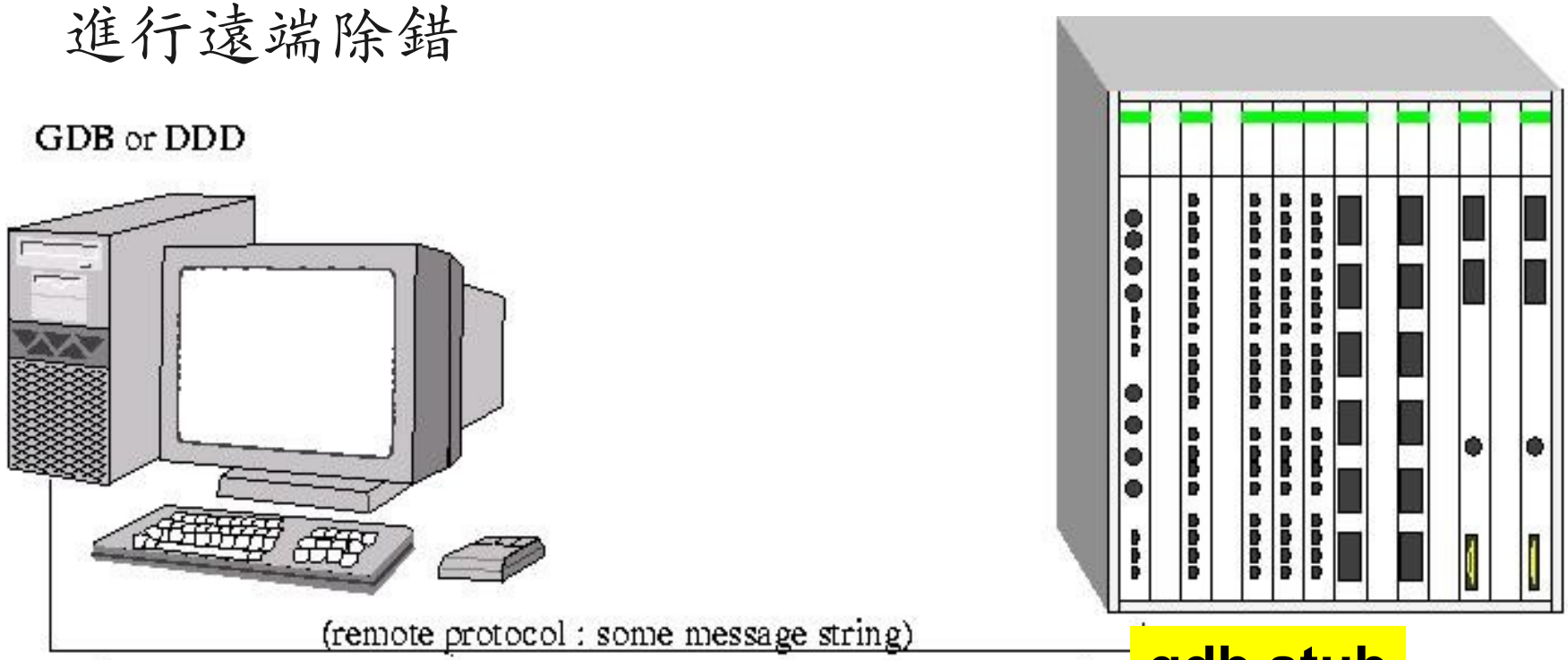
使用 **ARM target** 的 **ld-linux.so.2**

```
~/poky/build/tmp$ file ./rootfs/bin/busybox
./rootfs/bin/busybox: ELF 32-bit LSB executable, ARM, version 1 (ARM), for GNU/Linux
2.4.0, dynamically linked (uses shared libs), for GNU/Linux 2.4.0, stripped
~/poky/build/tmp$ ./qemu-arm ./rootfs/lib/ld-linux.so.2 \
--library-path ./rootfs/lib ./rootfs/bin/busybox uname -a
Linux venux 2.6.20-12-generic #2 SMP Sun Mar 18 03:07:14 UTC 2007 armv5tel
unknown
```

Processor 變成 **armv5te (Xscale)**

gdb stub

- 考慮在 system emulation 模式下，該如何喚起 gdb ？
- Remote Debugging： gdb 可透過 serial line 或 TCP/IP 進行遠端除錯



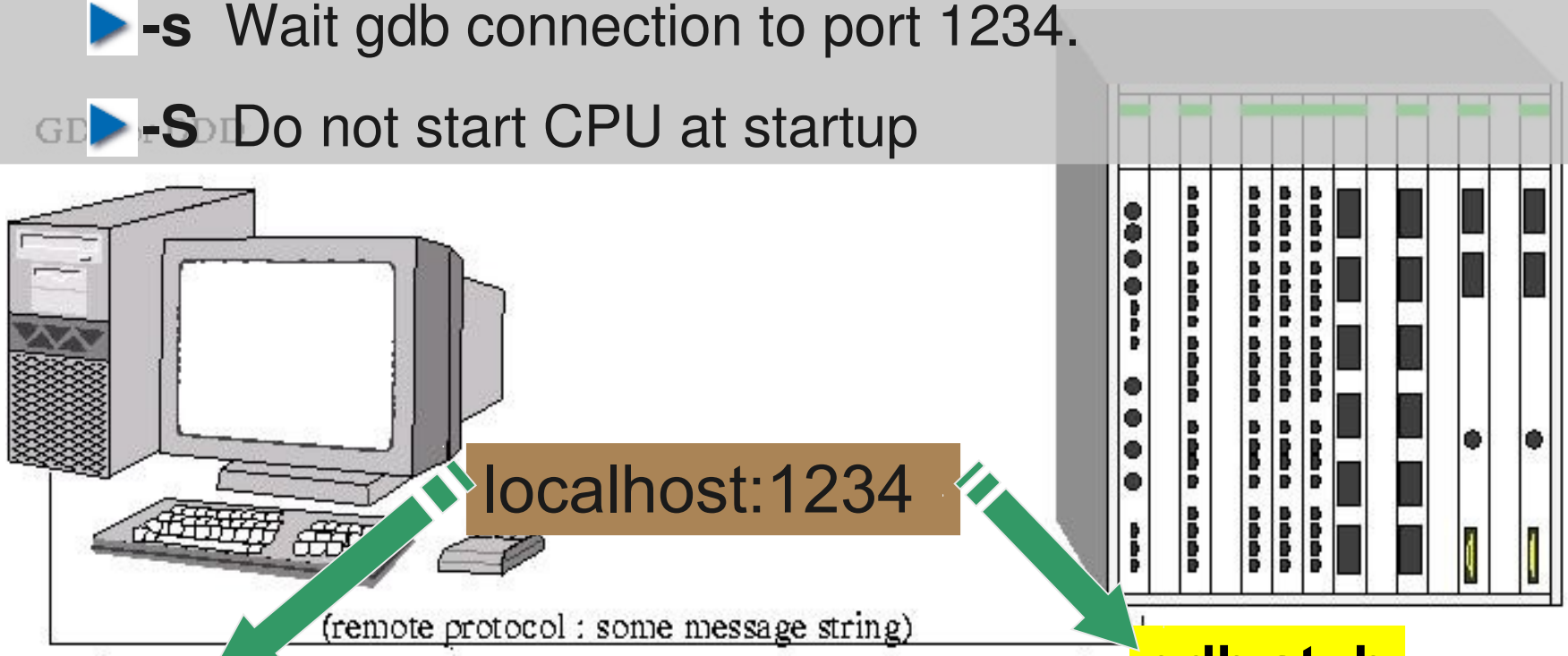
開發平台 (Host)
運作完整的 GDB

Qemu 所模擬的機器

QEMU 回顧 (4)

gdb stub ：透過 TCP/IP

- (gdb) target remote localhost:1234
- qemu 執行選項：
  - **-s** Wait gdb connection to port 1234.
  - **-S** Do not start CPU at startup



開發平台 (Host)
運作完整的 GDB

Qemu 所模擬的機器

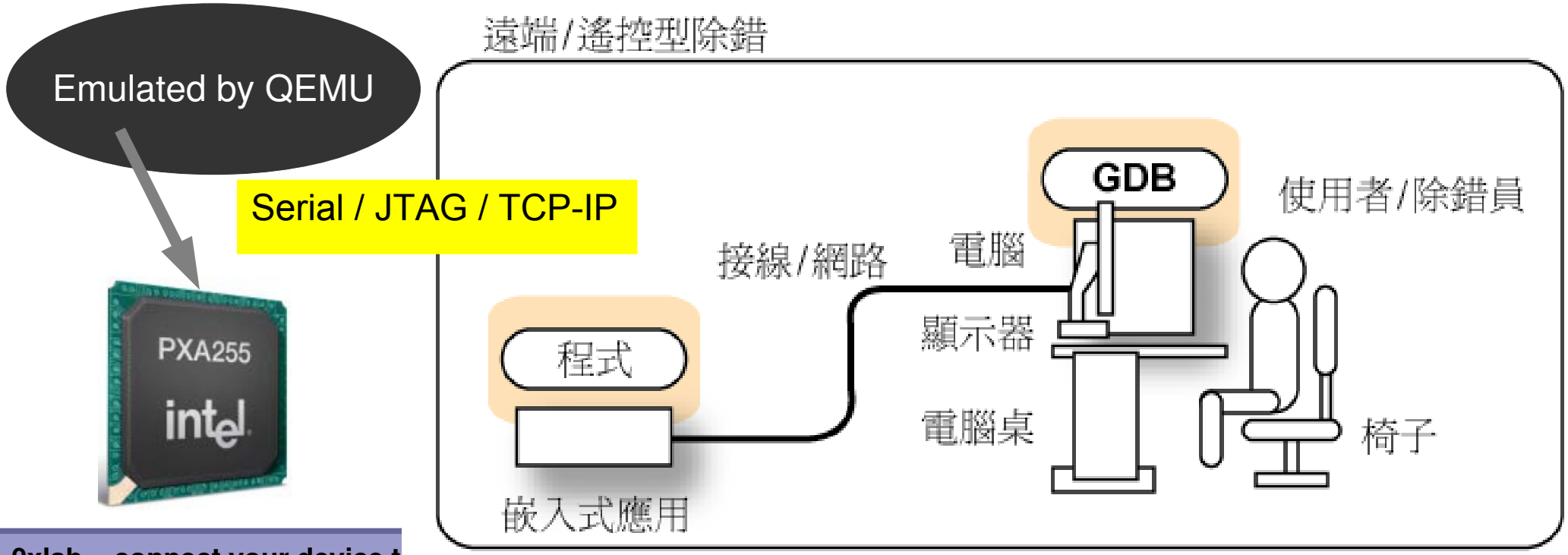
[PLAN] 將自製的 CuRT 作業系統運作於 PXA255 (emulated by QEMU)，透過 gdb stub 及 TCP/IP，與 host 端進行 Remote debugging

```
# dd of=flash-image bs=1k count=16k if=/dev/zero
# dd of=flash-image bs=1k conv=notrunc if=curt_image.bin
# qemu-system-arm -M connex -pflash flash-image -serial stdio -s -S
```

wait gdb connection to port 1234.

