DUAL-CORE

LPC MCU KEY FEATURES

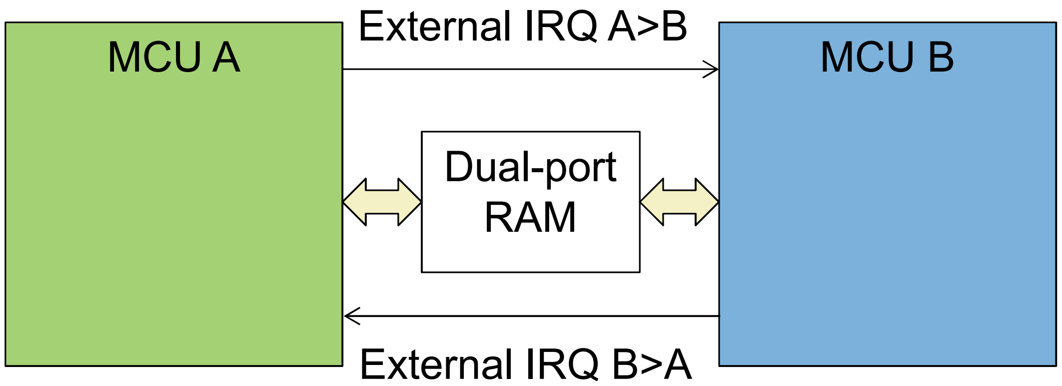
EXTERNAL USE

SECURE CONNECTIONS
FOR A SMARTER WORLD

BASIC CONCEPTS

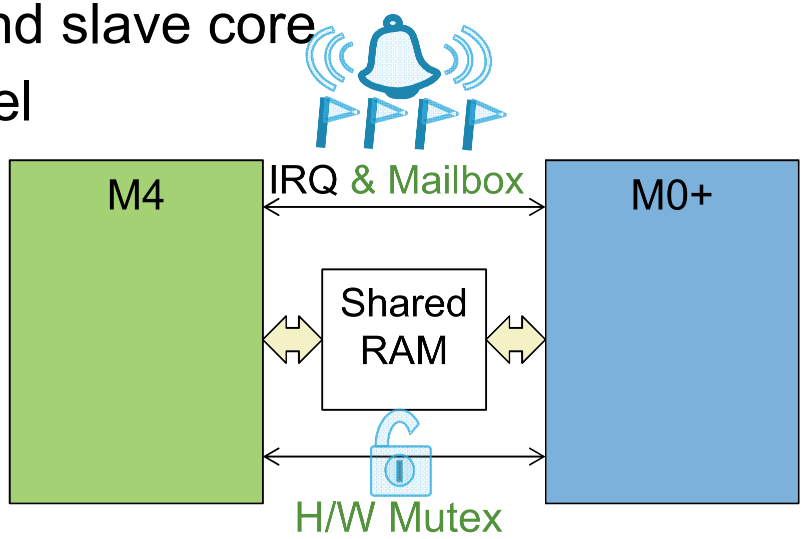
Prototype: Dual-MCU systems

- Needed when single MCU is insufficient
- Asymmetric: Main MCU and co processing MCU (co-MCU)
  - Main MCU handles main functions and algorithm operation
  - Co MCU handles I/O, data transfer, peripherals, misc
- Communication: bus or dual-port RAM
- Sync: External pin IRQ
- Cons
  - PCB area and BOM cost
  - May need two suites of tool chain
  - Poorer steability
- Distributed system with multiple MCUs
  - Communication via RS-485, CAN, etc.
  - Nodes in principle are independent, don't covered here.



From dual-MCU to asymmetric dual-core

- Single MCU with two CPU cores: master core and slave core
- Shares similar design objectives and using model
- Shares similar developing model
  - Two projects
  - Develop and Debug independently
  - Debug concurrently
- Master core has extra works
  - Bring up system
  - Setup dual-core control hardware and start slave core
  - (Optional) Integrate the image binary data of slave core.
- Two cores share power supply, clock and reset, may affects each other.
- Allocate on-chip resource and dispatch job carefully.



