COEN-4720 Embedded Systems Design
Lecture 3
**Intro to ARM Cortex-M3 (CM3) and LPC17xx MCU**

Cristinel Ababei
Dept. of Electrical and Computer Engineering
Marquette University

1

# Outline

- Overview of ARM Cortex-M3 processor
- NXP LPC17xx microcontroller unit (MCU)

2

## Cortex-M3 Processor

- RISC general purpose 32-bit microprocessor
- Released in 2006
- Cortex-M3 differs from previous generations of ARM processors by defining a number of key peripherals as part of the core:
  - interrupt controller
  - system timer
  - debug and trace hardware (including external interfaces)
- This enables for real-time operating systems and hardware development tools such as debugger interfaces be common across the family of processors
- Various Cortex-M3 based microcontroller families differ significantly in terms of hardware **peripherals and memory**

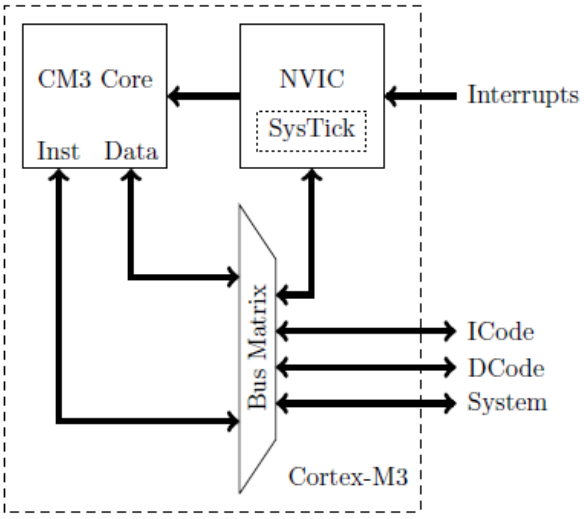
3

## Cortex-M3 Processor

- **Greater performance efficiency**: more work to be done without increasing the frequency or power requirements
  - Implements the new Thumb-2 instruction set architecture
    - 70% more efficient per MHz than an ARM7TDMI-S processor executing Thumb instructions
    - 35% more efficient than the ARM7TDMI-S processor executing ARM instructions for Dhrystone benchmark
- **Low power consumption**: longer battery life, especially critical in portable products including wireless networking applications
- **Improved code density**: code fits in even the smallest memory footprints
- Core pipeline has 3 stages
  - Instruction Fetch
  - Instruction Decode
  - Instruction Execute