CuRT 後續維護 (by cyt93cs)
http://code.google.com/p/curt-v1-rework/

CuRT 原始程式碼：(BSD 授權)
http://jserv.sayya.org/kernel/



CuRT 準備動作 (1)

Applications Places System 2:31 下午

rxvt-unicode

```
jserv@venux:~/realtime/CuRT_v1/app/shell$ export PATH=/usr/local/csl/arm-2009q1/bin:$PATH
jserv@venux:~/realtime/CuRT_v1/app/shell$ make
```

取得 CodeSourcery GNU Toolchain 2009q1
解開安裝於 /usr/local/csl 目錄下
於兩個 X 終端機視窗設定必要的 $PATH 環境變數

```
# cd app/shell
# make
```

rxvt-unicode

```
jserv@venux:~$ cd realtime/CuRT_v1/app/shell/
jserv@venux:~/realtime/CuRT_v1/app/shell$ export PATH=/usr/local/csl/arm-2009q1/bin:$PATH
jserv@venux:~/realtime/CuRT_v1/app/shell$
```

```
…（參考的編譯畫面）...
arm-none-linux-gnueabi-ld -nostdlib -static -e _start -p --no-undefined -X -T ld-script.lds \
        -o curt_image.elf \
        ./main.o ../../device/serial.o ../../lib/stdio.o ../../arch/arm/mach-pxa/port.o ../../arch/arm/mach-pxa/start.o ../../arch/arm/mach-pxa/asm_port.o ../../kernel/kernel.o ../../kernel/thread.o ../../kernel/list.o ../../kernel/sync.o ../../kernel/ipc.o
arm-none-linux-gnueabi-objcopy -O binary -R .note -R .note.gnu.build-id -R .comment -S curt_image.elf curt_image.bin
#
```

檔案 curt_image.elf 爲包含除錯資訊的 ELF 執行檔

CuRT 準備動作 (2)

```
jserv@venux:~/realtime/CuRT_v1/app/shell$ ./prepare-flash
16384+0 records in
16384+0 records out
16777216 bytes (17 MB) copied, 0.213253 s, 78.7 MB/s
16+1 records in
16+1 records out
17388 bytes (17 kB) copied, 9.7359e-05 s, 179 MB/s
jserv@venux:~/realtime/CuRT_v1/app/shell$ ./run-on-connex-debug
```

```
# ./prepare_flash
# ./run-on-connex-debug
```

```
jserv@venux:~/realtime/CuRT_v1/app/shell$ cgdb -d arm-none-linux-gnueabi-gdb
```

```
# cgdb -d arm-none-linux-gnueabi-gdb curt_image.elf
```

cgdb 是 CURSES 介面撰寫的 GNU GDB 前端，有著類似 vim 的操作環境
預先載入 CuRT 的除錯資訊檔案

CuRT 準備動作 (3)

Applications Places System 3:03 下午

rxvt-unicode

```
jserv@venux:~/realtime/CuRT_v1/app/shell$ ./run-on-connex-debug
```

QEMU [Stopped]

因爲 qemu 啓動時加上 -s -S 參數，會等待 remote gdb 連線才會繼續執行

QEMU

（在 QEMU 視窗出現後，按下 Ctrl-Alt-2 組合鍵，可切換到 QEMU monitor 畫面）

rxvt-unicode

```
30  extern void SerialInit();
31
32  int main()
33  {
34 ->        SerialInit();
35           init_interrupt_control();
36           init_curt();
37
38           sem_init(&sem, 1);
/home/jserv/realtime/CuRT_v1/app/shell/main.c
There is NO WARRANTY, to the extent permitted by law.  Type "show
copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-
none-linux-gnueabi".
For bug reporting instructions, please see:
<https://support.codesourcery.com/GNUToolchain/>...
(gdb) target remote :1234
```

```
(gdb) target remote :1234
```

一旦 qemu 視窗出現，即可準備 remote gdb 連線，在 cgdb 下方視窗敲入指令

連線到 qemu 後，按下 'c' 要求繼續執行

QEMU monitor

（在 QEMU 視窗出現後，按下 Ctrl-Alt-2 組合鍵，可切換到 QEMU monitor 畫面）

```
jserv@venux:~
pxa2xx_clkpwr_write: CPU frequenc
#######################################
#       Start CuRT....
#######################################
 ID        State        Name
           Ready        idle-thre
  1        Ready        shell
  2        Running      info_
  3        Ready        stati
  4        Ready        hello
  5        Ready        hello
********** CuRT statistics i
Total Thread Count : 6
Total Context Switch Count :
Current Time Tick : 0
**********************************
$
```

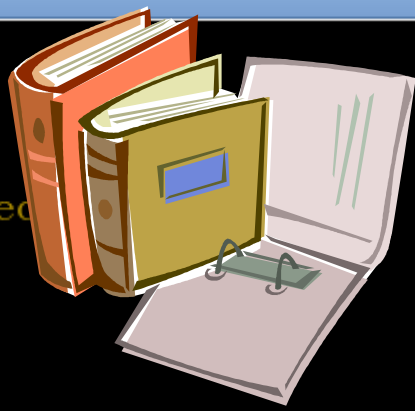
```
QEMU [Stopped]
QEMU 0.10.5 monitor - type 'help' for more information
(qemu) info registers
R00=00000000 R01=00000020 R02=00000020 R03=00000000
R04=04040404 R05=05050505 R06=06060606 R07=07070707
R08=08080808 R09=09090909 R10=10101010 R11=a000c438
R12=12121212 R13=a000c42c R14=a0001828 R15=a0000bbc
PSR=60000013 -ZC- A svc32
(qemu)
```

(qemu) **info registers**

```
 */
int SerialIsReadyChar(void)
{
        /* Make sure the data is received
        if (rFFLSR & 0x00000001)
                return 1;
        return 0;
}

/**
 * @brief Receives a character from serial
 * @retval
```

CuRT_v1/device/serial.c



在 gdb 中按下 Ctrl-C 組合鍵以搶得控制權

```
static void shell_thread_func(void *pdata){
    …
  While (1) {
        ...
        gets(buf);
```

```
Program received signal SIGINT, Interrupt.
SerialIsReadyChar () at ../../device/serial.c:81
(gdb)
```

Debugger

QEMU monitor

```
(qemu) info registers
R00=00000000 R01=00000020 R02=00000020 R03=00000000
R04=04040404 R05=05050505 R06=06060606 R07=07070707
R08=08080808 R09=09090909 R10=10101010 R11=a000c438
R12=12121212 R13=a000c42c R14=a0001828 R15=a0000bbc
PSR=60000013 -ZC- A svc32
```

r0	r8
r1	r9
r2	r10
r3	r11
r4	r12
r5	r13
r6	r14
r7	r15 (PC)

ARM Registers

CPSR

31 0

– Z C –

QEMU monitor

```
jserv@venux:~/realtime/CuRT_v1/ap
pxa2xx_clkpwr_write: CPU frequenc
#######################################
#       Start CuRT....
#######################################
 ID        State        Name
           Ready        idle-thre
  1        Ready        shell_thr
  2        Running      info_thre
  3        Ready        stati
  4        Ready        hello
  5        Ready        hello
********** CuRT
Total Thread Coun
Total Context Swi
Current Time Tick
**********
$
```

```
QEMU 0.10.5 monitor - type 'help' for more information
(qemu) info registers
R00=00000000 R01=00000020 R02=00000020 R03=00000000
R04=04040404 R05=05050505 R06=06060606 R07=07070707
R08=08080808 R09=09090909 R10=10101010 R11=a000c440
R12=12121212 R13=a000c43c R14=a0001828 R15=a0000b88
PSR=60000013 -ZC- A svc32
(qemu) info registers
R00=00000000 R01=00000020 R02=00000020 R03=00000000
R04=04040404 R05=05050505 R06=06060606 R07=07070707
R08=08080808 R09=09090909 R10=10101010 R11=a000c438
R12=12121212 R13=a000c42c R14=a0001828 R15=a0000bbc
PSR=60000013 -ZC- A svc32
(qemu)
```

Processor Mode
svc : supervisor

```
77  {
78          /* Make sure the data is received. */
            rFFLSR & 0x00000001)
                return 1;
            rn 0;

83
84  /**
85   * @brief Receives a character from erial device
/home/jserv/realtime/CuRT_v1/device/ ial.c
(gdb) c
Continuing.

Program received signal SIGINT, Interrupt.
SerialIsReadyChar () at ../../device/serial.c:81
(gdb) p $pc
$1 = (void (*)()) 0xa0000bbc <SerialIsReadyChar+52>
(gdb)
```

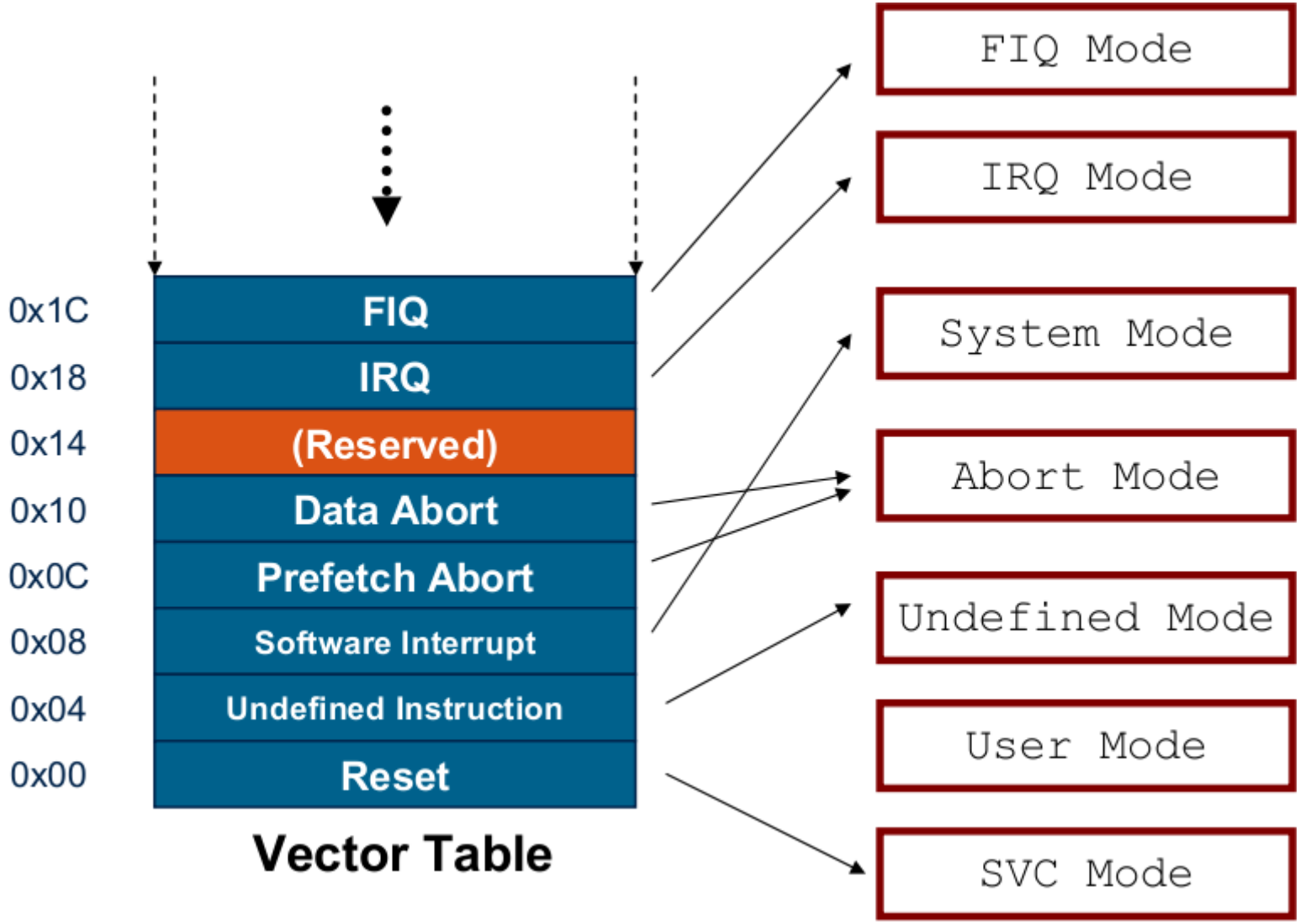
ARM Regs

r0	r8
r1	r9
r2	r10
r3	r11
r4	r12
r5	r13
r6	r14
r7	r15 (PC)

