# COE718: Embedded Systems Design

## Lecture 4:
## Multitasking with ARM (RTX/CMSIS)

Anita Tino

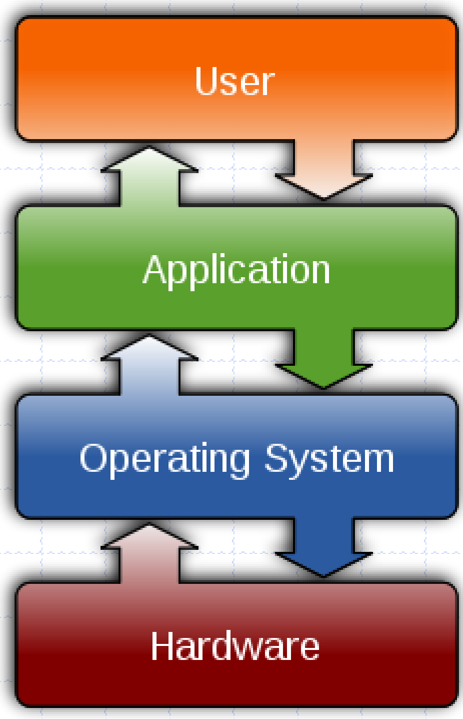
# Embedded System Applications

- Embedded systems (ES) address problems by decomposing an application into smaller pieces
  - Smaller pieces = processes or tasks
- These tasks must work together to produce the ES's functionality.

# Operating Systems

- Computer program that provides a software layer between the application software and the hardware
- Provides 3 major functions:

1. Schedule task execution
2. Dispatch a task to run
3. Ensure communication and synchronization between tasks



# Operating Systems

- When do we need an operating system in embedded systems?

# Operating Systems

- When do we need an operating system in embedded systems?
  - Certain applications only require **bare metal** - i.e. pure hardware implementation (as we've been doing so far)
  - Bare metal develops an application in one *super-loop* which executes its functions in a fixed order
    - If we have a critical task, then we typically use ISRs

# Operating Systems

- But what happens when we have multiple tasks that must be interleaved, need to assign task priorities etc?

# Operating Systems

- But what happens when we have multiple tasks that must be interleaved, need to assign task priorities etc?
  - We require an operating system which can manage, schedule and synchronize tasks and their data.

# The problem with OS and ES

- Typical OS have non-deterministic delays (due to various factors).
- In real-time systems however we need deterministic delays since response times are critical
- Therefore we use RTOS which are catered to real-time application requirements

# RTOS

- OS designed to serve real-time application processes and threads with deterministic delays
- often just consists of a OS kernel (nothing fancy, no user interface etc)
- Provides: task scheduling, task dispatching, and inter-task communication

# RTOS

- RTOS have 3 requirements:

1. Timing behaviour must be predictable - short and deterministic times, predictable memory accesses.

**Late answer = wrong answer**

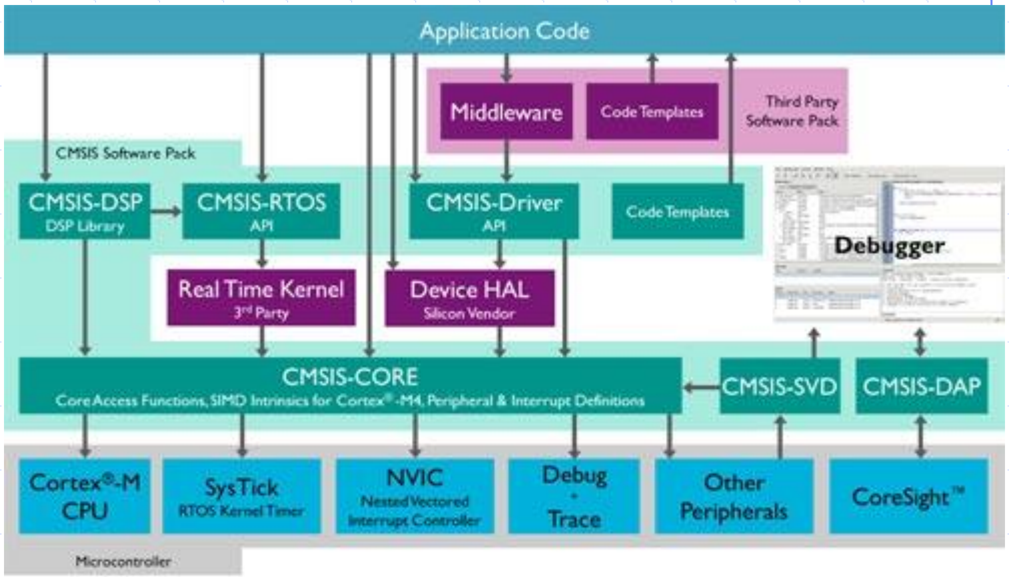
# RTOS

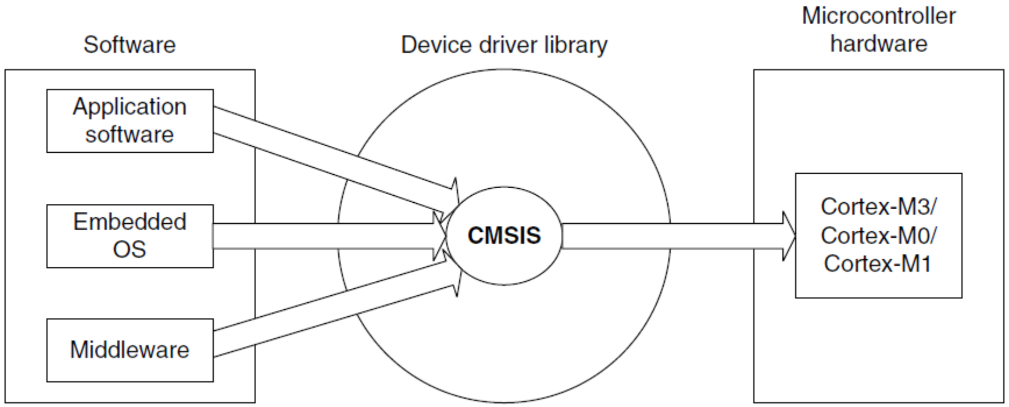
2. Must manage timing and scheduling of task - must be aware of task deadlines, and provide precise time services