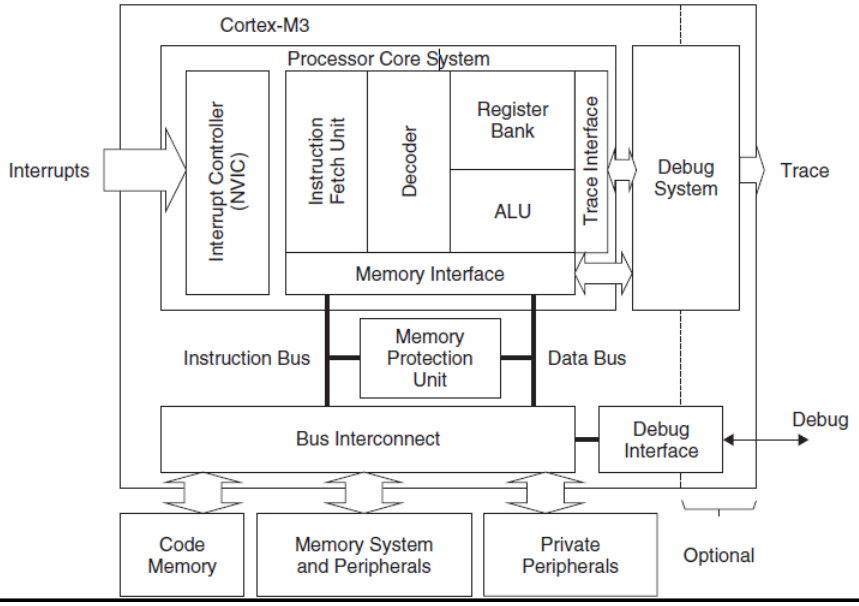
4

# Simplified Cortex-M3 Architecture




5

# Simplified Cortex-M3 Architecture




6

## Cortex-M3 Processor Architecture

- Harvard architecture: it uses separate interfaces to fetch instructions (Inst) and (Data)
- Processor is not memory starved: it permits accessing data and instruction memories simultaneously
- **From CM3 perspective, everything looks like memory**
  - Only differentiates between instruction fetches and data accesses
- Interface between CM3 and manufacturer specific hardware is through three memory buses:
  - ICode, DCode, and System (for peripherals), which are defined to access different regions of memory

7

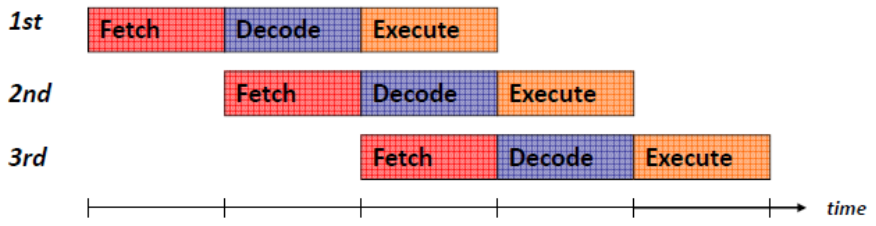
## Cortex-M3 Processor

- Cortex-M3 is a load/store architecture with three basic types of instructions
  1. **Register-to-register** operations for processing data
  2. **Memory operations** which move data between memory and registers
  3. **Control flow** operations enabling programming language control flow such as if and while statements and procedure calls

8

# Cortex-M3 Pipeline

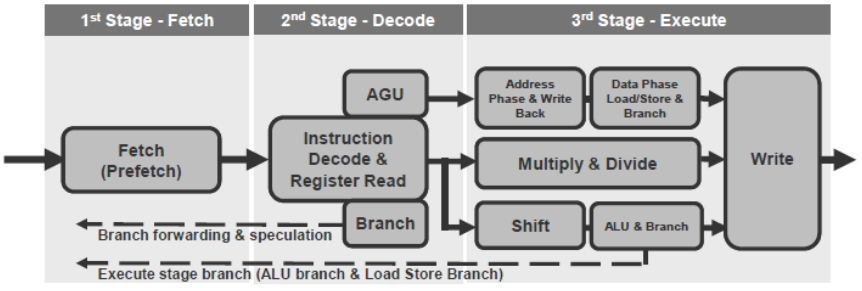
- The Cortex-M3 Uses the 3-stage pipeline for instruction executions
  - Fetch ⇒ Decode ⇒ Execute
  - Pipeline design allows effective throughput to increase to one instruction per clock cycle
  - Allows the next instruction to be fetched while still decoding or executing the previous instructions




9

# Cortex-M3 Pipeline

- Cortex-M3 has 3-stage fetch-decode-execute pipeline
  - Similar to ARM7
  - Cortex-M3 does more in each stage to increase overall performance




This is Slide #27 of "ARM Cortex-M3 Introduction, ARM University Relations". Download from:
http://www.arm.com/files/pdf/CortexM3_Uni_Intro.pdf

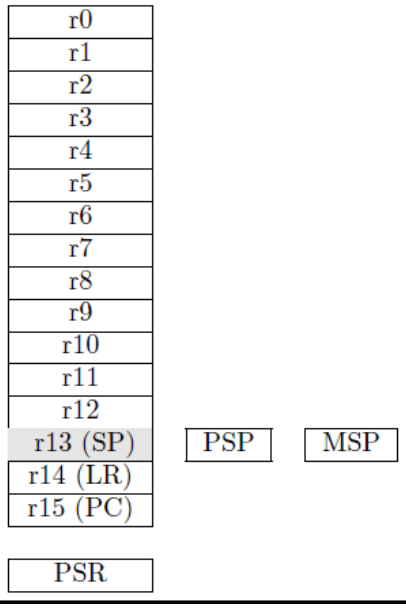
10

# Processor Register Set

- Cortex-M3 core has **16 user-visible registers**
  - All processing takes place in these registers
- Three of these registers have dedicated functions
  - **program counter (PC)** - holds the address of the next instruction to execute
  - **link register (LR)** - holds the address from which the current procedure was called
  - **"the" stack pointer (SP)** - holds the address of the current stack top (CM3 supports multiple execution modes, each with their own private stack pointer).
- **Processor Status Register (PSR)** which is implicitly accessed by many instructions