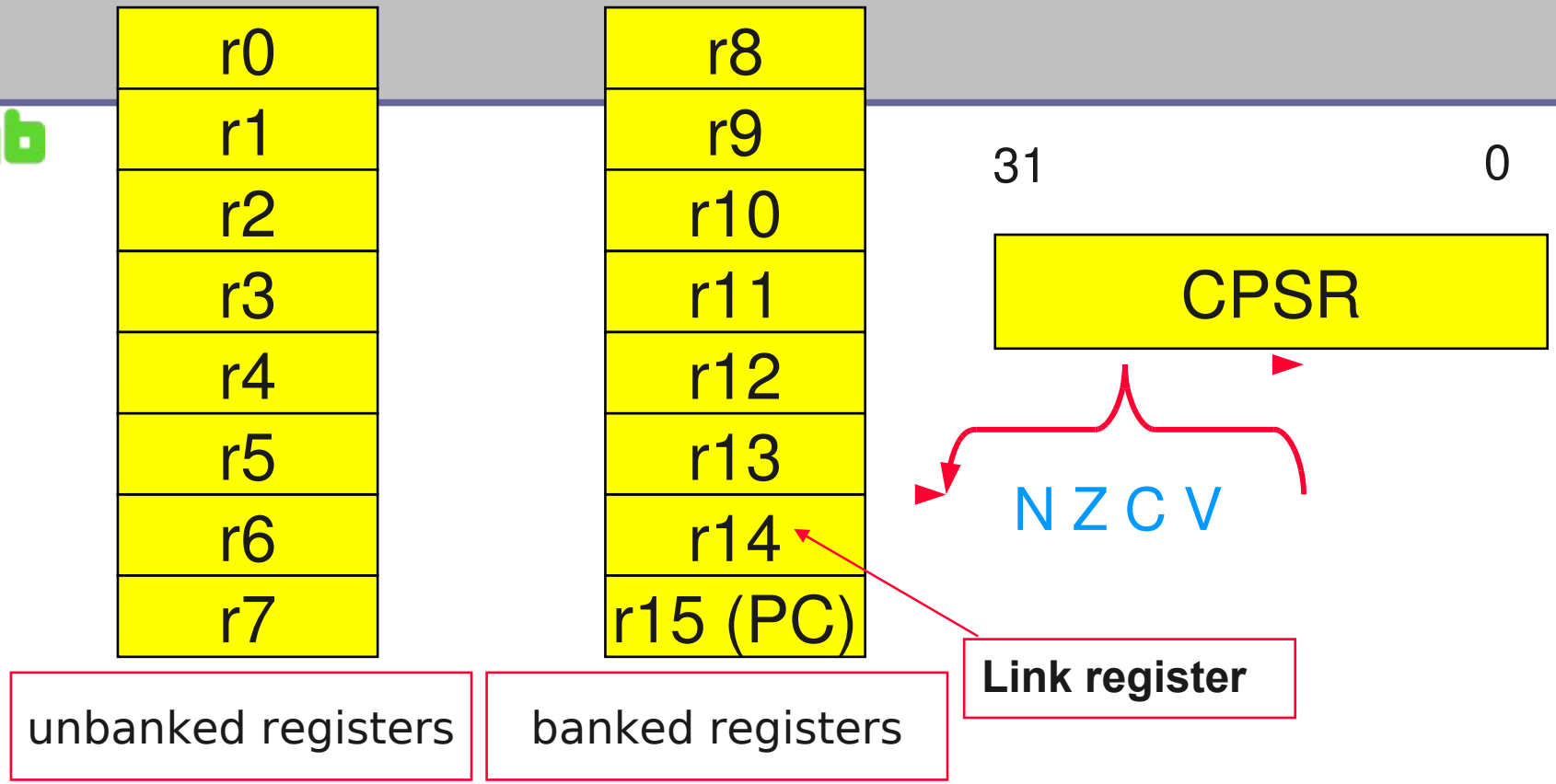
Basic Processor Modes

- **User** (usr) – Normal program execution modes
- **FIQ** (fiq) – Support a high-speed data transfer or channel process
- **IRQ** (irq) – Used for general-purpose interrupt handling
- **Supervisor** (svc) – A protected mode for OS entered on reset and when a Software Interrupt instruction executed.
- **Abort** (abt) – Implements VM and/or memory protection
- **Undefined** (und) – Support software emulation of HW coprocessors
- **System**: sys – Run privileged OS tasks

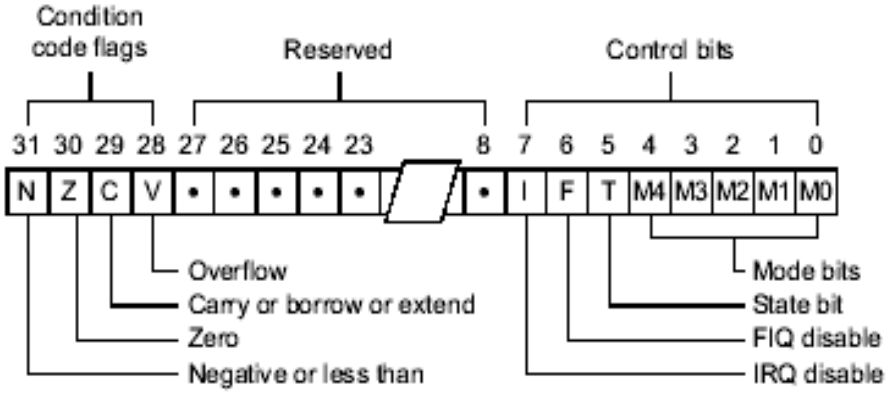
fiq, irq, svc, abt, und – *exception modes*



Interrupt 是 Exception 的特例



- Every arithmetic, logical, or shifting operation may set CPSR (*current program statues register*) bits:
  - N (negative), Z (zero), C (carry), V (overflow).
- Examples:
  - -1 + 1 = 0: NZCV = 0110.
  - $2^{31}-1+1 = -2^{31}$: NZCV = 0101.



ARM Register Set

Current Visible Registers

Abort Mode

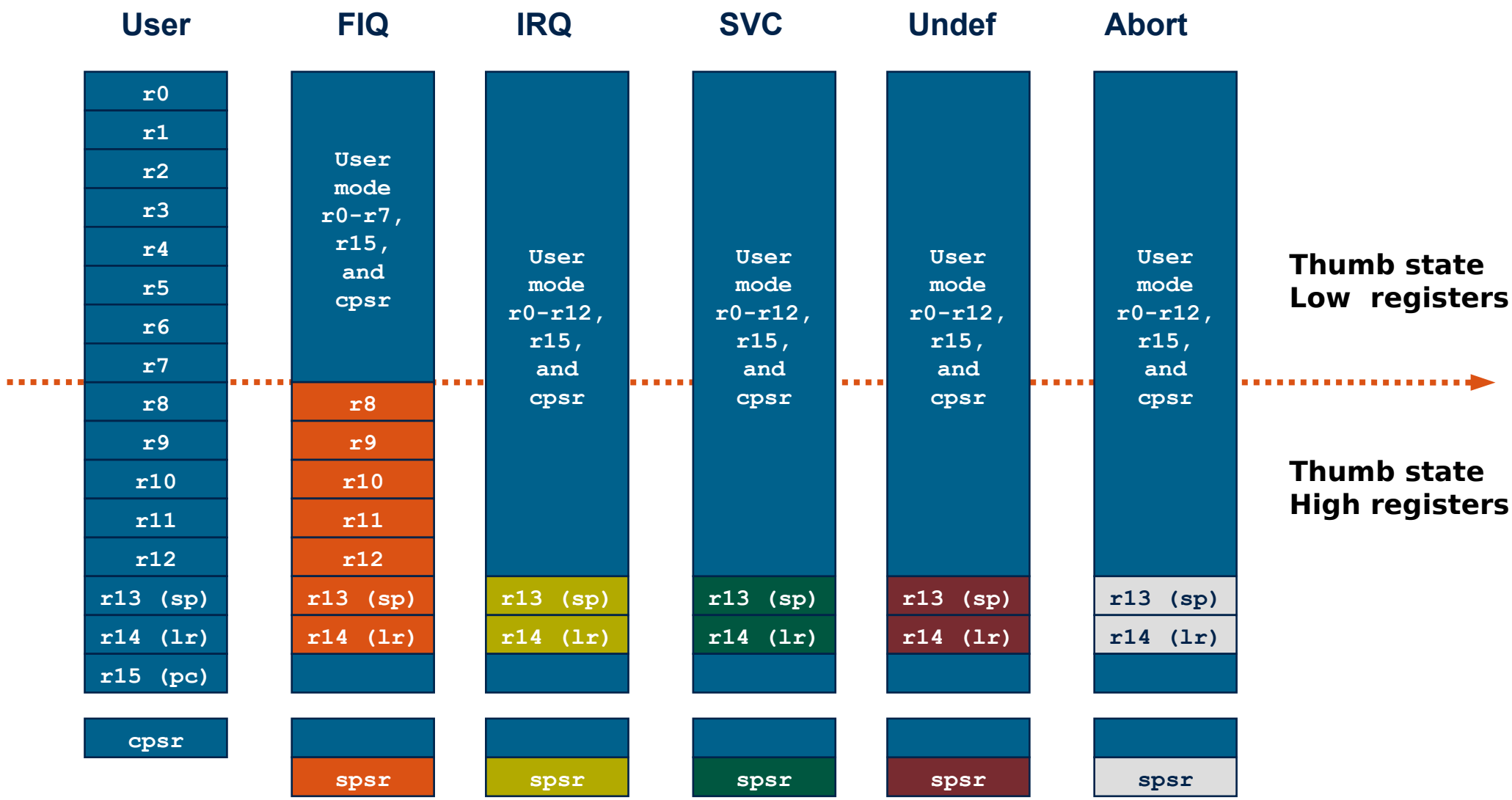
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
r13 (sp)
r14 (lr)
r15 (pc)

cpsr
spsr

Banked out Registers

User	FIQ	IRQ	SVC	Undef
	r8			
	r9			
	r10			
	r11			
	r12			
r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)	r13 (sp)
r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)	r14 (lr)
	spsr	spsr	spsr	spsr

ARM Register Set – All modes



Note: System mode uses the User mode register set

ARM Registers

- ARM 有 37 個暫存器，皆爲 32-bit 長度
  - 1 個專屬的 PC (program counter)
  - 1 個專屬的 CPSR (current program status register)
  - 5 個專屬的 SPSR (saved program status registers)
  - 30 個通用的暫存器
- 規劃
  - 通用組 : r0-r12 registers
  - 挪作特別使用 : r13 (stack pointer, sp), r14 (link register, lr)
  - program counter, r15 (pc)

故實際僅有 16 個可見的暫存器

IRQ and FIQ

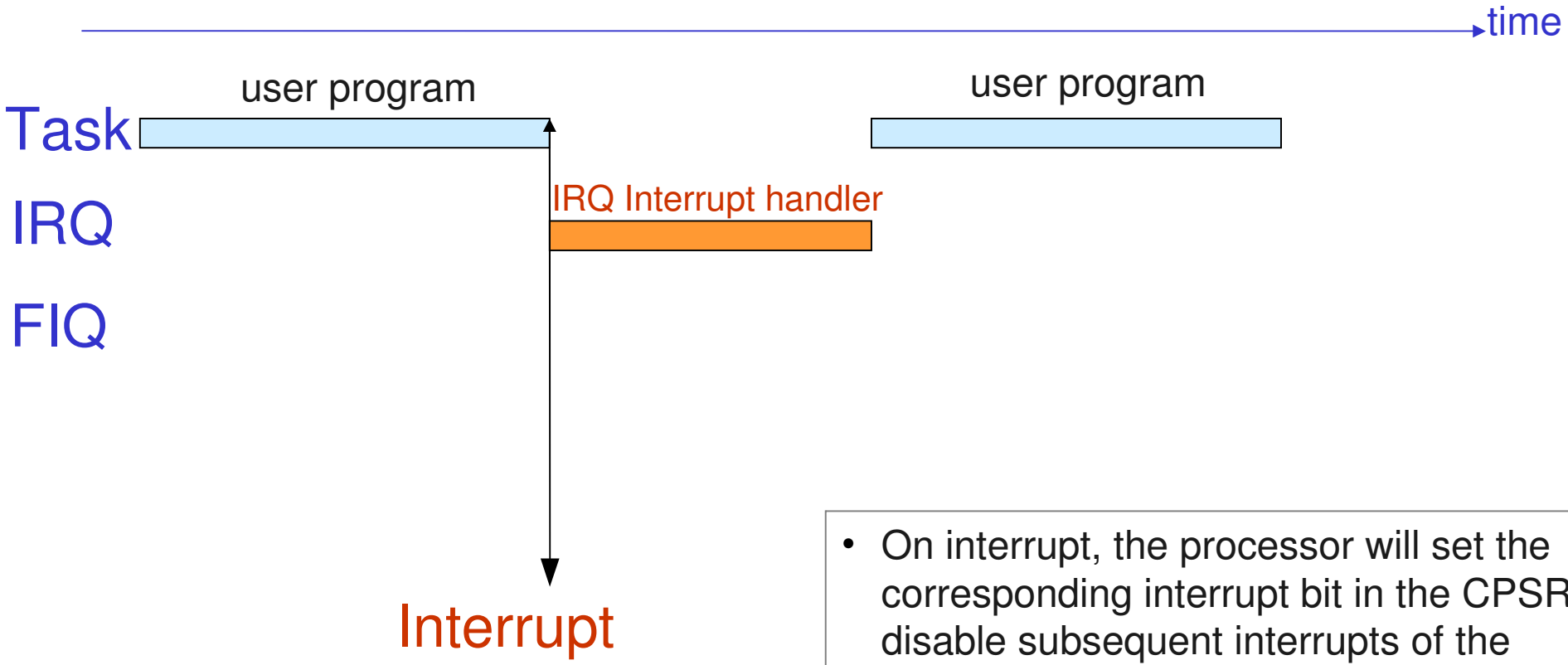
- Program Status Register

31	30	29	28	27	…	8	7	6	5	4	3	2	1	0
N	Z	C	V					I	F		M4	M3	M2	M1	M0

  - 若要抑制 interrupts，將 "F" 或 " I" bit 設定爲 1
- 一旦 interrupt 觸發，處理器將變更至 FIQ32_mode registers 或 IRQ32_mode registers
  - Switch register banks
  - Copies CPSR to SPSR_mode (saves mode, interrupt flags, etc.)
  - Changes the CPSR mode bits (M[4:0])
  - Disables interrupts
  - Copies PC to R14_mode (to provide return address)
  - Sets the PC to the vector address of the exception handler