M4 vs. M0/M0+

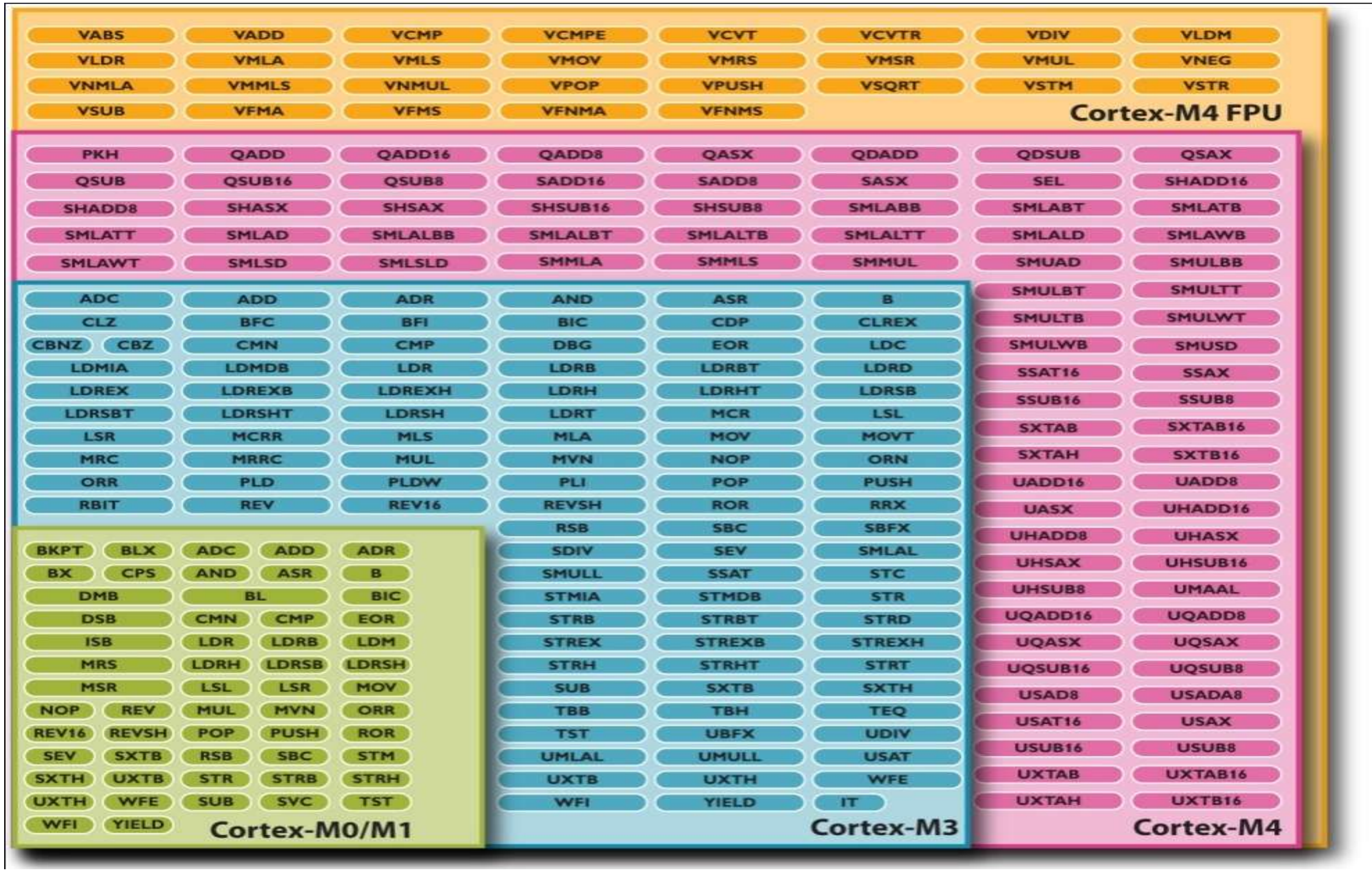
Thumb 2
DSP +FPU
1.25 DMIPS/MHz

Cortex M4F

+

Thumb only
No hardware divide
0.9 DMIPS/MHz

Cortex M0/M0+



Cortex
Intelligent Processors by ARM

Case of asymmetric dual-core：LPC541xx & LPC43xx

Cortex-M4F
100MHz (541xx)
204 MHz (43xx)

+

Cortex-M0/M0+
100MHz (541xx)
204MHz (43xx)

Processing, computation

Math and algorithm

+

Real time control

Peripherals

Data transfer

= Solution

The slave core takes the responsibilities of misc tasks, relief the master core to concentrate on computation intensive tasks.

Architectural requirement to support dual-core

- Multiple blocks of SRAM
  - Code and data for slave core
  - Shared data
  - Parallel data flow
- Multi-layer AHB matrix
  - Slave core get access of resources through AHB matrix.
- (Enhanced in LPC541xx) Inter-core communication, synchronization, and mutex.
- Debug facilities
  - LPC43xx: 2 JTAG scan chains: one for M4, the other for M0+, must use JTAG to debug M0
  - LPC541xx: Multi-drop SWD DAP, use SWD to debug both core (Same debug port, select different Access Port for different core).

DUAL-CORE USING MODELS AND ADVANTAGE

Warm hints: Slave core is not must to use

- Extra die cost to implement secondary core is almost OK to ignore.
- M4 usually leads to faster speed and/or lower energy (see later).
- Slave core requires extra overhead:
  - RAM space for data and/or code
  - Very large bus bandwidth for instruction fetch: 48MHz M0+ can requires 96MB/s
  - Software complexity for inter-core communication, synchronization, mutual exclusion.
- Do NOT use slave core just for using it.

Overview of slave core using models

- In most cases, M0/M0+ is suitable to act as slave core
  - M4 usually leads to faster speed and/or lower energy (see later).
- Avoid math intensive task to M0/M0+, including integer division, DSP, FP.
  - M4 may be several to dozens time faster than M0/M0+ w/ dedicated instructions.
- What is Suitable for M0/M0+:
  - Handling high frequency IRQ or tough real-time constraints in complex systems.
  - Irregular data manipulation and movement (beyond normal DMA)
  - non-standard data/communication interface, bus, or protocols.
  - Simulate/enhance standard interfaces (UART/SPI/I2S/I2C)
  - In some cases, get lower energy consumption.
  - Above conditions often meet together: e.g., High data rate leads to high frequency IRQ

Slave core handles high frequency IRQ

- IRQ overhead
  - ISR entrance: 12 (M4) or 24 (M0/M0+) clocks
  - ISR execution: Dozens of clocks or more
  - ISR exit: 12 (M4) or 24 (M0/M0+) clocks
    - Note: Cortex-M has “tail-chaining” mechanism to shorten back-to-back IRQ latency.
- When to use slave core:
  - Insufficient master core horse power budget
  - master core has long critical sections to miss deadline of some real time constraints.
- Slave core: Handles IRQ and processing related data buffers
- Examples:
  - Software QEI decoder.

Slave core acts as "smart" DMA

- Some data transfer can't be handled by DMA:
  - Irregular (non-linear) address generation and increment: 2D graphics window operation
  - Irregular data width: non 8-bit/16-bit/32-bit,
  - Very short and/or precise delays (such as dozens of clocks) are required to meet timing constraints.
  - Special time-out constraints:
    - E.g., UART RX on LPC5411x with high baudrate, can use slave core to add time-out .
- Slave core to handle
  - some bit-field manipulation (E.g., Split/join data bits)
  - Manipulate communication ports, GPIO or SGPIO (LPC43xx) to transmit or receive data

Slave core to implement data/communication interfaces

- Non standard communication ports
  - E.g.: Camera interface,
  - SCT and SGPIO (LPC43xx only) may help in many cases
- Simulate more or enhanced standard communication ports
- (LPC4300) M0 with SGPIO can simulate many interfaces/buses
  - SGPIO can do many sorts of serial<->parallel conversion with its shift register array.
  - SGPIO IRQ need to be handled by M0.

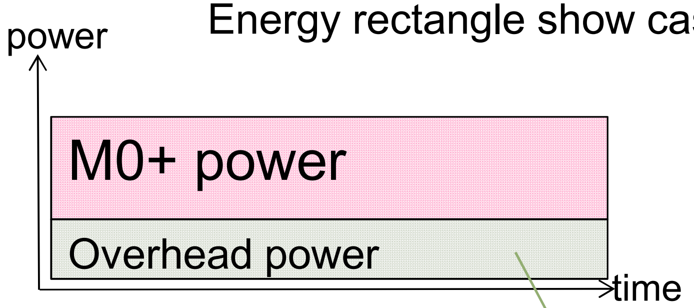
Slave core handles misc non-computational tasks

- In some cases, satisfy some requirements otherwise will need CPLD/FPGA
  - Most of which are GPIO related operations
- High speed, accurate executor
  - GPIO operation
- Run some high real-time protocols
  - Mainly for various field bus specifications.