11

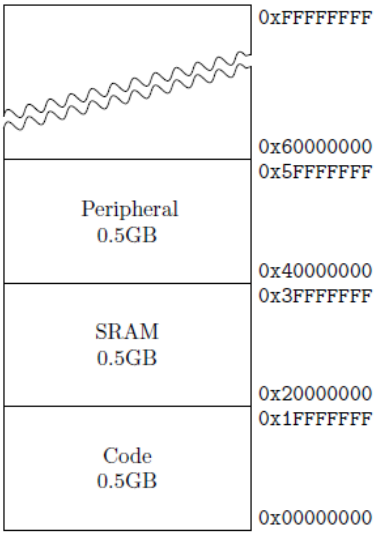
# Processor Register Set




12

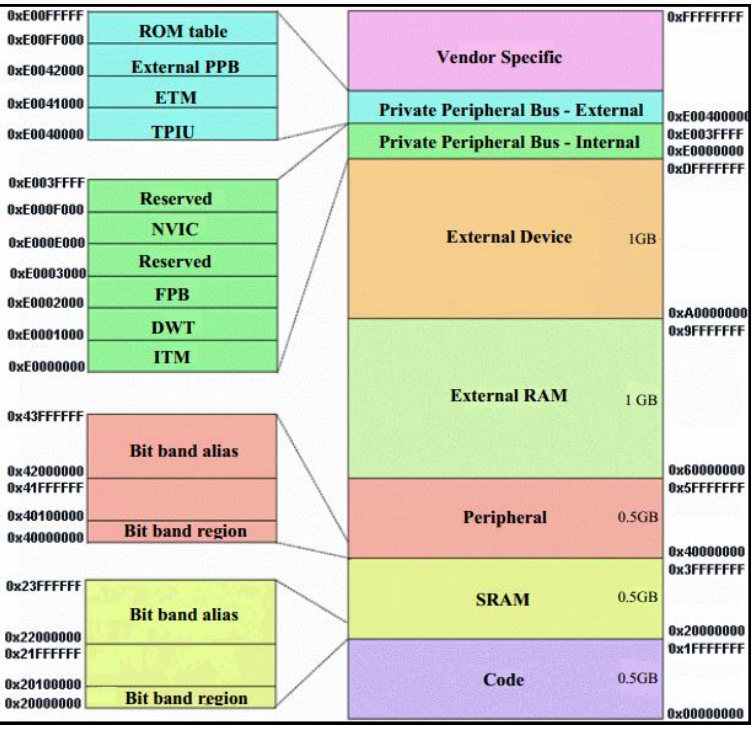
# Cortex-M3 Memory Address Space

- ARM Cortex-M3 processor has a single 4 GB address space
- The **SRAM and Peripheral** areas are accessed through the System Bus
- The **"Code" region** is accessed through the ICode (instructions) and DCode (constant data) buses




13

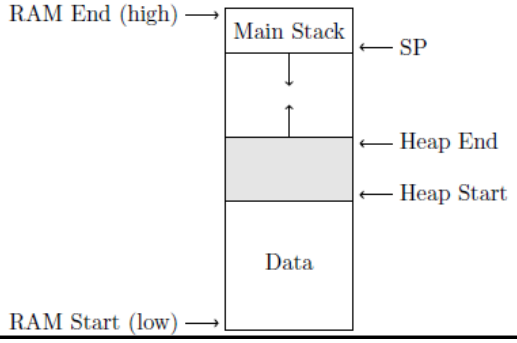
# Memory Map




14

## Program Memory Model

- RAM for an executing program is divided into three regions
  1. Data in RAM are allocated during the link process and initialized by startup code at reset
  2. The (optional) heap is managed at runtime by library code implementing functions such as the malloc and free which are part of the standard C library
  3. The stack is managed at runtime by compiler generated code which generates per-procedure-call stack frames containing local variables and saved registers