Interrupt Handlers

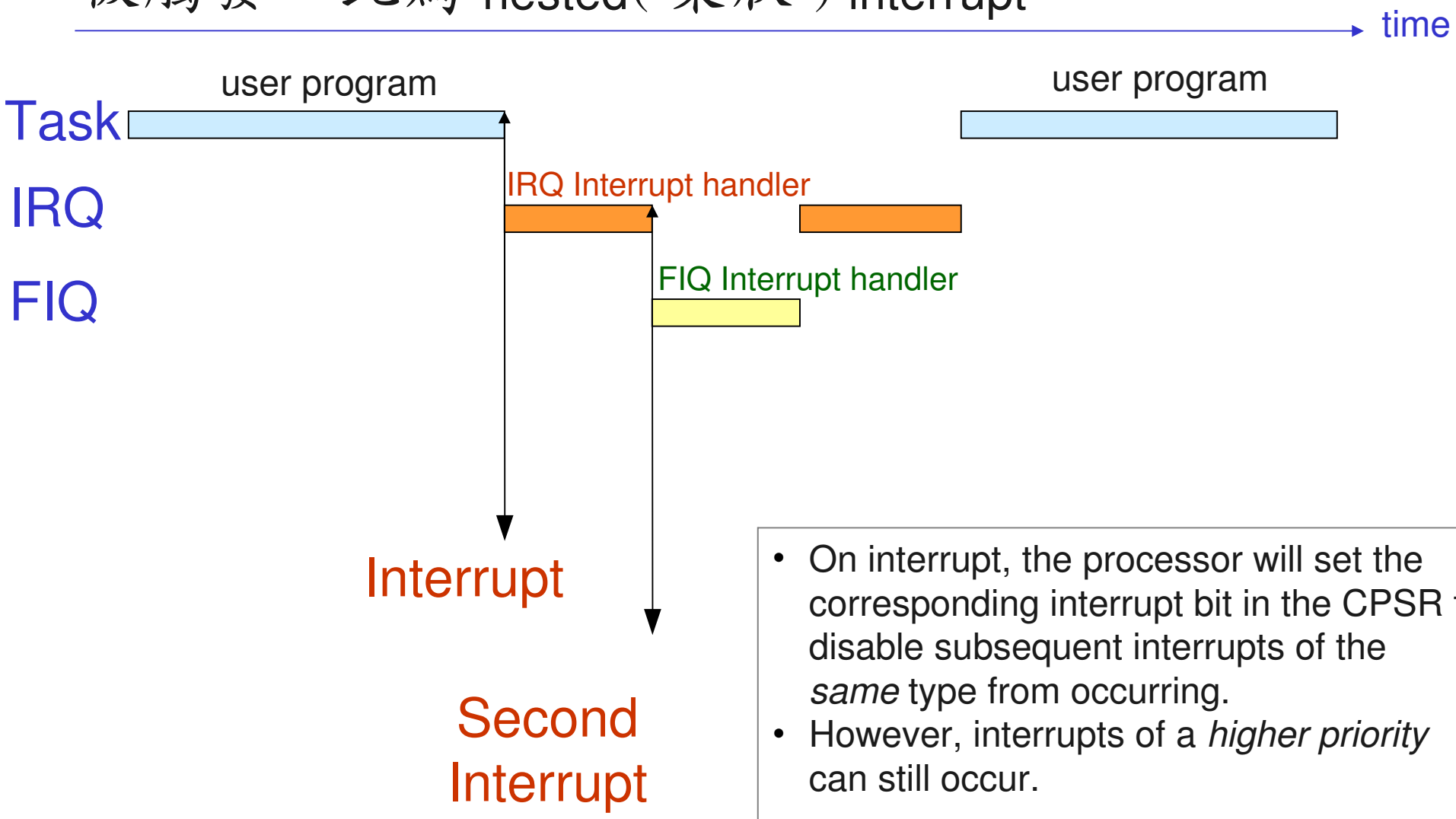
- 當 interrupt 發生時，硬體會跳躍到 interrupt handler



- On interrupt, the processor will set the corresponding interrupt bit in the CPSR to disable subsequent interrupts of the *same* type from occurring.
- However, interrupts of a *higher priority* can still occur.

Nested/Re-entrant Interrupts

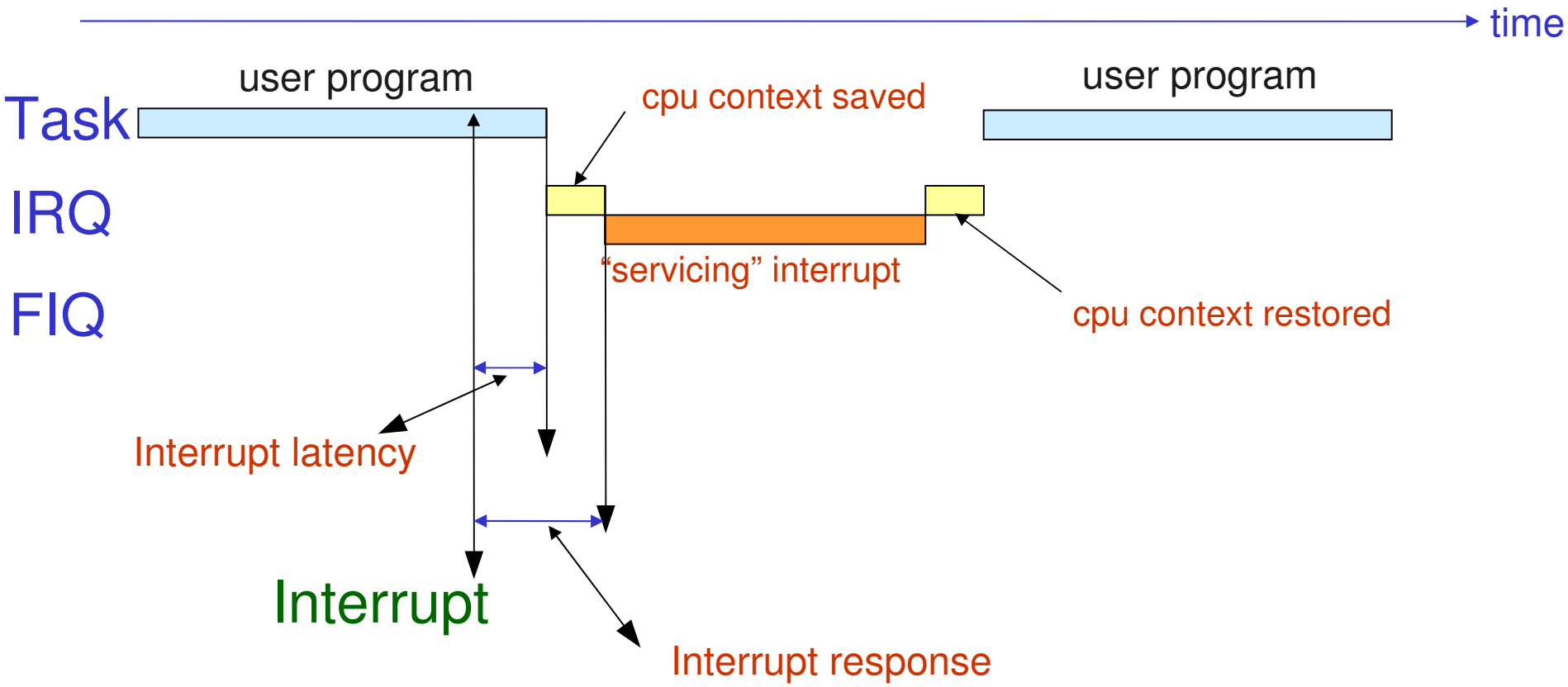
- 但是， interrupts 也可能在執行 interrupt handlers 時被觸發，此爲 nested(巢狀) interrupt



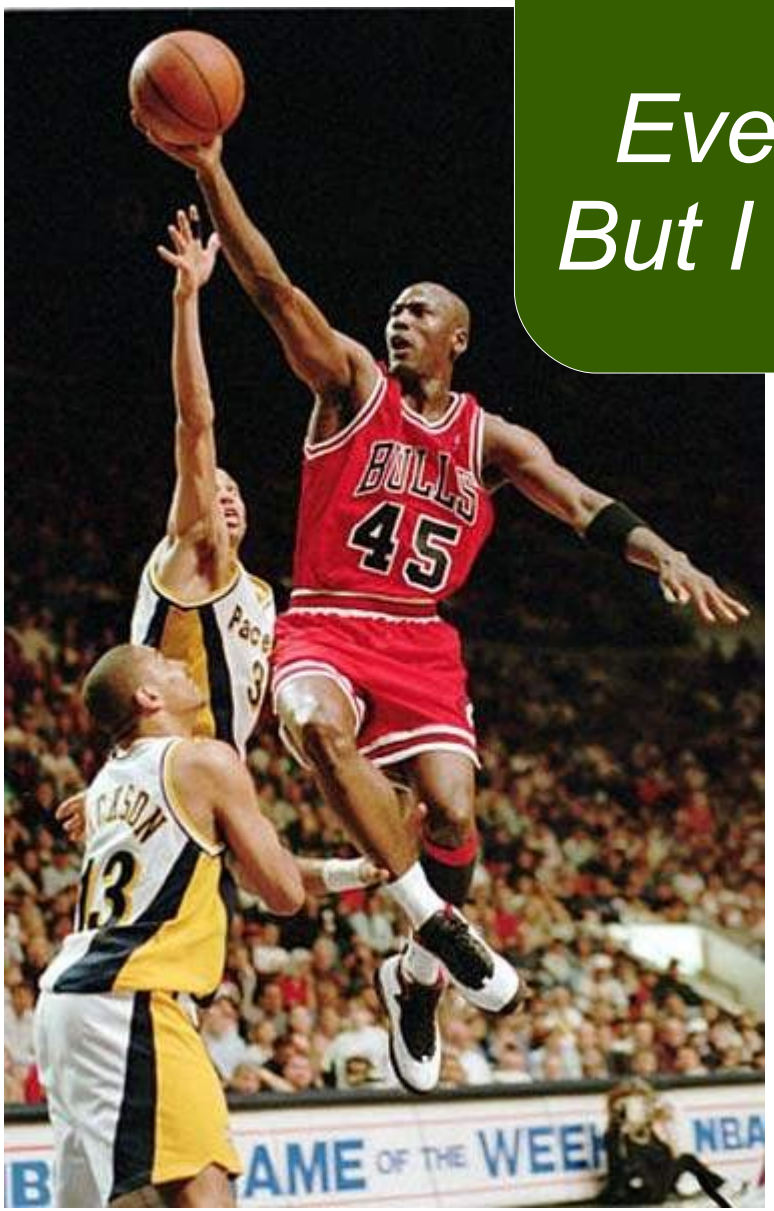
- On interrupt, the processor will set the corresponding interrupt bit in the CPSR to disable subsequent interrupts of the *same* type from occurring.
- However, interrupts of a *higher priority* can still occur.

Timing of Interrupts

- 在 interrupt handler 實際運作前，必須保存目前程式 (context) 的 register (若觸及這些 register)
- 這也是何以 FIQ 需要額外 register 的緣故，爲了降低 CPU 保存 context 的成本開銷