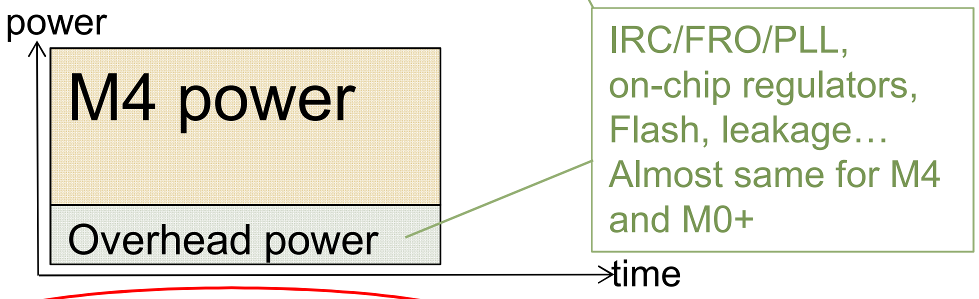
Use M0+ to get lower energy consumption

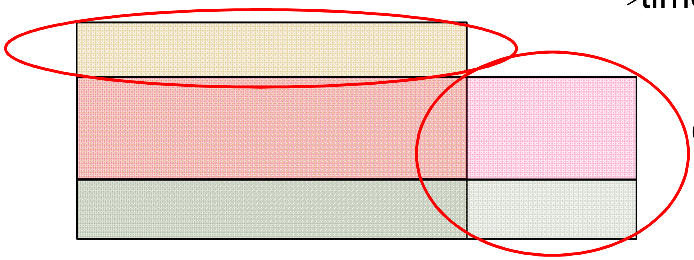
- Conditions that makes M0+ to save energy:
  - Clock frequency is high (typically 48MHz+), or CPU can't sleep due to some strict performance constraints, such as high IRQ rate, accurate timing, etc.
    - On LPC5410x, M0+ is as low as 55% power of M4 when CPU MHz is > 48MHz
  - M4 and M0+ can run in parallel under active mode.
  - M0+ code does not involve math and M4 only stuffs:
    - integer DIV (and MUL on LPC5410x, 32 times slower than M4)
    - DSP, SIMD, and floating point.
    - Other M4 advantages: bit-field manipulation, bitmap based allocator helpers (CLZ, RBIT), high bandwidth data transfer, high frequency IRQ handling (not for this sake).
- M0+ uA/MHz is lower than M4, but usually cost longer time to complete same task, power * time (energy) is not must be lower.
  - M0 has weaker instruction set and single bus master.
- Be aware, CPU is not the only power sink
  - IRC/FRO/PLL, regulator, flash, leakage, 700uA+

Energy rectangle show case



Energy = power * time ---- **AREA** of rectangle





Energy rectangle comparison

RESOURCE ALLOCATION

Resource allocation

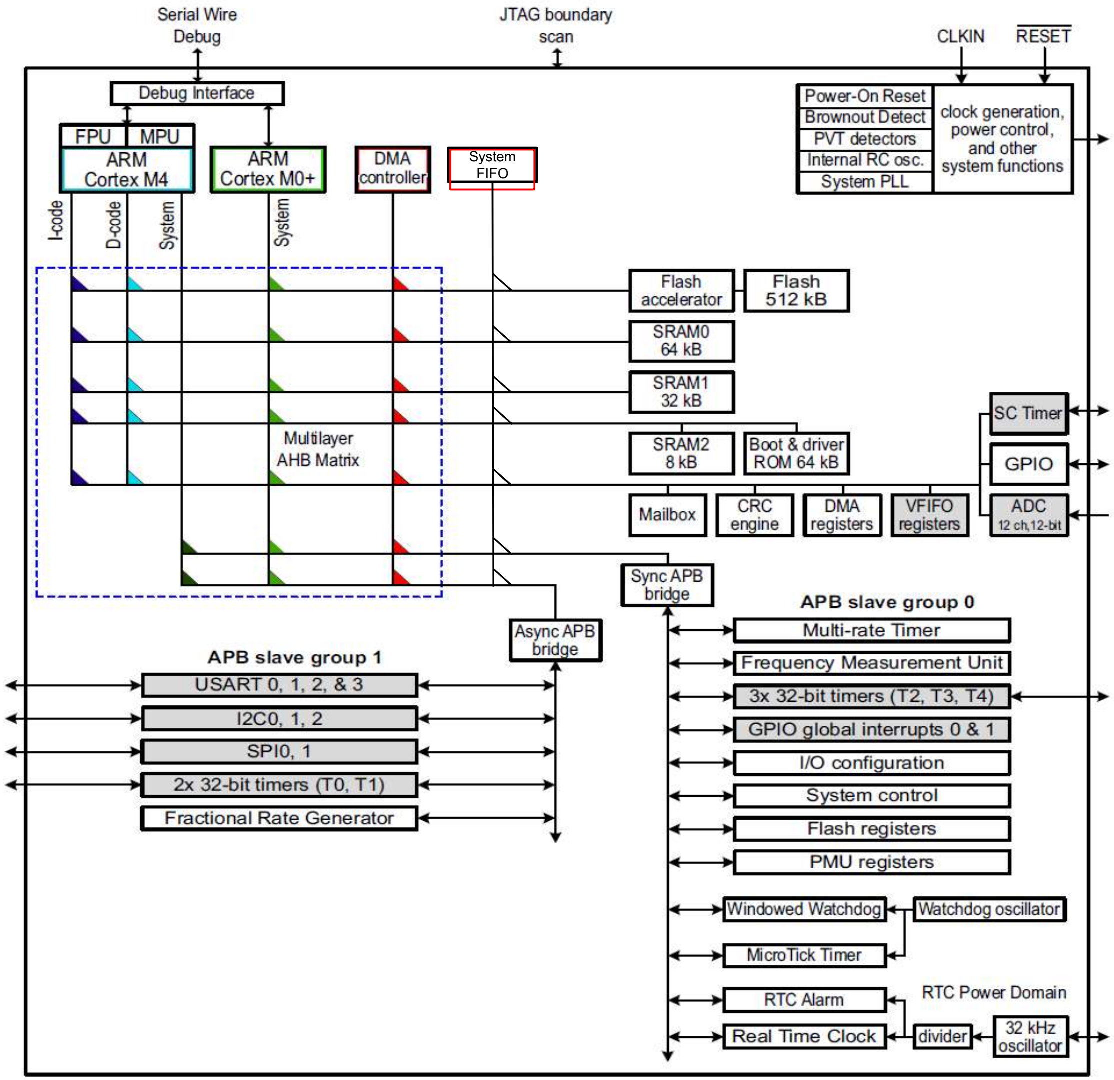
- **Resource types**
  - Dedicated memory for code & data
  - Shared RAM
  - On-chip peripheral partition

16

Memory map considerations

- M4 (Harvard architecture) has 3 master interfaces:
  - I-Code：Fetch instruction from bottom 512MB space
  - D-Code：Access data from bottom 512MB space
  - System：Access most remaining area for both instruction and data
- M0 has only 1 master interface
- On-chip RAM are partitioned to multiple blocks at different AHB slave ports.
- Take advantage of I and D buses on M4 core:
- Whenever feasible, use a dedicated SRAM block for shared data.