15

## Operating Modes

- Cortex-M3 processor has two modes and two privilege levels
- The operation modes - determine whether the processor is running a normal program or running an exception handler
  - thread mode
  - handler mode
- The privilege levels - provide a mechanism for safeguarding memory accesses to critical regions as well as providing a basic security model
  - privileged level
  - user level

| | *Privileged* | *User* |
|---|---|---|
| *When running an exception handler* | Handler mode | |
| *When not running an exception handler (e.g., main program)* | Thread mode | Thread mode |

16

## Nested Vector Interrupt Controller (NVIC)

- A programmable device that sits between the CM3 core and the microcontroller
- CM3 uses a prioritized vectored interrupt model – the vector table is defined to reside starting at memory location 0
- First 16 entries in this table are defined for all Cortex-M3 implementations while the remainder, up to 240, are implementation specific
- NVIC supports dynamic redefinition of priorities with up to 256 priority levels
- Two entries in the vector table are especially important:
  - address 0 contains the address of the initial stack pointer
  - address 4 contains the address of the "reset handler" to be executed at boot time

17

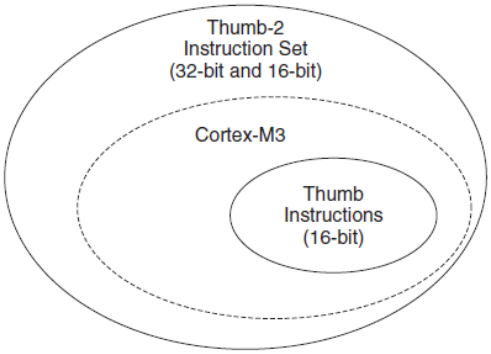
## Nested Vector Interrupt Controller (NVIC)

- Provides key system control registers including the System Timer (SysTick) that provides a regular timer interrupt
- Provision for a built-in timer across the Cortex-M3 family has the significant advantage of making operating system code highly portable – all operating systems need at least one core timer for time-slicing
- Registers used to control the NVIC are defined to reside at address 0xE000E000 and are defined by the Cortex-M3 specification
- These registers are accessed with the system bus

18

# Thumb-2 Instruction Set

- Thumb-2 instruction set is a superset of the previous 16-bit Thumb instruction set
- Provides
  - A large set of 16-bit instructions, enabling 2 instructions per memory fetch
  - A small set of 32-bit instructions to support more complex operations
- **Specific details of this ISA not our focus (we'll mostly program in C)**



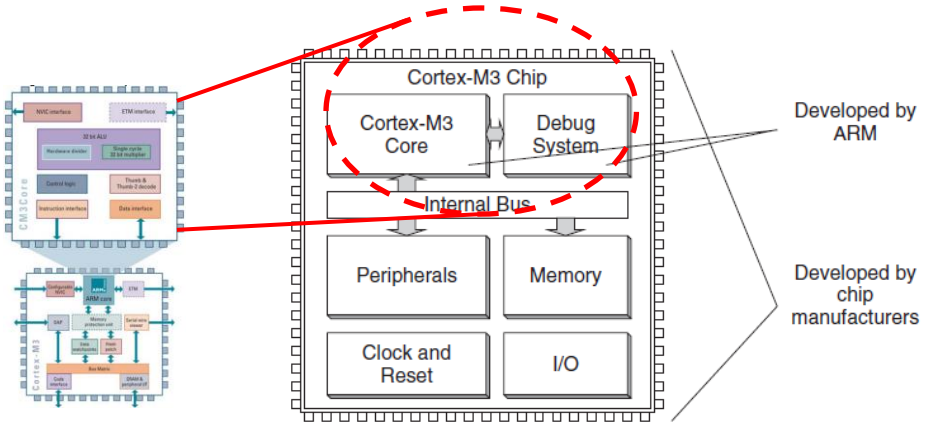

19

# Outline

- Overview of ARM Cortex-M3 processor
- NXP LPC17xx microcontroller unit (MCU)

20

## Cortex-M3 Processor vs. CM3-based Microcontroller Units




ARM NXP, TI, ST, etc.