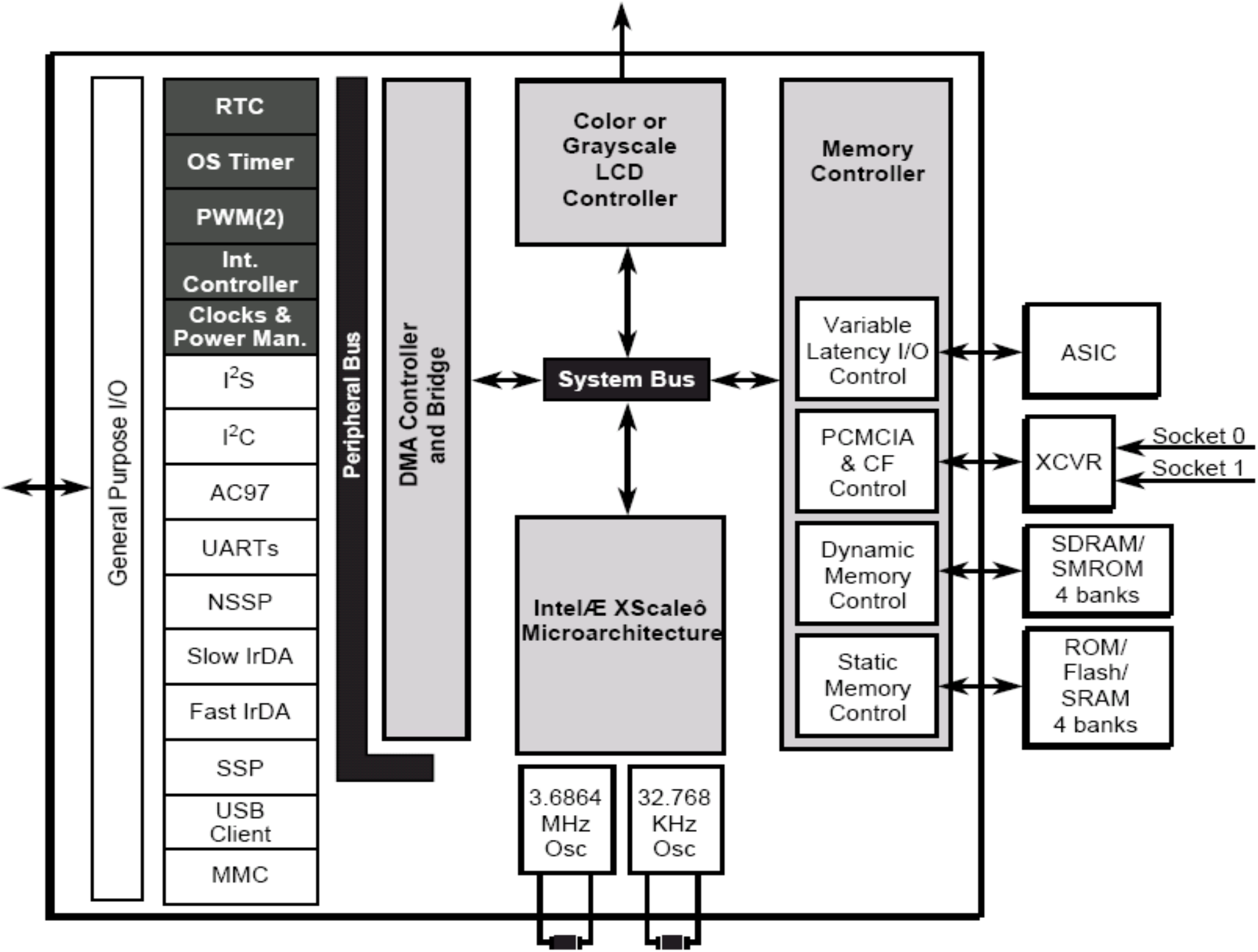
Michael Jordan said...



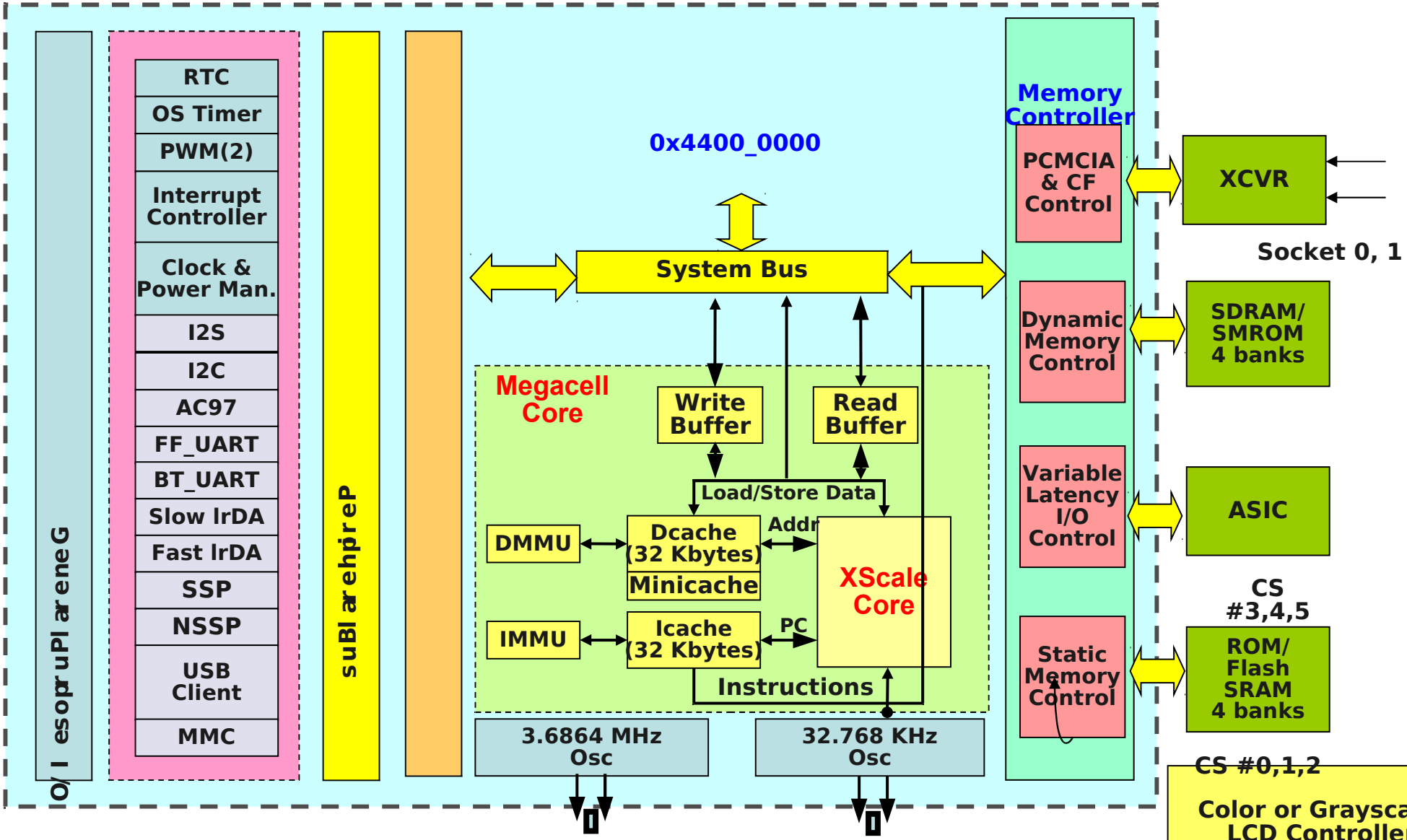
I can accept failure.
Everyone fails at something.
But I cannot accept not *trying.*

在 Part II 議程中，將會以 CuRT 爲例，探討前述觀念的實務

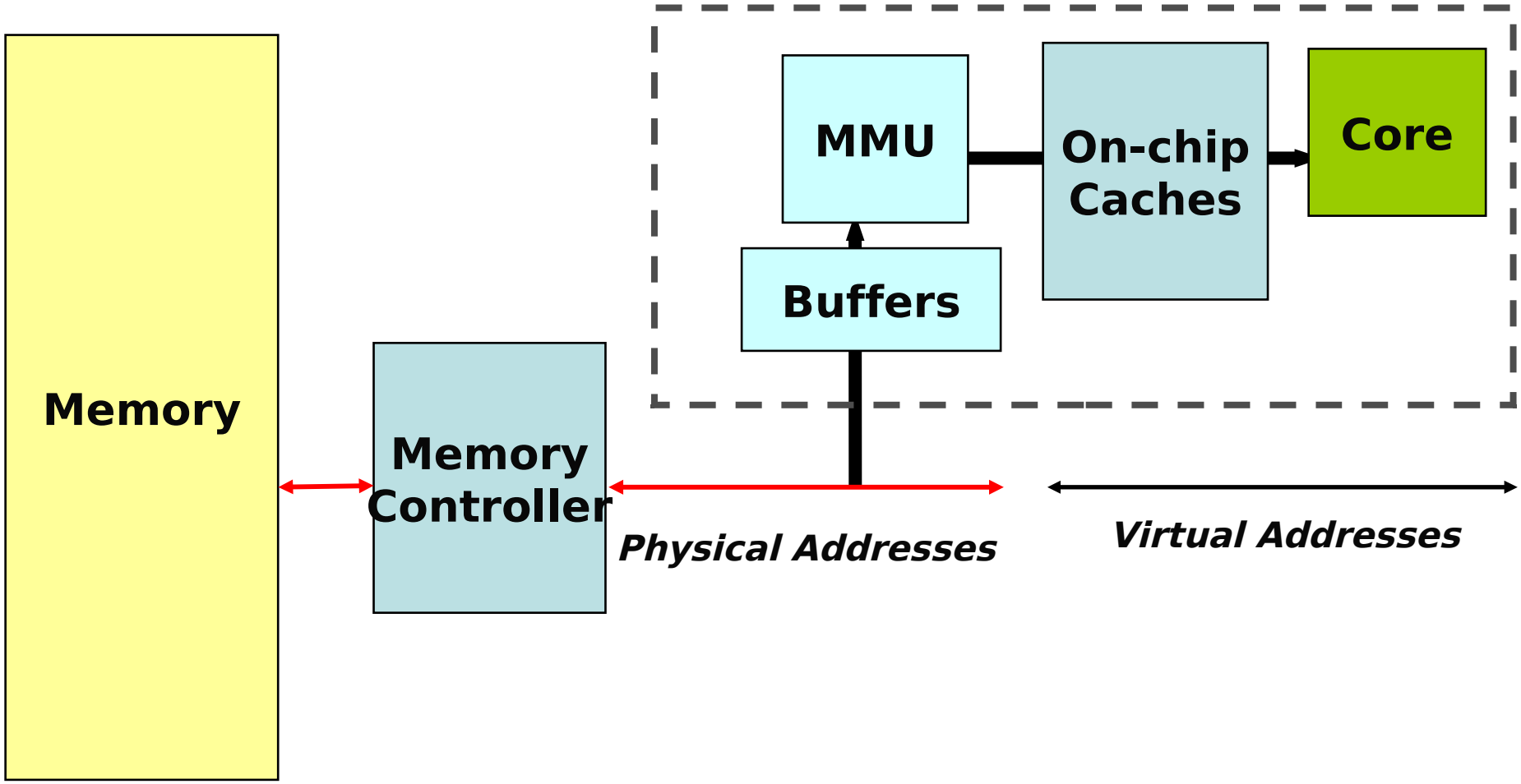
繼續探索 PXA255 SoC


General Purpose I/O
RTC
OS Timer
PWM(2)
Int. Controller
Clocks & Power Man.
I2S
I2C
AC97
UARTs
NSSP
Slow IrDA
Fast IrDA
SSP
USB Client
MMC
Peripheral Bus
DMA Controller and Bridge
Color or Grayscale LCD Controller
System Bus
IntelÆ XScaleô Microarchitecture
3.6864 MHz Osc
32.768 KHz Osc
Memory Controller
Variable Latency I/O Control
PCMCIA & CF Control
Dynamic Memory Control
Static Memory Control
ASIC
XCVR
Socket 0
Socket 1
SDRAM/ SMROM 4 banks
ROM/ Flash/ SRAM 4 banks