MemoryAHB Matrix in LPC54102

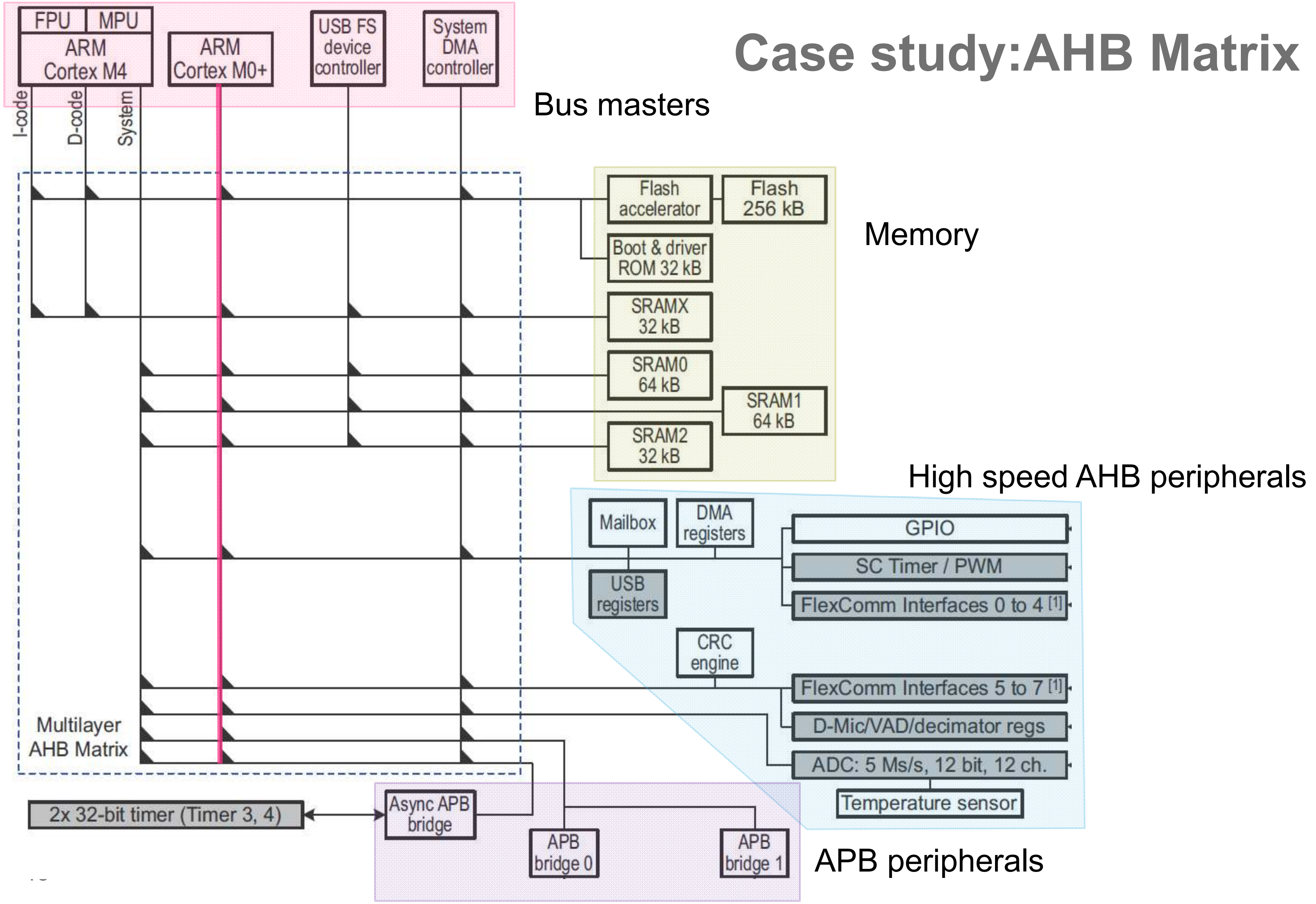


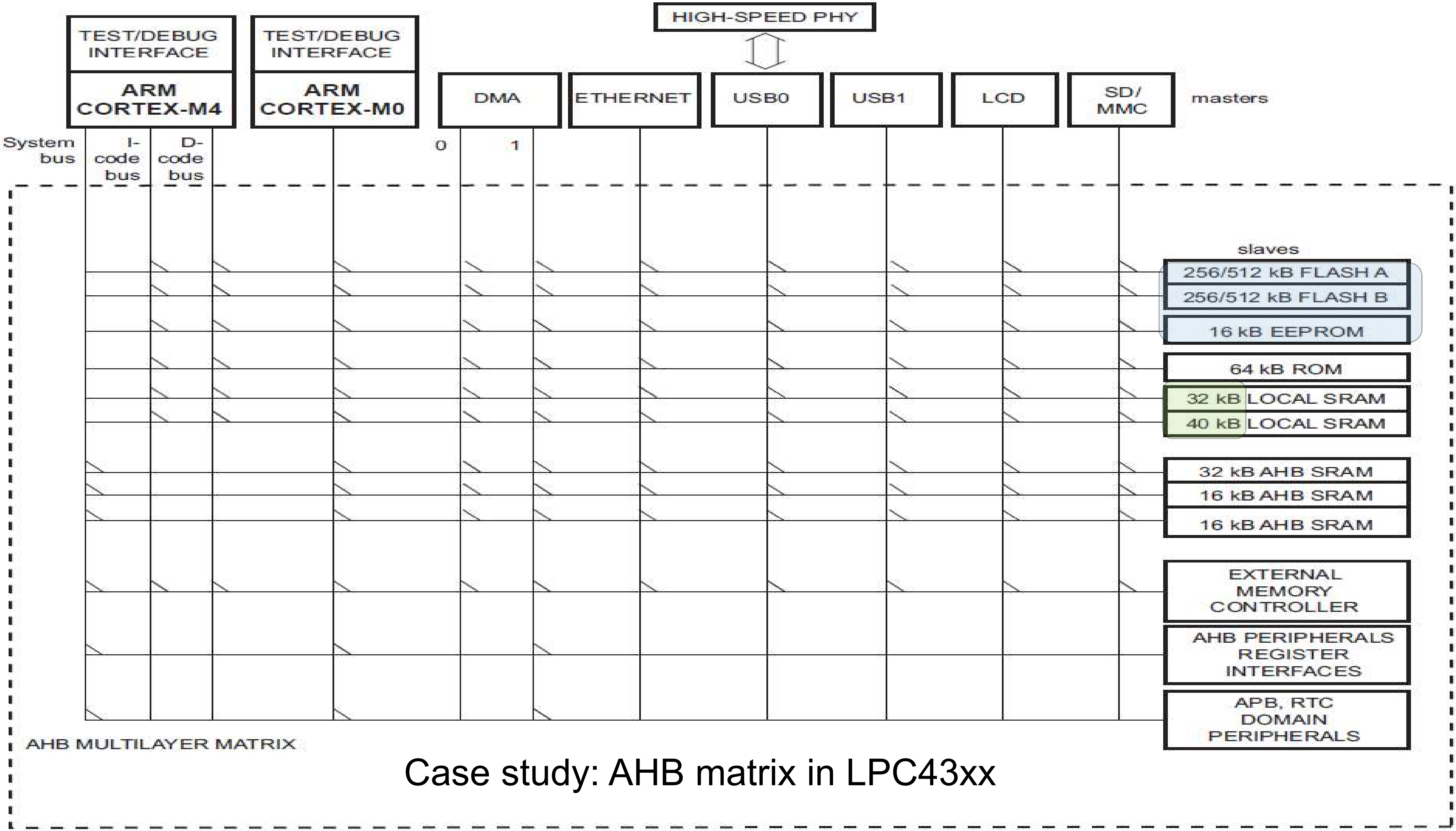
- M4 and M0+ are both AHB masters
- M4: I-Code and D-Code bus is optimized to special address range.
- Both cores have full access to peripherals and memories
- Bus prioritization is configurable according to system needs:

AHB matrix priority register (AHBMATPRIO, address 0x4000 0004) bit description

Bit	Symbol	Description	Reset value
1:0	PRI_ICODE	I-Code bus priority (master 0). Should be lower than PRI_DCODE for proper operation.	0
3:2	PRI_DCODE	D-Code bus priority (master 1).	0
5:4	PRI_SYS	System bus priority (master 2).	0
7:6	-	Reserved. Read value is undefined, only zero should be written.	-
9:8	PRI_DMA	DMA controller priority (master 5).	0
13:10	-	Reserved. Read value is undefined, only zero should be written.	-
15:14	PRI_FIFO	System FIFO bus priority (master 9).	0
17:16	PRI_M0	Cortex-M0+ bus priority (master 10).	0
31:18	-	Reserved. Read value is undefined, only zero should be written.	-

Case study:AHB Matrix in LPC54114





Case study: AHB matrix in LPC43xx

Slave core memory allocation strategies

(Rare) Slave core code execute in flash, data in one block of RAM

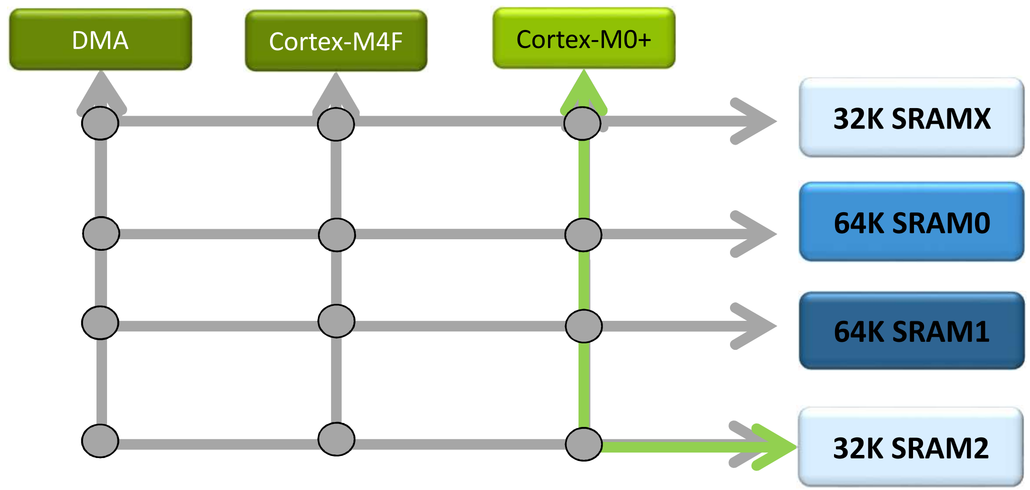
- Pros: Saves SRAM from copying slave code
- Cons:
  - Very poor performance if both core run simultaneously.
  - Flash consumes extra power.
- Suitable area: M4 is slave, or slave code size is big.

(Mainstream) Slave core has both code and data in the same SRAM block

- Pros:
  - Better performance,
  - lower power.
- Cons: Extra RAM required to store slave code.
- Suitable area: M0/M0+ is slave and code size is not very big.

Memory allocation examples on LPC541xx

- Example 1: M0+ is slave, and 32kB SRAM2 is for both code and data of M0+. Shared data can be put in any SRAM blocks.



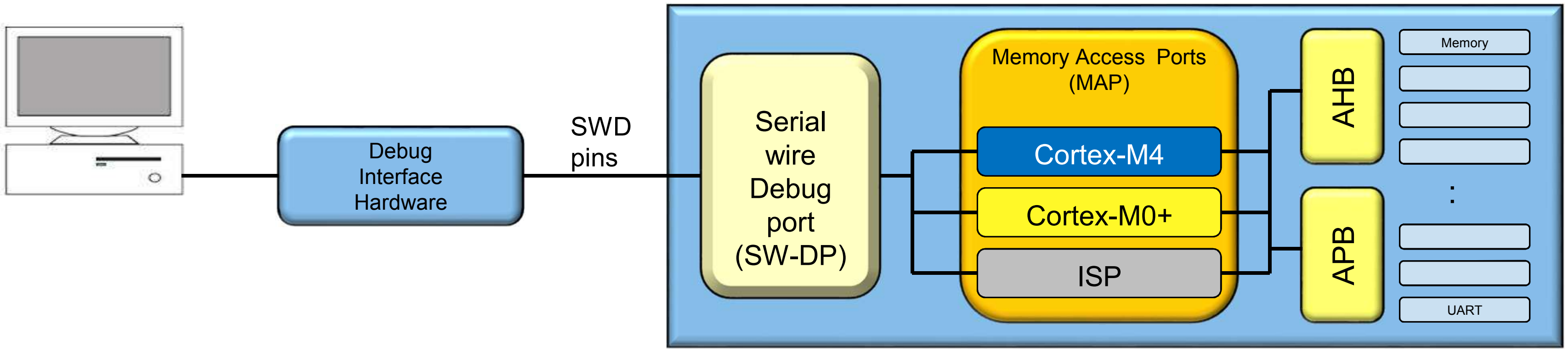
DUAL-CORE DEBUG INTERFACE

Dual-core debug overview

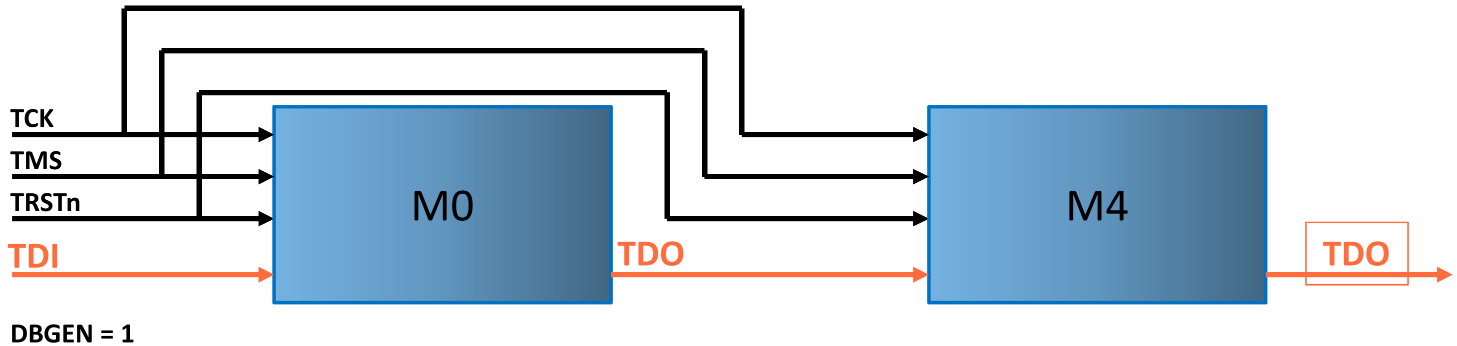
- Each core has its project, single project can be debugged as before.
- Debug architecture:
  - LPC541xx uses ARM's "Multi-drop SWD" technology to implement dual-core debug facility.
    - Multi-drop SWD is supported by Cortex-M CoreSight technologies introduced by ARM.
  - LPC43xx attaches both M4 and M0 inJTAG scan chain, and M4 can be debugged by SWD too.
- User can debug one project at a given time.
- In some cases, two projects for both core can be debugged simultaneously.