21

While there is significant overlap between the families and their peripherals, there are also important differences

**In the lab of this course we focus on the NXP's LPC17xx family**

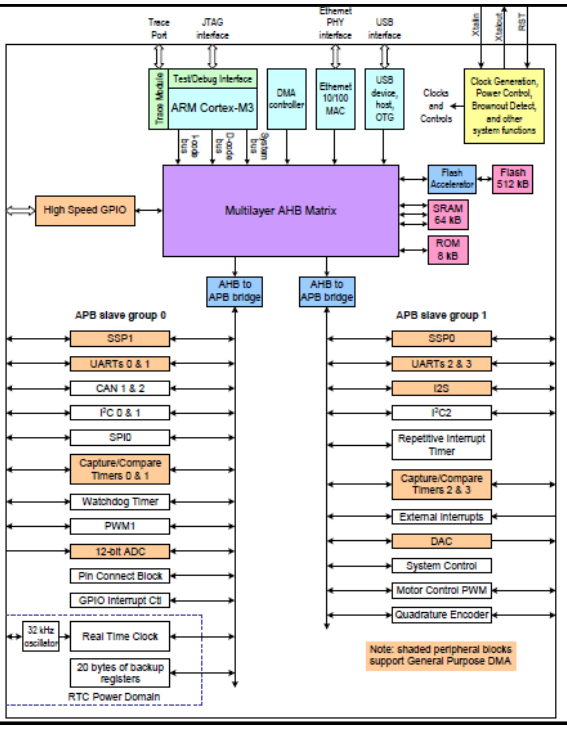
22

# LPC17xx

- LPC17xx (of NXP) is an ARM Cortex-M3 based microcontroller
- The Cortex-M3 is also the basis for microcontrollers from other manufacturers including TI, ST, Toshiba, Atmel, etc.
- LPC1768 operates at up to a 100 MHz CPU frequency
- Sophisticated clock system
- Peripherals include:
  - up to 512 kB of flash memory, up to 64 kB of data memory
  - Ethernet MAC
  - a USB interface that can be configured as either Host, Device, or OTG
  - 8 channel general purpose DMA controller
  - 4 UARTs, 2 CAN channels, 2 SSP controllers, SPI interface
  - 3 I2C interfaces, 2-input plus 2-output I2S interface
  - 8 channel 12-bit ADC, 10-bit DAC, motor control PWM
  - Quadrature Encoder interface, 4 general purpose timers,
  - 6-output general purpose PWM
  - ultra-low power RTC with separate battery supply
  - up to 70 general purpose I/O pins