3. Must be fast - avoid standard OS calls, memory-related delays etc

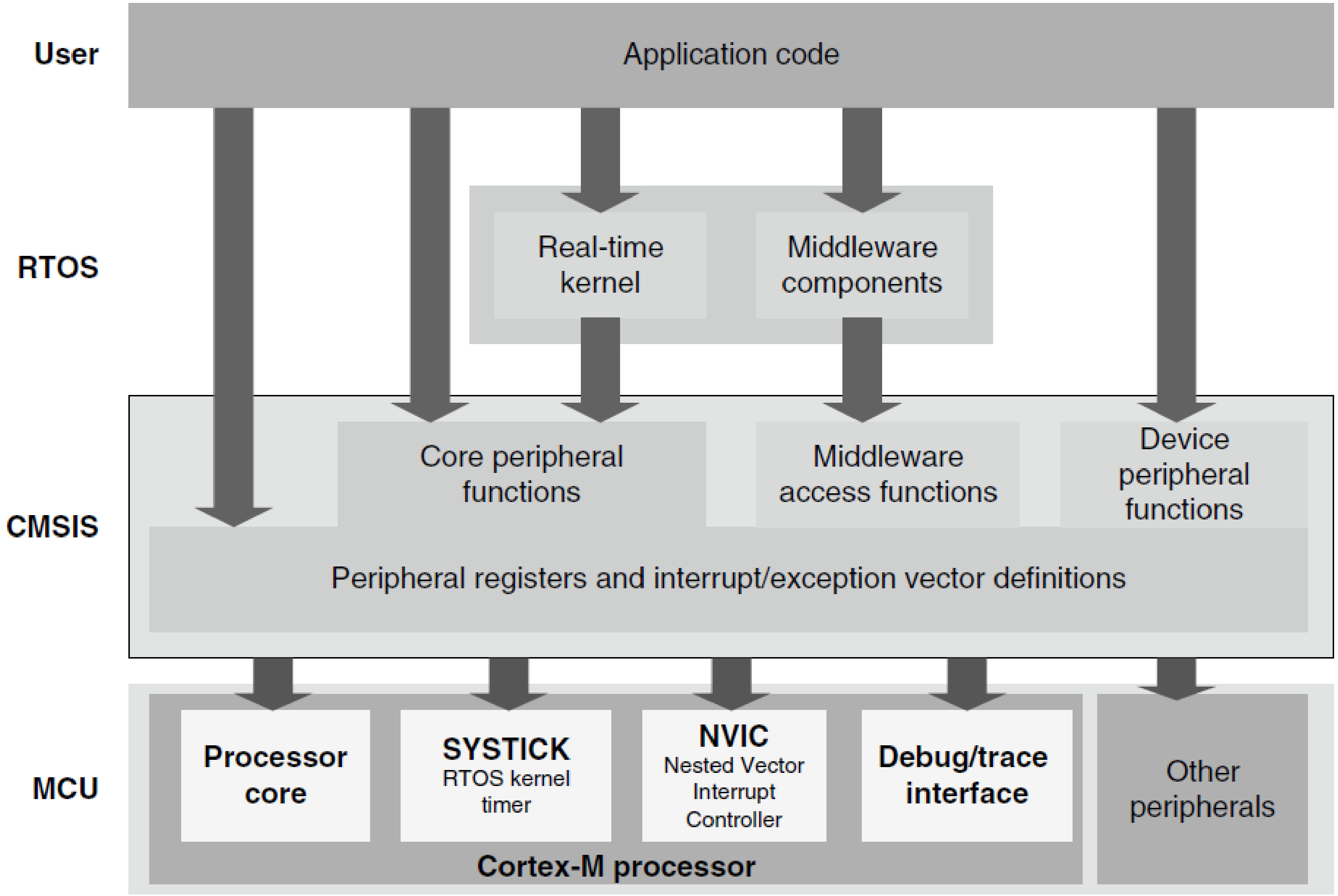
# CMSIS

- Cortex Microcontroller Software Interface Standard
- Device driver library
  - independent hardware abstraction layer used for interfacing applications to the uC

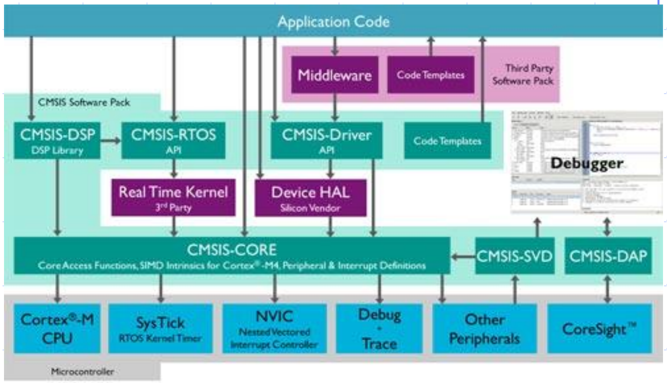




# CMSIS

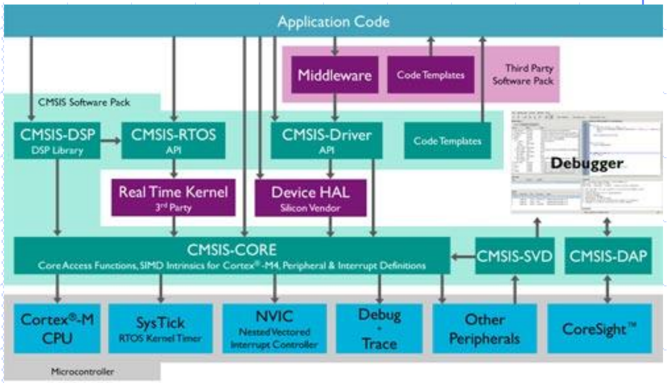

User
Application code
RTOS
Real-time kernel
Middleware components
CMSIS
Core peripheral functions
Middleware access functions
Device peripheral functions
Peripheral registers and interrupt/exception vector definitions
MCU
Processor core
SYSTICK
RTOS kernel timer
NVIC
Nested Vector Interrupt Controller
Debug/trace interface
Other peripherals
Cortex-M processor