PXA255 Function Block

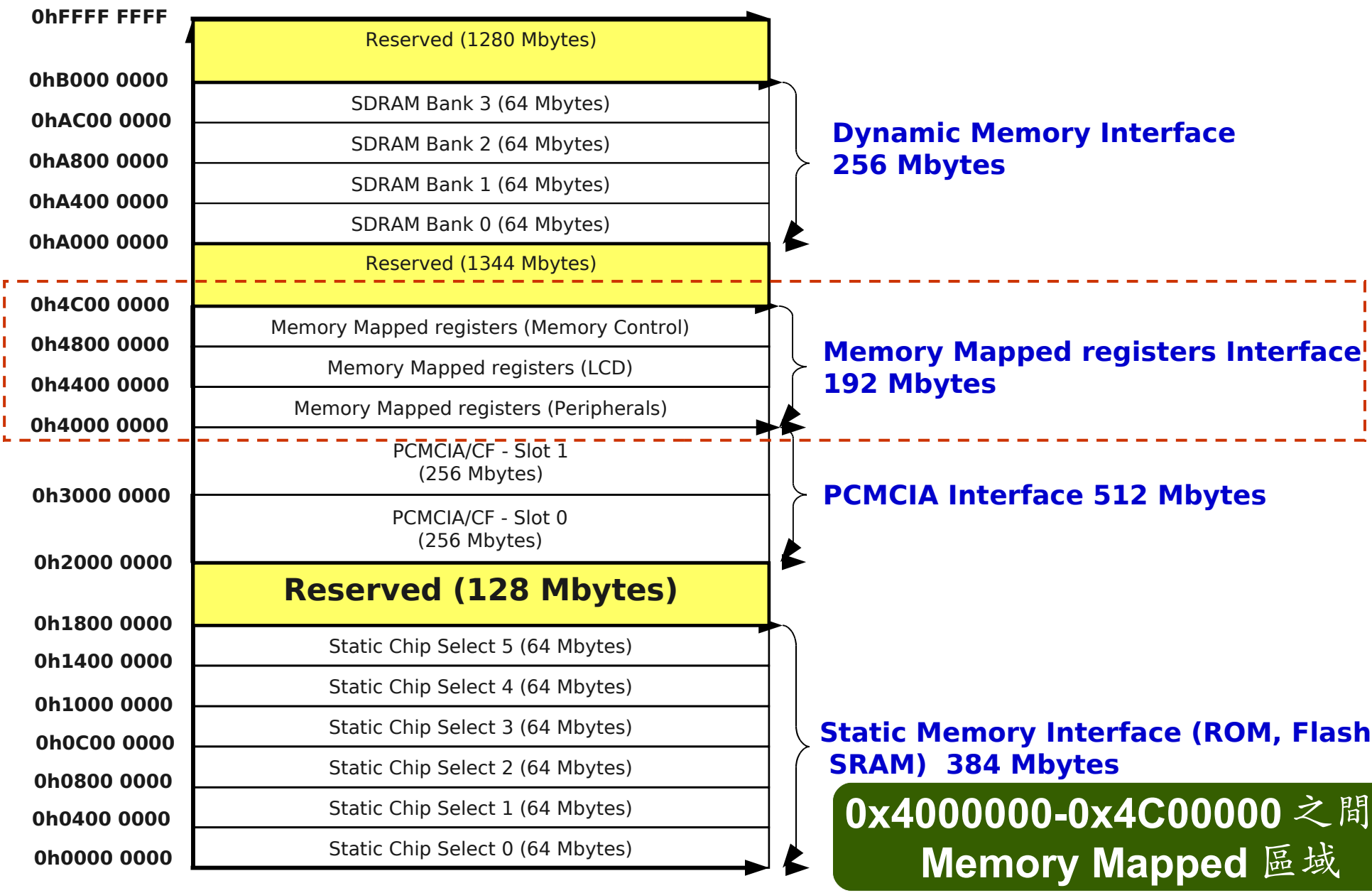


PXA255 Memory Model



http://www.intel.com/design/pca/prodbref/252780docs.htm

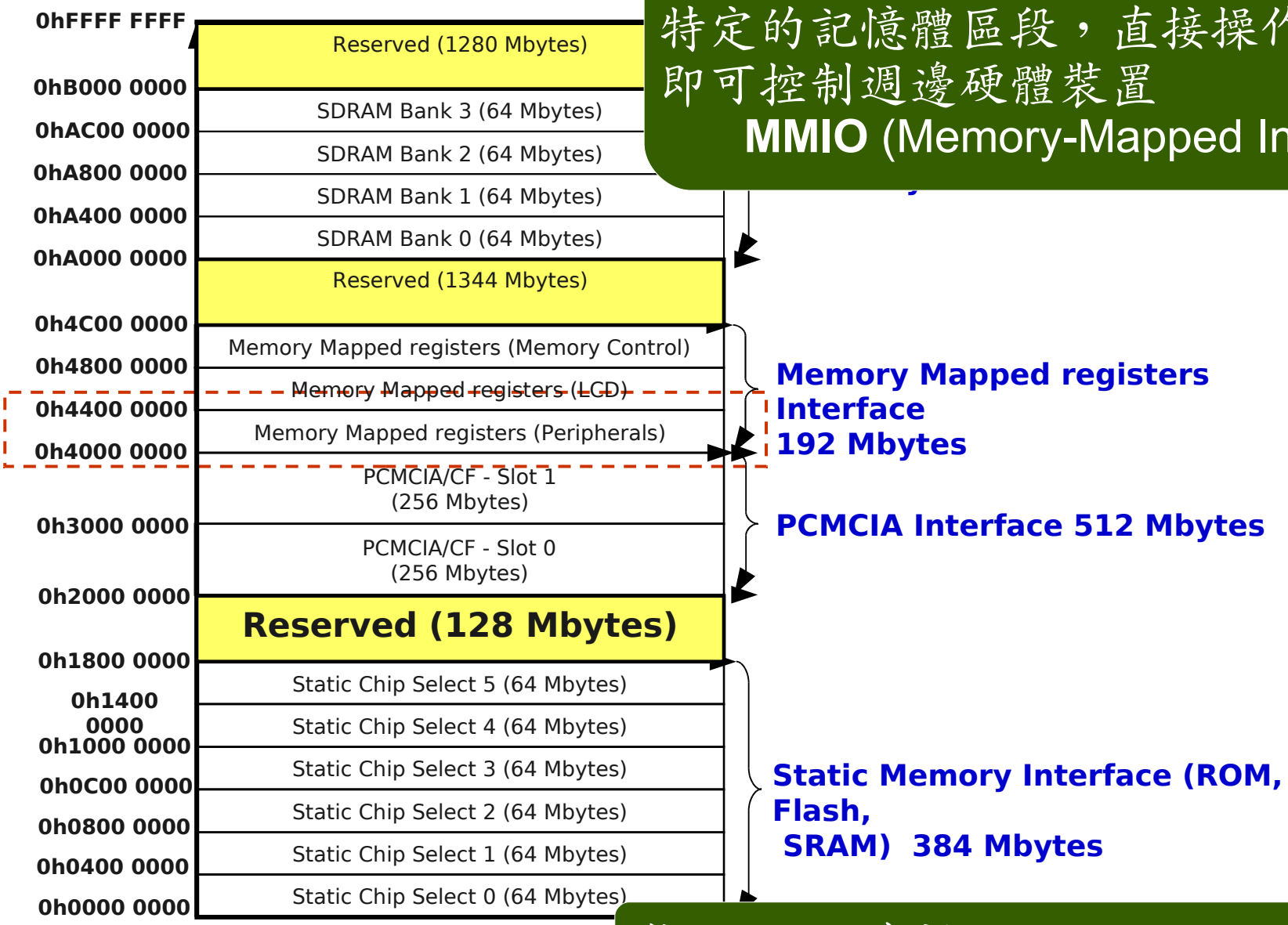
Memory Map


0hFFFF FFFF
Reserved (1280 Mbytes)
0hB000 0000
SDRAM Bank 3 (64 Mbytes)
0hAC00 0000
SDRAM Bank 2 (64 Mbytes)
0hA800 0000
SDRAM Bank 1 (64 Mbytes)
0hA400 0000
SDRAM Bank 0 (64 Mbytes)
0hA000 0000
Reserved (1344 Mbytes)
0h4C00 0000
Memory Mapped registers (Memory Control)
0h4800 0000
Memory Mapped registers (LCD)
0h4400 0000
Memory Mapped registers (Peripherals)
0h4000 0000
PCMCIA/CF - Slot 1
(256 Mbytes)
0h3000 0000
PCMCIA/CF - Slot 0
(256 Mbytes)
0h2000 0000
Reserved (128 Mbytes)
0h1800 0000
Static Chip Select 5 (64 Mbytes)
0h1400 0000
Static Chip Select 4 (64 Mbytes)
0h1000 0000
Static Chip Select 3 (64 Mbytes)
0h0C00 0000
Static Chip Select 2 (64 Mbytes)
0h0800 0000
Static Chip Select 1 (64 Mbytes)
0h0400 0000
Static Chip Select 0 (64 Mbytes)
0h0000 0000
Dynamic Memory Interface
256 Mbytes
Memory Mapped registers Interface
192 Mbytes
PCMCIA Interface 512 Mbytes
Static Memory Interface (ROM, Flash,
SRAM) 384 Mbytes
0x4000000-0x4C00000 之間的
Memory Mapped 區域

Memory Map

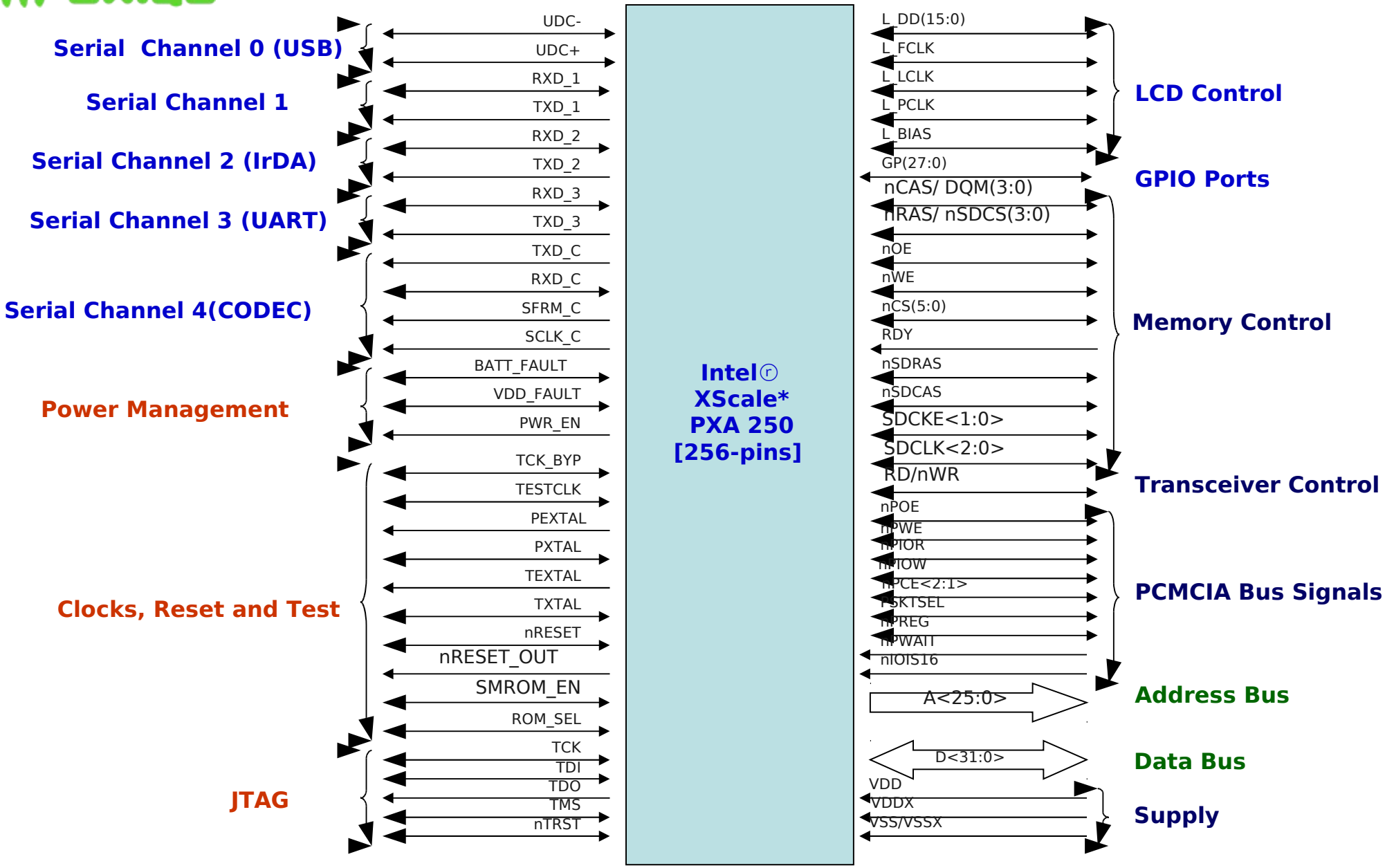
在 SoC 上，週邊 (peripheral) 被對應到特定的記憶體區段，直接操作該記憶體，即可控制週邊硬體裝置

MMIO (Memory-Mapped Input/Output)