Debug Access Port Structure (LPC541xx)

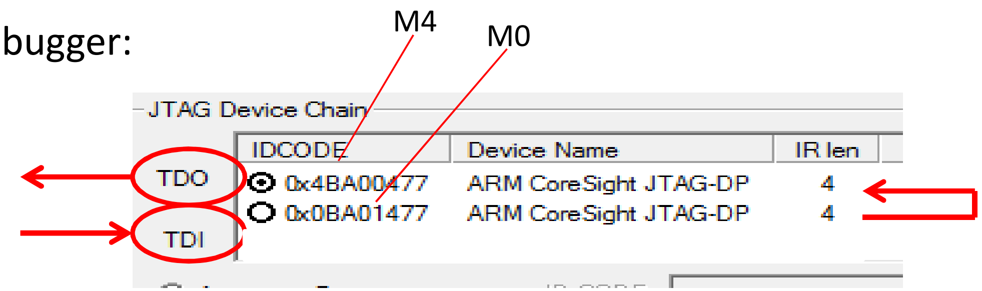
- The DAP can act as a bus master and can allow memory access to Advanced High-performance Bus (AHB) and Advanced Peripheral Bus (APB) even while the core is running.
- The busses are connected to Memory Access Ports (MEM-AP) of the DAP.
- M4 and M0+ each has its own access port.
- On LPC541xx, only **SWD** can be used to debug either core (JTAG isn't supported).



(LPC4300) Dual-core JTAG scan chain



Keil debugger:



LPC4300 takes another approach: Use JTAG scan chain to attach 2nd core.
This makes on LPC4300, only **JTAG** can be used to debug M0.

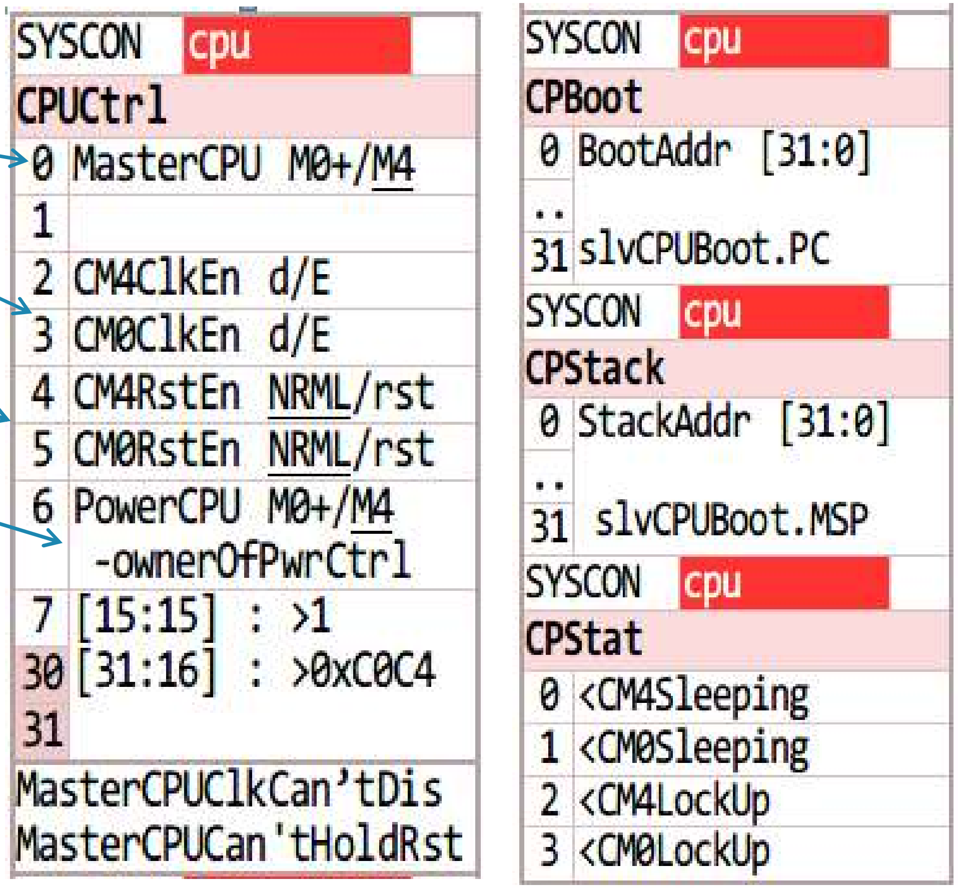
LPC541XX DUAL-CORE IMPLEMENTATION

LPC541xx Cortex-M4/M0+ Implementation Details

- **Cortex-M4:**
  - Memory protection Unit (MPU)
  - Single precision FPU
  - 3 bit interrupt priority levels
  - SysTick timer
  - VTOR register
  - Sleep mode power saving + extended modes
  - SWD with 8 breakpoints, 4 data watchpoints
- **Cortex-M0+:**
  - Multiply support in hardware
    - LPC5410x: 32 clocks per MUL
    - LPC5411x: 1 clock per MUL
  - SysTick timer
  - VTOR register
  - Sleep mode power saving + extended modes
  - SWD with 4 breakpoints and 2 data watchpoints

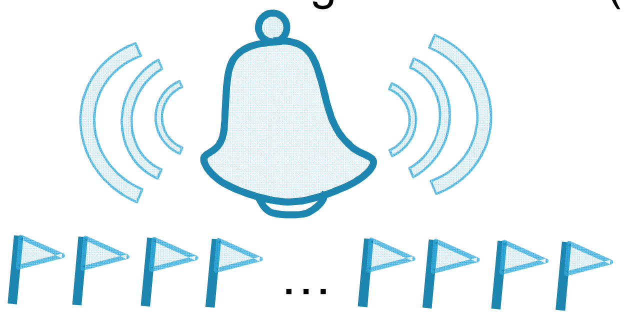
LPC541xx System control : Dual-core basic setup and control

- Core state control (SYSCON->**CPUCTRL**):
  - Determine which core is master
  - Gating the clock of current SLAVE core
    - Can't gate clock of current master core
  - Holding SLAVE core in reset
    - Can't reset master core in this way
  - Determine which core can initiate low power mode enter sequence.
- Startup parameters for slave core:
  - Initial stack top of slave (MSP initial value) (SYSCON->**CPSTACK**)
  - Reset vector of slave (vector 0) (SYSCON->**CPBOOT**)
  - Startup code uses these 2 registers to startup slave core.
- status of both core (SYSCON->**CPSTAT**)