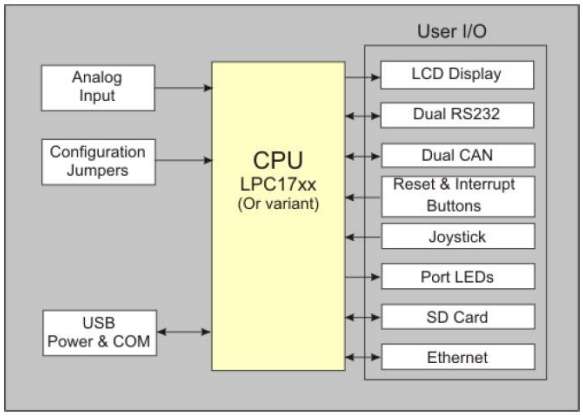
23

# LPC1768

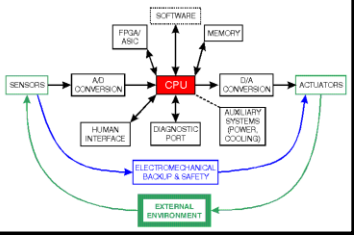



24

## Abstract Representation of a Development Board (such as LandTiger 2.0)




**Recall from lecture#1:**




25

## LPC1768

- LPC1768 microcontrollers are based on the Cortex-M3 processor with a set of peripherals distributed across three buses – Advanced High-performance Bus (AHB) and its two Advanced Peripheral Bus (APB) sub-buses APB1 and APB2.
- These peripherals:
  - are controlled by the CM3 core with load and store instructions that access **memory mapped registers**
  - can "interrupt" the core to request attention through peripheral specific interrupt requests routed through the NVIC
- Data transfers between peripherals and memory can be automated using DMA
- Labs cover among others:
  - basic peripheral configuration
  - how interrupts can be used to build effective software
  - how to use DMA to improve performance and allow processing to proceed in parallel with data transfer

26

# LPC1768

- **Peripherals are "memory-mapped"**
  - core interacts with the peripheral hardware by reading and writing peripheral "registers" using load and store instructions
- The various peripheral registers are documented in the user and reference manuals
  - documentation include bit-level definitions of the various registers and info on how to interpret those bits
  - actual physical addresses are also found in the reference manuals
- Examples of base addresses for several peripherals (**see page 14 of the LPC17xx user manual**):

```
0x40010000 UART1
0x40020000 SPI
0x40028000 GPIO interrupts
0x40034000 ADC
…
```

- No real need for a programmer to look up all these values as they are defined in the library file **lpc17xx.h** as:

```
LPC_UART1_BASE
LPC_SPI_BASE
LPC_GPIOINT_BASE
LPC_ADC_BASE
…
```

27

# LPC1768

- Typically, each peripheral has:
  1. **Control registers** to configure the peripheral
  2. **Status registers** to determine the current peripheral status
  3. **Data registers** to read data from and write data to the peripheral

28

# LPC1768

- In addition to providing the addresses of the peripherals, **lpc17xx.h** also provides C language level structures that can be used to access each peripheral
- For example, the SPI and GPIO ports are defined by the following register structures:

```
typedef struct
{
  __IO uint32_t SPCR;
  __I  uint32_t SPSR;
  __IO uint32_t SPDR;
  __IO uint32_t SPCCR;
       uint32_t RESERVED0[3];
  __IO uint32_t SPINT;
} LPC_SPI_TypeDef;
```

29

# LPC1768

```
typedef struct
{
  union {
    __IO uint32_t FIODIR;
    struct {
      __IO uint16_t FIODIRL;
      __IO uint16_t FIODIRH;
    };
    struct {
      __IO uint8_t  FIODIR0;
      __IO uint8_t  FIODIR1;
      __IO uint8_t  FIODIR2;
      __IO uint8_t  FIODIR3;
    };
  };
  uint32_t RESERVED0[3];
  union {
    __IO uint32_t FIOMASK;
    struct {
      __IO uint16_t FIOMASKL;
      __IO uint16_t FIOMASKH;
    };
    struct {
      __IO uint8_t  FIOMASK0;
      __IO uint8_t  FIOMASK1;
      __IO uint8_t  FIOMASK2;
      __IO uint8_t  FIOMASK3;
    };
  };
  union {
    __IO uint32_t FIOPIN;
    struct {
      __IO uint16_t FIOPINL;
      __IO uint16_t FIOPINH;
    };
    struct {
      __IO uint8_t  FIOPIN0;
      __IO uint8_t  FIOPIN1;
      __IO uint8_t  FIOPIN2;
      __IO uint8_t  FIOPIN3;
    };
  };
  union {
    __IO uint32_t FIOSET;
    struct {
      __IO uint16_t FIOSETL;
      __IO uint16_t FIOSETH;
    };
    struct {
      __IO uint8_t  FIOSET0;
      __IO uint8_t  FIOSET1;
      __IO uint8_t  FIOSET2;
      __IO uint8_t  FIOSET3;
    };
  };
  union {
    __O  uint32_t FIOCLR;
    struct {
      __O  uint16_t FIOCLRL;
      __O  uint16_t FIOCLRH;
    };
    struct {
      __O  uint8_t  FIOCLR0;
      __O  uint8_t  FIOCLR1;
      __O  uint8_t  FIOCLR2;
      __O  uint8_t  FIOCLR3;
    };
  };
} LPC_GPIO_TypeDef;
```

