ARM Cortex-M4 Architecture

Prof. Hugo Vieira Neto
2020/1

Objective

- To revise the main concepts of the ARM Cortex-M4 core:
  - Programmers model
  - Memory model
  - Load/Store architecture
  - Pipeline and conditional execution
  - ARM Architecture Procedure Call Standard

Architecture vs Organization

- Architecture = specification document
  - Instruction Set
  - Exceptions/Interrupts
  - Memory Model
  - Registers
  - Ex: ARMv4, ARMv7, etc.
- There is no cost, can be obtained directly from ARM's website

Architecture vs Organization

- Organization = physical implementation (silicon)
  - Ex: ARM7TDMI, ARM Cortex-M4, etc.
- ARM sells core implementations in VHDL or diffusion masks to licensed enterprises

ARM Cortex-M4

- ARMv7E-M architecture
- Thumb-2 instruction set
- 21 registers (32-bit)
- A single program status register
- Fixed memory map
  - Memory-mapped peripherals
- No MMU (Memory Management Unit) or cache memory

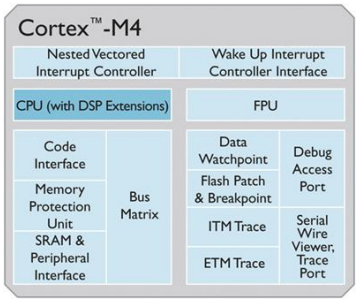
ARM Cortex-M4

- Interrupt controller is part of the processor core macrocell
- Interrupt vector table contains addresses, not instructions
- Interrupts automatically save and restore the processor state
  - Very efficient interrupt servicing
- Power management

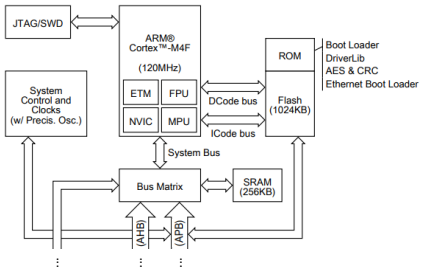
ARM Cortex-M4

- Designed to be programmed in C
  - Even interrupt servicing
- Support for operating systems (RTOS)
  - User/Supervisor Model
  - SVC, PendSV and SysTick exceptions
  - Memory Protection Unit (MPU)
- Cortex-M4F has a Floating Point Unit (FPU)

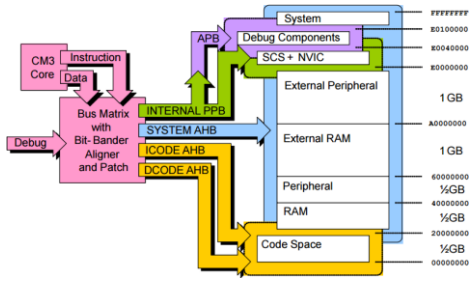
Cortex-M4 Simplified View



Block Diagram – TM4C1294



Cortex-M4 Memory Map



Memory Map – TM4C1294

Start	End	Description
0x0000 0000	0x000F FFFF	On-chip Flash (1MiB)
0x0010 0000	0x01FF FFFF	Reserved
0x0200 0000	0x02FF FFFF	On-chip ROM (16 MiB)
0x0300 0000	0x1FFF FFFF	Reserved
0x2000 0000	0x2003 FFFF	On-chip SRAM (256KiB)
0x2004 0000	0x21FF FFFF	Reserved
0x2200 0000	0x2234 FFFF	Bit-band alias of SRAM
0x2235 0000	0x3FFF FFFF	Reserved
0x4000 0000	0xDFFF FFFF	Peripherals
0xE000 0000	0xFFFF FFFF	Private Peripherals

Core Registers (32-bit)

- 13 general purpose registers
  - R0 to R7 (low registers)
  - R8 to R12 (high registers)
- 3 specific purpose registers
  - R13 = Stack Pointer (SP)
  - R14 = Link Register (LR)
  - R15 = Program Counter (PC)

Core Registers (32-bit)

- 5 special registers
  - xPSR = Program Status Register
  - PRIMASK = Priority Mask Register
  - FAULTMASK = Fault Mask Register
  - BASEPRI = Base Priority Mask Register
  - CONTROL = Control Register