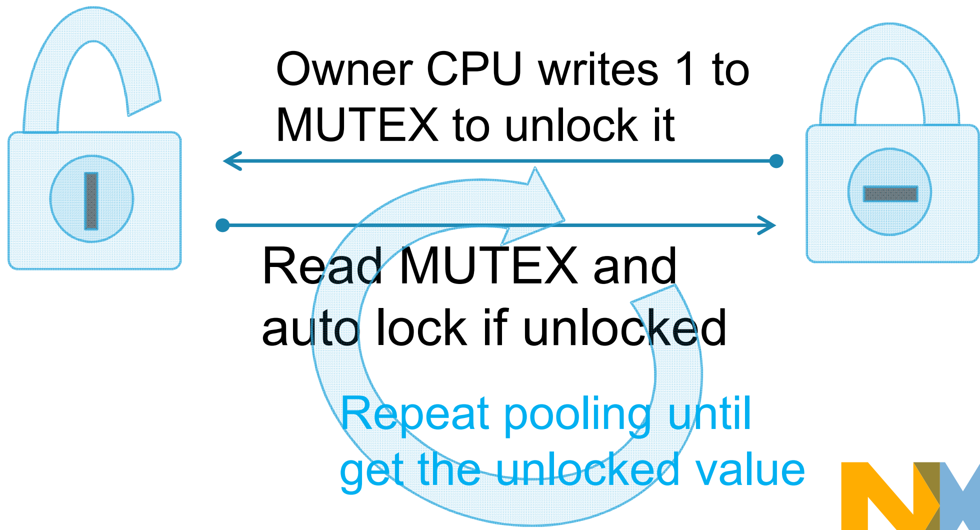


LPC541xx Mailbox: Helper peripheral for inter-core operations

- IRQ to each other core
  - Both cores' NVIC has the "Mailbox IRQ"
  - Writing non-zero to MAILBOX->**IRQ0** sets pending mailbox IRQ on **M0**
  - Writing non-zero to MAILBOX->**IRQ1** sets pending mailbox IRQ on **M4**
  - Value of IRQ0/1 is interpreted by user.
    - Both are 32 bit value.
    - Often used as flags or address (pointer).



- Hardware mutual exclusion (spin lock): MAILBOX->**MUTEX** register:
  - Read: return current value and automatically clear to 0 **within the same access**.
  - Write non 0 to restore



Related API

- `void Chip_CPU_CM0Boot(uint32_t *coentry, uint32_t *costackptr);`
  - Setup M0+ boot parameters (reset vector and initial stack top) and reset M0
- `Void Chip_MBOX_SetValue(LPC_MBOX_T *pMBOX, uint32_t cpu_id, uint32_t mboxData);`
  - Write mailbox IRQ0/1 registers, non-0 value **triggers mailbox IRQ** to opponent.
- `uint32_t Chip_MBOX_GetValue(LPC_MBOX_T *pMBOX, uint32_t cpu_id);`
- `void Chip_MBOX_SetMutex(LPC_MBOX_T *pMBOX);`
  - Set MUTEX to 1 to unlock the shared resource.
- `uint32_t Chip_MBOX_GetMutex(LPC_MBOX_T *pMBOX);`
  - Read MUTEX and AUTOMATICALLY lock.
  - Usually used in a while loop to implement a spin lock:
    - `while (Chip_MBOX_GetMutex(LPC_MBOX) == 0) {}`

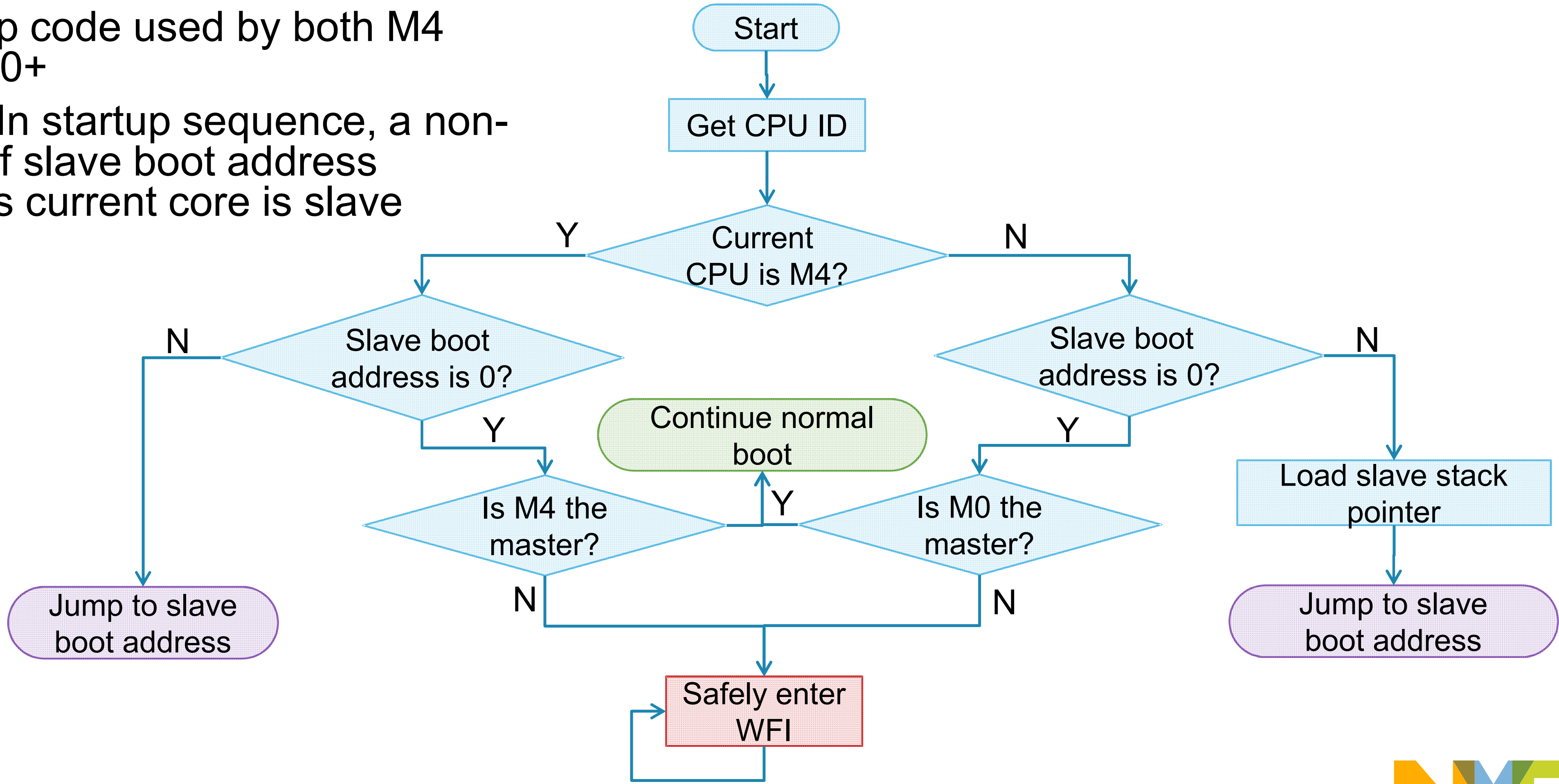
LPC541XX DUAL CORE BOOTSTRAP

Boot sequence

- Both cores startup after chip leaves reset
- Both cores fetch reset vector from flash address 0
- The initial reset vector (startup code) is shared by both cores
  - If startup code is only for M4, M0 will soon hard fault due to invalid instruction.
  - Hand written for M0+ instruction set
- Boot code jobs
  - Identify the current core: What is it, M4 or M0+
  - Check which core is master
  - Do master boot or slave boot according to the core and master settings