30

# LPC1768

- The register addresses of the various ports are defined in the library (see **lpc17xx.h**):

```
#define LPC_APB0_BASE          (0x40000000UL)
#define LPC_UART1_BASE         (LPC_APB0_BASE + 0x10000)
#define LPC_ADC_BASE           (LPC_APB0_BASE + 0x34000)
…
#define LPC_GPIO_BASE          (0x2009C000UL)
#define LPC_GPIO1_BASE         (LPC_GPIO_BASE + 0x00020)
#define LPC_GPIO1   ((LPC_GPIO_TypeDef *) LPC_GPIO1_BASE)
…
```

- For example, to turn on the LED marked as D11 on the LandTiger 2.0 board (which is driven by the pin P2.1 of the MCU), the following code can be used:

```
LPC_GPIO1->FIOSET |= 1 << 1;
```

- CHECK the Datasheet of LPC1768, page #131, NOW!!!

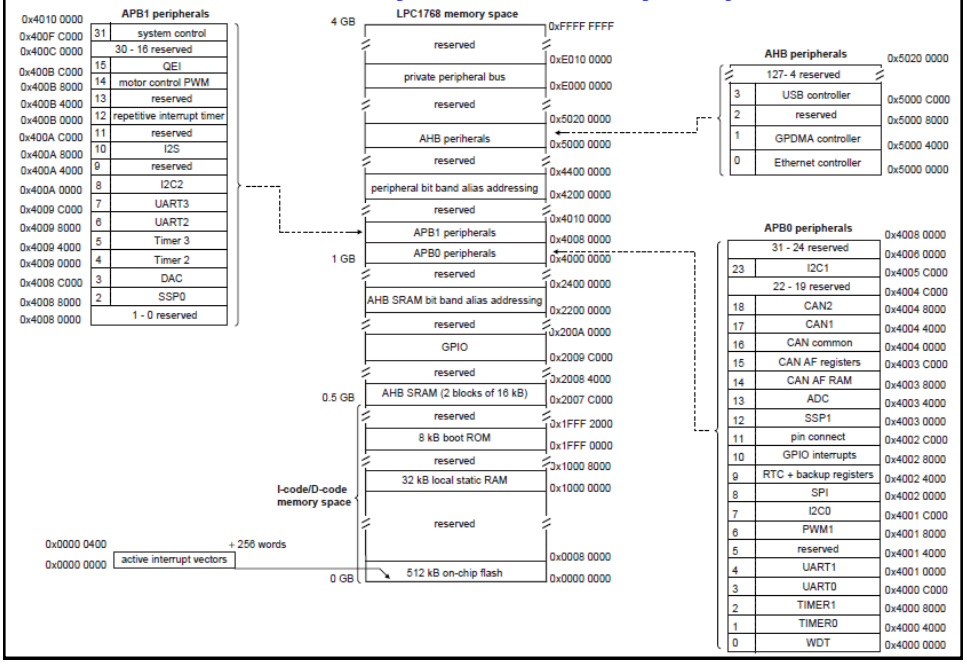
31

# Memory

- On-chip flash memory system
  - Up to 512 kB of on-chip flash memory
  - Flash memory accelerator maximizes performance for use with the two fast AHB-Lite buses
  - Can be used for both code and data storage
- On-chip Static RAM
  - Up to 64 kB of on-chip static RAM memory
  - Up to 32 kB of SRAM, accessible by the CPU and all three DMA controllers are on a higher-speed bus
  - Devices with more than 32 kB SRAM have two additional 16 kB SRAM blocks

32

## LPC17xx system memory map




33

## References & Credits

- Joseph Jiu, The Definitive guide to the ARM Cortex-M3, 2007
- LPC17xx microcontroller USER MANUAL
- Cortex-M3 Processor TECHNICAL REFERENCE MANUAL
- Lab manual (G. Brown, Indiana)
- EECS-373, UMich

**See website of class for links to download any of the above:**
**http://dejazzer.com/coen4720/index.html**

34