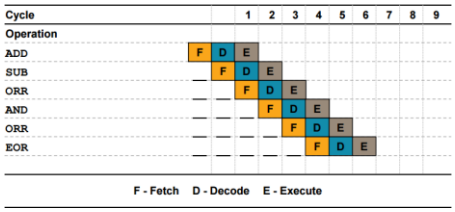
Load/Store Architecture

- Memory access:
  - Only LD instructions read data from memory
  - Only ST instructions write data to memory
  - Data processing instructions do not have access to memory
- Operations on data stored in memory require:
  - Read from memory
  - Operation (in register)
  - Write to memory

Three-stage Pipeline

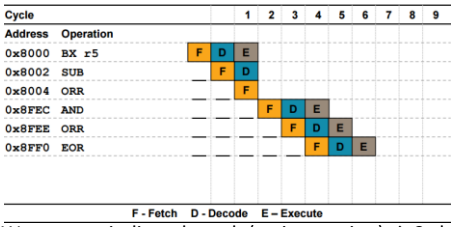
1. Fetch
   - Fetch instruction from memory
2. Decode
   - Decode the registers used in the instruction
3. Execute
   - Read registers
   - Perform operations
   - Write registers

Pipeline: Ideal Situation



- All operations performed in registers →
  6 instructions in 6 clock cycles (Cortex-M4)

Pipeline: Branch Effect



- Worst case: indirect branch (`BX` instrutcion) → 3 clock cycles to complete the branch (Cortex-M4)

Conditional Execution

- If-Then (IT) Block
  - Up to 3 conditional "then" (T) or "else" (E) instructions can be added to the block
  - Conditions the execution of up to 4 consecutive instructions

```
ITTET EQ
Inst 1
Inst 2
Inst 3
Inst 4
```

→

```
MOVEQ
ADDEQ
SUBNE
ORREQ
```

Exceptions and Interrupts

- Exceptions (faults, Debug, SVC, PendSV)
- 1 non-maskable interrupt (NMI)
- 1 SysTick interrupt
- 1 to 240 external interrupts with priority control
  - Implementation defines the number of interrupts
- Interrupt controller (NVIC) tightly coupled to the processor core

Power Management

- Sleep modes
  - Sleep Now
    - Wait for Interrupt (WFI) / Wait for Event (WFE)
  - Sleep On Exit
    - Immediately after the exit from the lowest priority interrupt service
  - Deep Sleep
    - Long term, PLL off
- Controlled by the NVIC

AAPCS

- Standardizes function calls in the ARM architecture (*ARM Architecture Procedure Call Standard*)
- Parameter passing to the function:
  - First parameters in R0, R1, R2 and R3
  - Other parameters in the stack
- Function return value:
  - R0 (32-bit)
  - R1:R0 (64-bit)

AAPCS

- Strongly related to automatic context saving in interrupts
- Registers R0, R1, R2, R3 and R12 belong to the called function
- Other registers (R4, R5, R6, R7, R8, R9, R10 and R11) belong to the calling function
- Stack alignment must be 64-bit :
  - Even number of registers in PUSH/POP

AAPCS

- Who has the obligation to save the contents of the registers used in a function?
  - Called function must save R4 to R11 in the stack before changing their contents (attention to 64-bit alignment!)
  - Called function can freely use R0 to R3 and R12 (must be saved in the stack, if necessary, by the calling function)

Extraclass Activity

- Reading:
  - ARM standard: “Procedure Call Standard for the ARM Architecture (AAPCS)” (Section 5)
  - Book: “The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors” (Chapter 8)
  - Book: “The Designer’s Guide to The Cortex-M Processor Family” (Chapter 3, with special attention to Sections “Priority and Preemption” and “Exception Model”)

Extraclass Activity

- Analyze the “c_asm” project in the “EK-TM4C1294XL_IAR” workspace
  - Check with the debugger how the parameters are passed to the called function (Assembly) and how the return value is passed to the calling function (C language)
  - Experiment with passing different amounts of parameters, with different sizes (8, 16, 32, 64 bits), to the function implemented in Assembly