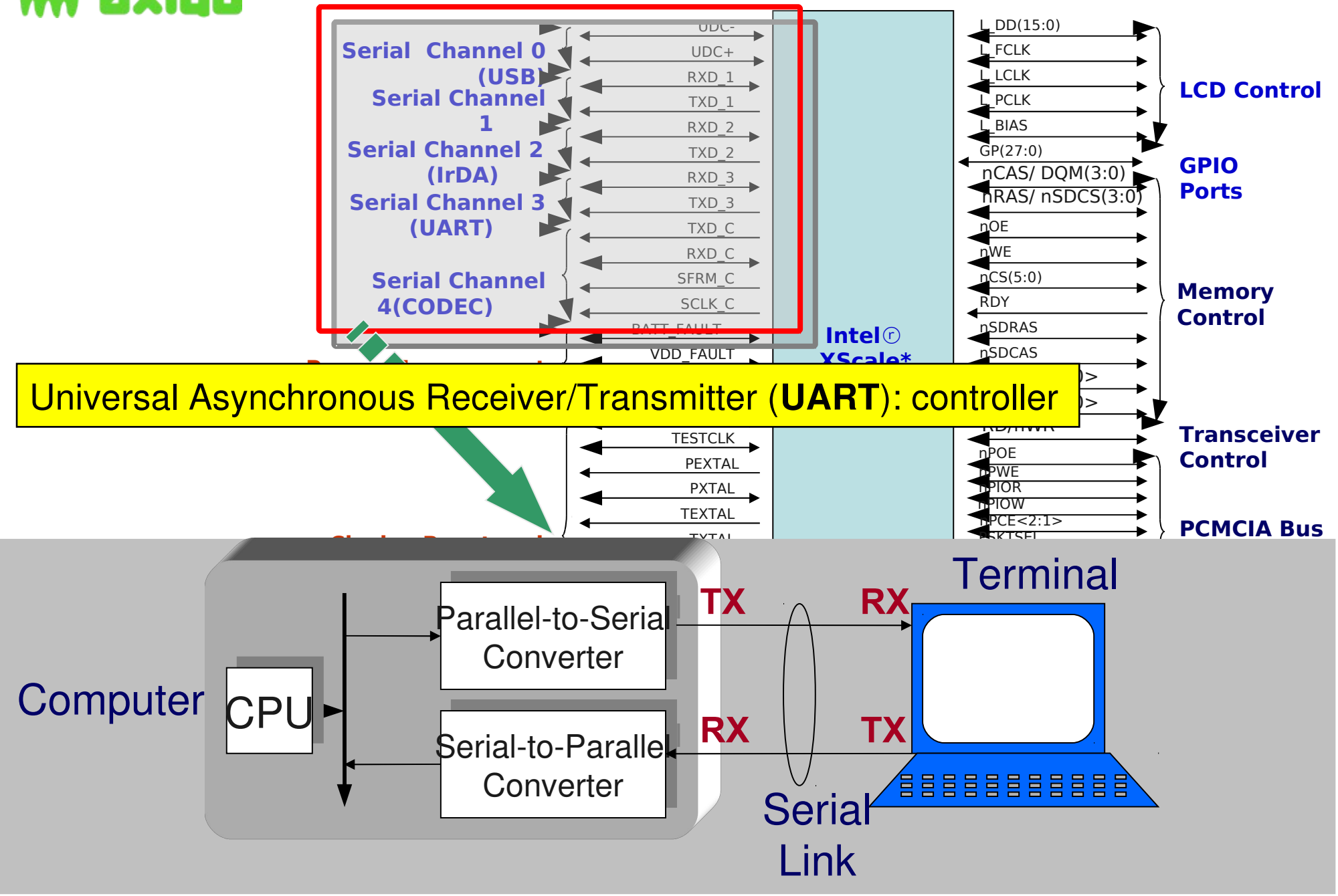


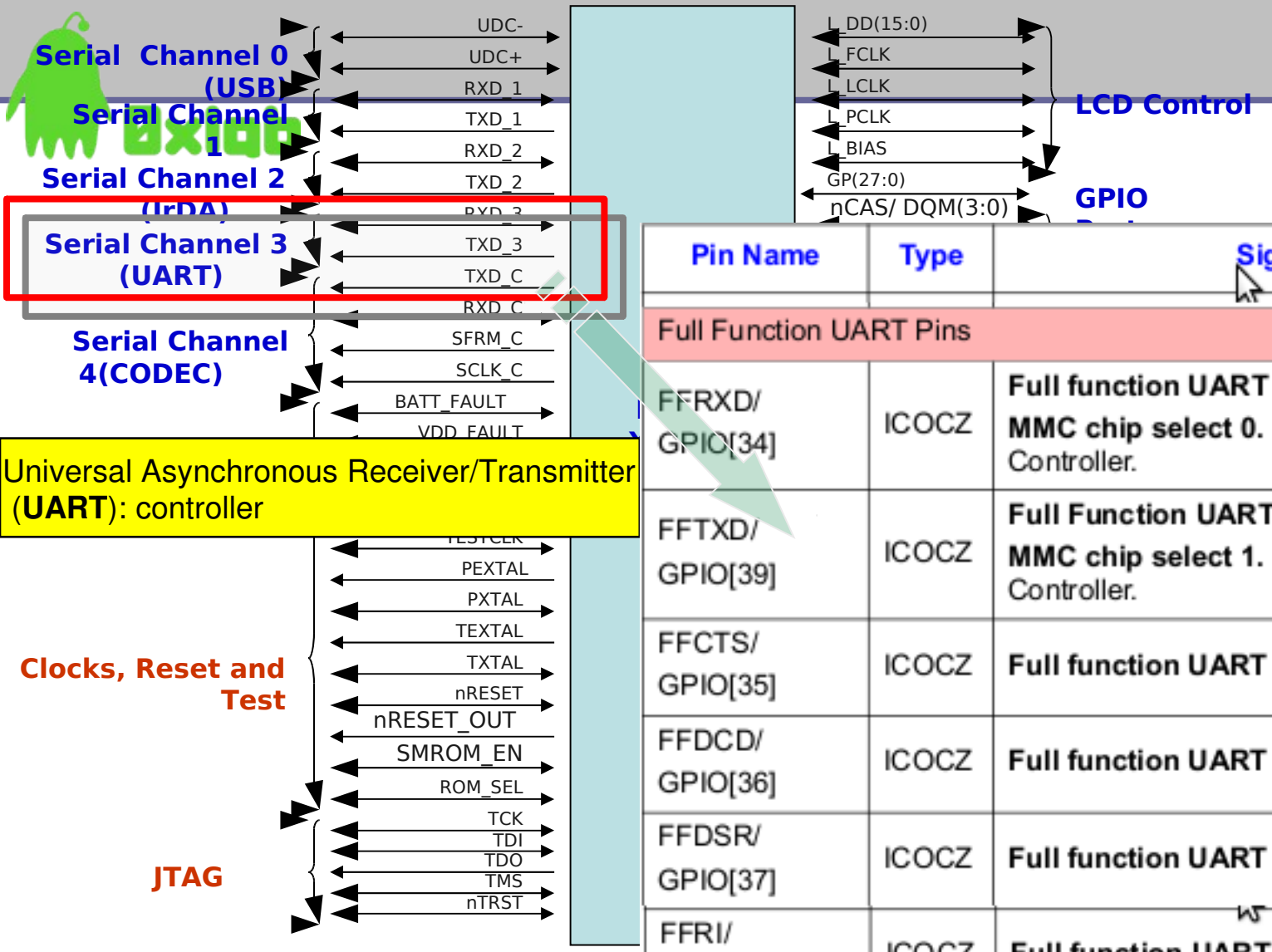
對 PXA255 來說，即 0x0x4000000-0x4400000

PXA255 core Function Diagram



PXA255 Serial/UART





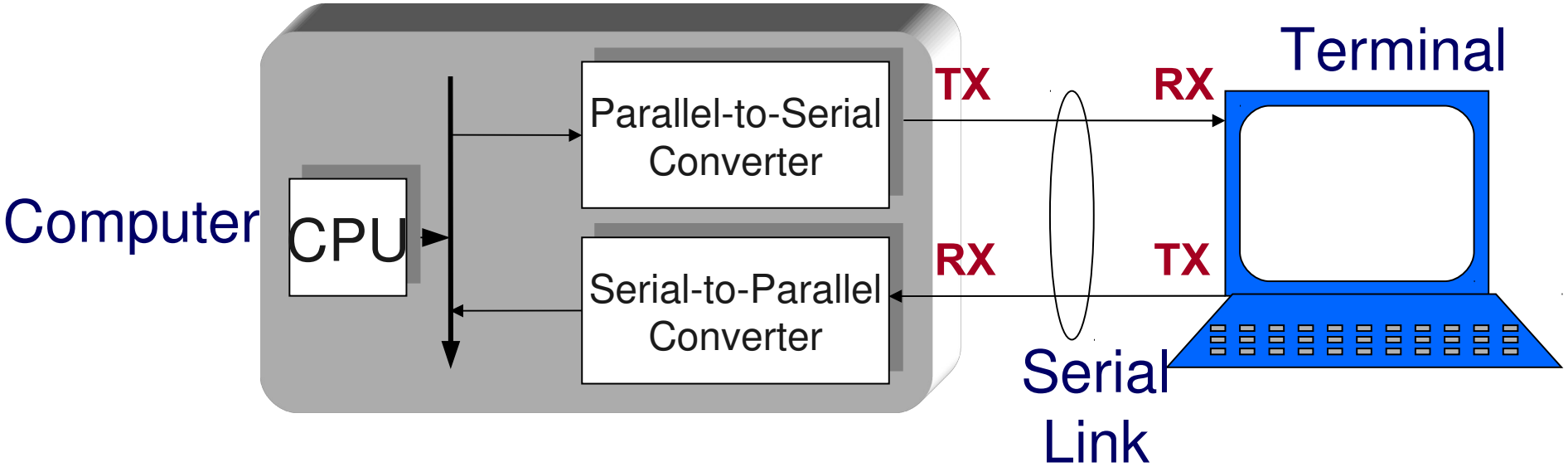
Universal Asynchronous Receiver/Transmitter (**UART**): controller

Pin Name	Type	Signal Descriptions
Full Function UART Pins		
FFRXD/ GPIO[34]	ICOCZ	**Full function UART receive.** (input) **MMC chip select 0.** (output) Chip select 0 for the MMC Controller.
FFTXD/ GPIO[39]	ICOCZ	**Full Function UART transmit.** (output) **MMC chip select 1.** (output) Chip select 1 for the MMC Controller.
FFCTS/ GPIO[35]	ICOCZ	**Full function UART clear-to-send.** (input)
FFDCD/ GPIO[36]	ICOCZ	**Full function UART data-carrier-detect.** (input)
FFDSR/ GPIO[37]	ICOCZ	**Full function UART data-set-ready.** (input)
FFRI/ GPIO[38]	ICOCZ	**Full function UART ring indicator.** (input)
FFDTR/ GPIO[40]	ICOCZ	**Full function UART data-terminal-ready.** (output)
FFRTS/ GPIO[41]	ICOCZ	**Full function UART request-to-send.** (output)

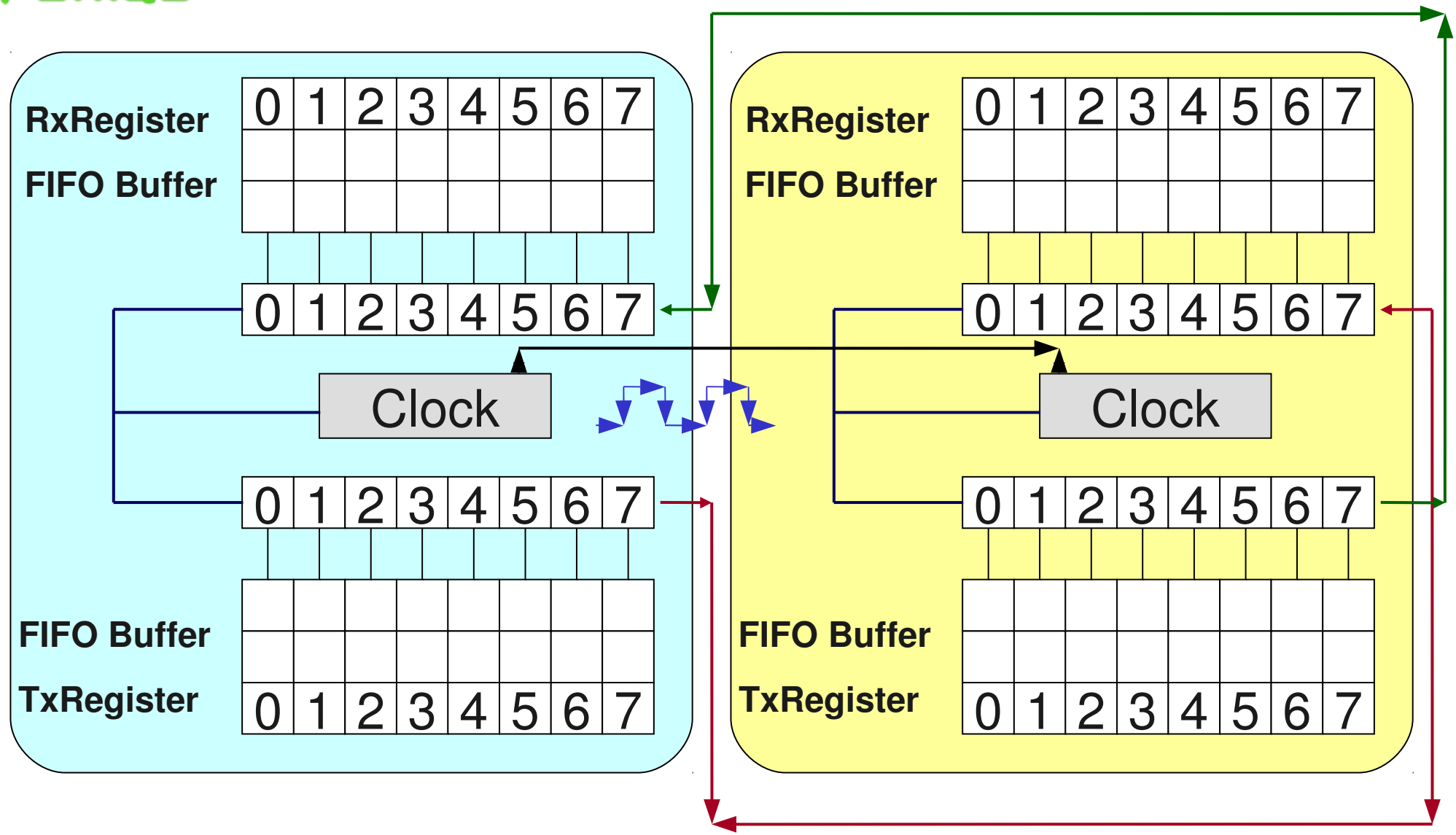
FFUART (Full function UART) 的 (memory mapped) base address 爲 **0x40100000**