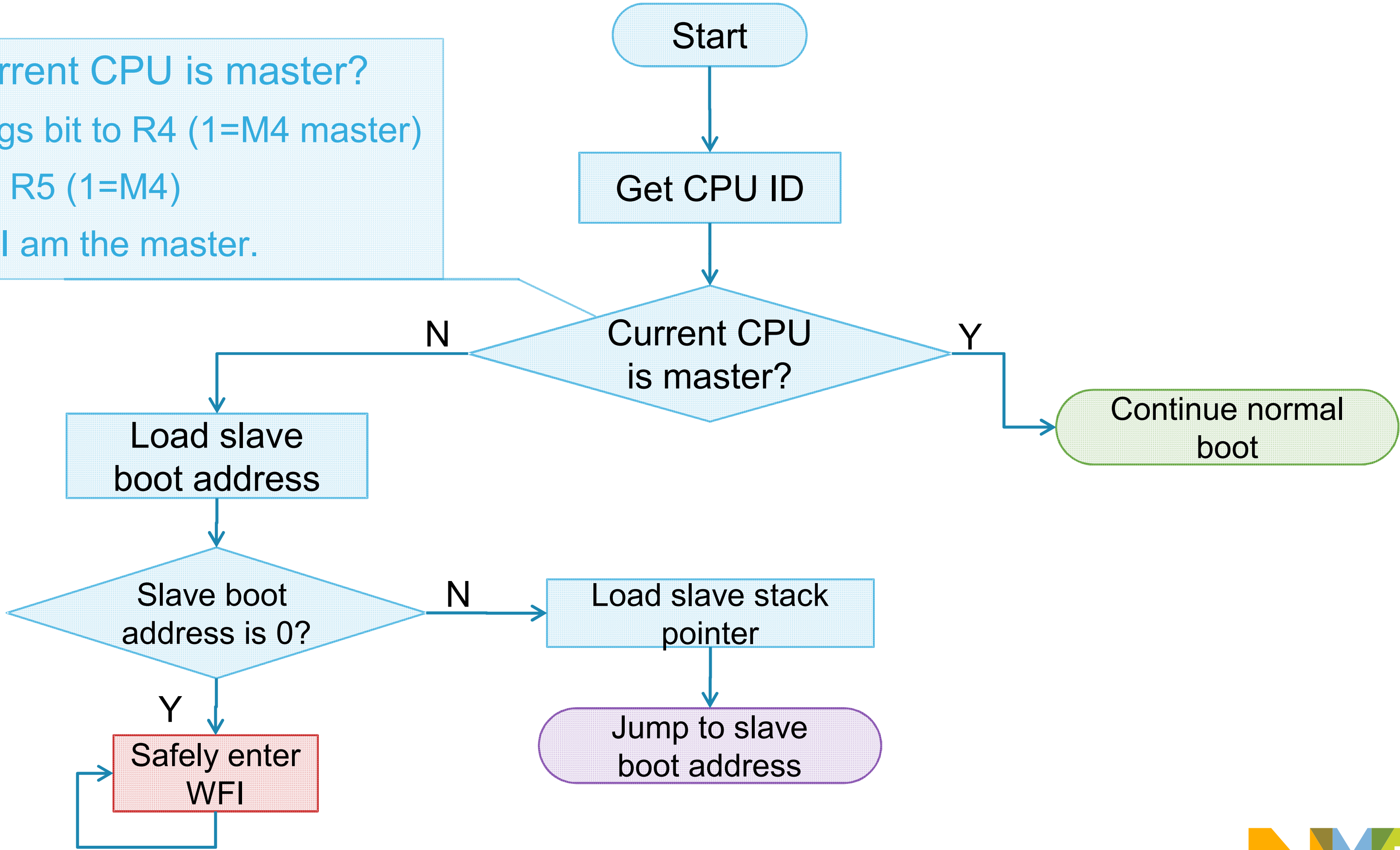
LPC541xx: Shared startup sequence (LPCXPresso)

- Startup code used by both M4 and M0+
- Note: In startup sequence, a non-zero of slave boot address implies current core is slave



LPC541xx: Shared startup sequence (KEIL/IAR)

- How to check if current CPU is master?
  - Put master settings bit to R4 (1=M4 master)
  - Put CPU ID bit to R5 (1=M4)
  - If R4 == R5 then I am the master.

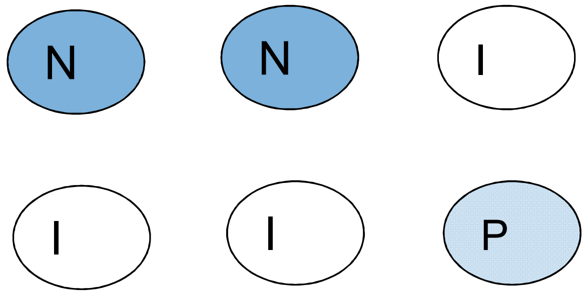


ADVANCED INTER-CORE OPERATIONS

Shared data implementation

- Share the same RAM range between two cores.
- User code define and interpret the data structure
- Define the “shared data” struct and put in a “.h” file.
  - Both projects need to include this header.
- Both project define exactly one instance of this struct
  - Must make sure the address of the shared object is manually set
  - Need to **configure linker to put the shared data to designated place**.
  - (LPC541xx): Can use IRQ0/1 registers to pass the address of shared data.
- Code can
  - **poll** the shared data periodically, or
  - **send IRQ** to the other when the code update some fields in the shared data.
- Don’t forget “volatile” keyword to ensure compiler always access true variable.

Extension of mailbox: Software Message pool



- Each item in pool is a message
  - Message items have fixed length.
  - Item has 3 status: New, Idle, Preprocessed
- Sender traverses the pool to scan idle item
  - Write message data to idle item and update item to "new"
- Receiver side:
  - ISR traverses the pool to scan new items, and do first step processing.
    - If ISR can process it completely, update the item status to "idle" again.
    - If ISR can't do all job, mark a "new" item to "preprocessed"
  - RTOS can then dispatch preprocessed items to tasks
- Drawbacks :
  - Message items can't be too many to make traverse time too long.
  - Risk: Insufficient items when messages are flooding.

Further Mutual exclusion besides hardware MUTEX

- For shared ring buffer/queue structures, enforce below constraints:
  - Only sender is allowed to modify write index
  - Only receiver is allowed to modify read index
- For message pool:
  - Only sender is allowed to change “idle” state to “new” state
  - Only receiver is allowed to change “new” state to “preprocessed” or “idle” state.
- When pointers are involved：Only one core is allowed to modify pointer
  - The other core can set some flags to ask pointer owner to modify pointer
- Note: LPC4300 does not support hardware MUTEX, above are only choices.

SECURE CONNECTIONS
FOR A SMARTER WORLD