Data Communication

- 透過 serial line 建立電腦與終端機模擬程式的通訊
  - Data is converted from parallel (bytes) to serial (bits) in the bus interface
  - Bits are sent over wire (TX) to terminal (or back from terminal to computer)
  - Receiving end (RX) translates bit stream back into parallel data (bytes)



Serial Port



Processor **Peripheral**

Sample shell for CuRT

Applications Places System

root@venux: ~

```
jserv@venux:~/realtime/CuRT_v1/app/shell$ ./run-on-connex
pxa2xx_clkpwr_write: CPU frequency chang
#######################################
#       Start CuRT....                #
#######################################
 ID        State        Name
           Ready        idle-thread
  1        Ready        shell_thread
  2        Running      info_thread
  3        Ready        statistics_thread
  4        Ready        hello_thread
  5        Ready        hello2_thread
********** CuRT statistics info ******
Total Thread Count : 6
Total Context Switch Count : 1
Current Time Tick : 0
**********************************
$
```

CuRT_v1/app/shell/main.c

```
int main()
{
    SerialInit();
    init_interrupt_control();
    init_curt();

    ...
    shell_tid = thread_create(&shell_thread,
        &thread_stk[0][THREAD_STACK_SIZE-1],
        shell_thread_func,
        "shell_thread",
        5, NULL);
```

CuRT_v1/app/shell/main.c

```
static void shell_thread_func(void *pdata)
{
    ...
    char buf[80] = { '\0' };
    while (1) {
        printf("$ ");
        gets(buf);
        printf("\n");
        ...
        else if (!strcmp(buf, "stat")) {
            thread_resume(stat_tid);
```

Serial “Driver” in CuRT

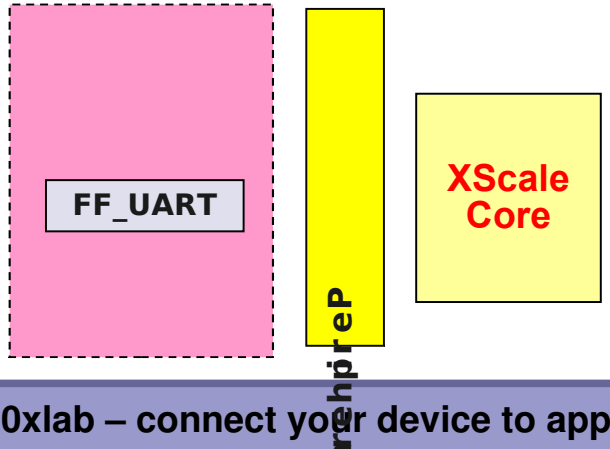
CuRT_v1/device/serial.c

```
void SerialInit(void)
{
        /* GP39, GP40, GP41 UART(10) */
        GAFR1_L |= 0x000A8000;
        GPDR1 |= 0x00000380;

        /* 8-bit, 1 stop, no parity */
        rFFLCR = 0x00000003;

        /* Reset tx, rx FIFO. clear. FIFO enable */
        rFFFCR = 0x00000007;

        /* UART Enable Interrupt */
        rFFIER = 0x00000040;
```



對 PXA255 之 FF_UART
(Full Function UART) 作初始化

Serial “Driver” in CuRT

CuRT_v1/device/serial.c

```
void SerialOutputByte(const char c) {
        /* FIFO : wait for ready */
        while ((rFFLSR & 0x00000020) == 0 )
                /* wait */;
        rFFTHR = ((ulong) c & 0xFF);
        if (c == '\n')
                SerialOutputByte('\r');
}
int SerialInputByte(char *c) {
        /* FIFO */
        if ((rFFLSR & 0x00000001) == 0) {
                return 0;
        }
        else {
                *(volatile char *) c = (char) rFFRBR;
                return 1;
        }
}
```

CuRT_v1/includes/arch/arm/mach-pxa/pxa255.h

```
/** Full Function UART */
#define FFUART_BASE     0x40100000
#define rFFTHR          (*((volatile ulong *)(FFUART_BASE+0x00)))
#define rFFLSR          (*((volatile ulong *)(FFUART_BASE+0x14)))
```

Applications Places System 5:38 上午

```
jserv@venux:~/realtime/CuRT_v1/ap
pxa2xx_clkpwr_write: CPU frequenc
#######################################
#       Start CuRT....
#######################################
 ID        State        Name
           Ready        idle-thre
  1        Ready        shell_thr
  2        Running      info_thre
  3        Ready        statistic
  4        Rea
  5        Rea
********** Cu
Total Thread C
Total Context
Current Time T
**********
$
```

QEMU [Stopped]

```
QEMU 0.10.5 monitor - type 'help' for more information
(qemu) print 0x40100000 + 0x00
0x40100000
(qemu) x/1d 0x40100000
40100000:          0
(qemu) x/1d 0x40100000
40100000:         65
(qemu)
```

```
(qemu) print 0x40100000 + 0x00
0x40100000
(qemu) x/1d 0x40100000
0x40100000:        0
(qemu) x/1d 0x40100000
0x40100000:        65
```

```
CuRT_v1/includes/arch/arm/mach-pxa/pxa255.h
/** Full Function UART */
#define FFUART_BASE      0x40100000
#define rFFTHR           (*((volatile ulong *)(FFUART_BASE+0x00)))
#define rFFLSR           (*((volatile ulong *)(FFUART_BASE+0x14)))
```

```
78          /* Make sure the data is received. */
79          if (rFFLSR & 0x00000001)
80                  return 1;
81 ->       return 0;
82  }
83
/home/jserv/realtime/CuRT_v
Continuing.
(gdb)
Program received signal SIGINT, Interrupt.
SerialIsReadyChar () at ../../device/serial.c:81
(gdb)
```

在 CuRT 的 serial 輸入 'A' 字元

'A' = 0x41 = 65

CuRT serial driver is disabled

Applications Places System 5:56 上午

```
jserv@venux:~/realtime/CuRT_v1/ap
pxa2xx_clkpwr_write: CPU frequenc
#######################################
#       Start CuRT....
#######################################
 ID        State        Name
           Ready        idle-thre
  1        Ready        shell_thr
  2        Running      info_thre
  3        Ready        statistic
  4        Ready        hello_thr
  5        Ready        hello2_th
********** CuRT statistics info
Total Thread Count : 6
Total Context Switch Count : 1
Current Time Tick : 0
*****************************
$ a
```

QEMU [Stopped]

```
QEMU 0.10.5 monitor - type 'help' for more information
(qemu) print 0x40100000 + 0x00
0x40100000
(qemu) x/1d 0x40100000
40100000:          0
(qemu) x/1d 0x40100000
40100000:         65
(qemu)
```

繼續在 CuRT 的 serial 打字，模擬在 FIFO Buffer ，等累積若干字元後，再到 gdb 視窗中敲入 'c' 指令以恢復 CuRT 的執行

rxvt-unicode

```
49          while ((rFFLSR & 0x00000020) == 0 )
50                  /* wait */ ;
51
52          rFFTHR = ((ulong) c & 0xFF);
53
54          /* regardless of c=='\n' or "\n\r", the same outpu
/home/jserv/realtime/CuRT_v1/device/serial.c
Program received signal SIGINT, Interrupt.
SerialIsReadyChar () at ../../device/serial.c:81
(gdb) p SerialOutputByte(97)
$1 = void
(gdb)
```

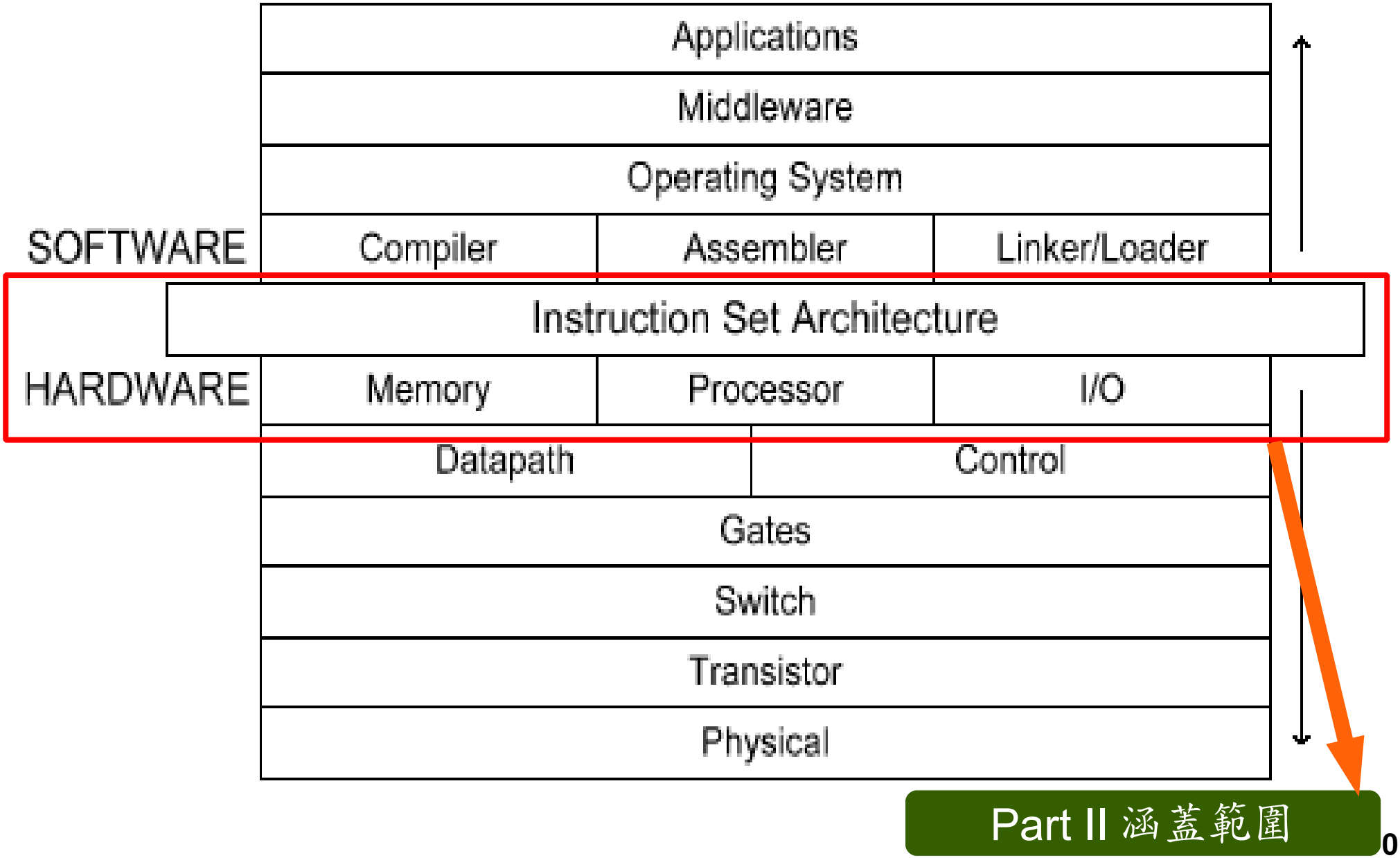
'a' = 0x61 = 97

Part I 回顧

- ARM 架構
  - Architecture version vs. Implementation
  - ISA feature
- ARM SoC 平台
  - 整合多種不同功能的複雜 IC 組合，針對特定的市場或應用需求
  - 典型組成
- 關鍵概念
  - 工作模式、暫存器組、系統狀態
  - 指令集、例外處理

Part II 提綱：系統控制

- PXA255 SoC 與 CuRT 的硬體啓動程序
- ARM 定址與組合語言概況
- ARM Interrupt, ISR, Exception 的處理


Applications
Middleware
Operating System
SOFTWARE
Compiler
Assembler
Linker/Loader
Instruction Set Architecture
HARDWARE
Memory
Processor
I/O
Datapath
Control
Gates
Switch
Transistor
Physical
Part II 涵蓋範圍

參考資訊

- ARM Limited *ARM Architecture Reference Manual*, Addison Wesley, June 2000
- ARM Architecture Manual
- Trevor Martin *The Insiders Guide To The Philips ARM7-Based Microcontrollers,* Hitex (UK) Ltd., February 2005
- Steve Furber *ARM System-On-Chip Architecture (2nd edition),* Addison Wesley, March 2000
- Intel Xscale Programmers Reference Manual
- Cortex-A8 Technical Reference Manual
- *The Definitive Guide to the ARM Cortex-M3*, Joseph Yiu