# RTX RTOS



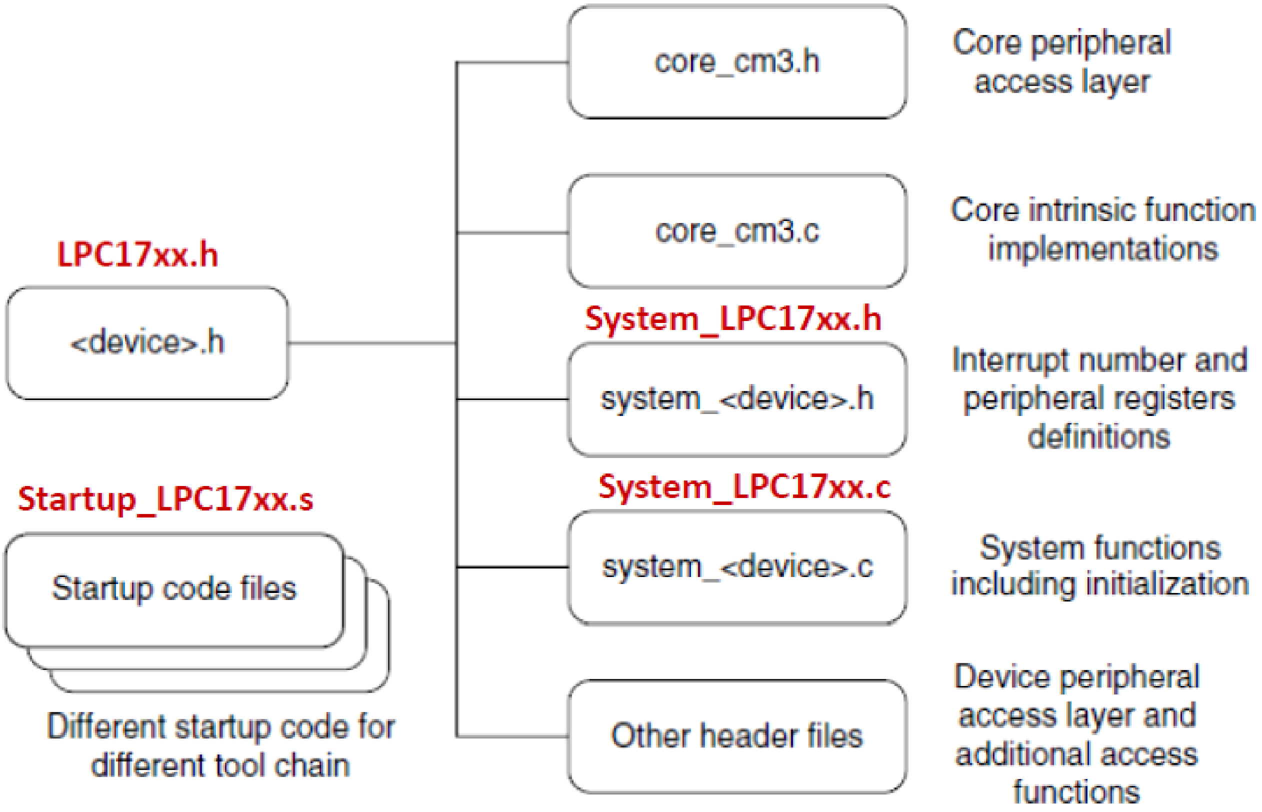
- Will be using RTX kernel = RTOS
- RTX = Keil's **R**eal **T**ime e**X**ecutive for ARM CPUs
- <RTL.h> file defines the RTX functions and macros we need to declare tasks and access all RTOS features
  - Offers interrupt handling, multitasking, periodic task activations, scalable task creation

# RTX RTOS



- Use RTX_Config_CM.c to specify paramaters and configuration in the RTOS/RTX kernel (lab3a and b)
  - Ports the kernel to your CPU
  - Includes cmsis_os.h
- Include cmsis_os.h so that your application (.c) may access the CMSIS RTOS API
  - Explicitly used in lab 3b for thread management

# CMSIS Files


core_cm3.h
Core peripheral access layer
core_cm3.c
Core intrinsic function implementations
LPC17xx.h
<device>.h
System_LPC17xx.h
system_<device>.h
Interrupt number and peripheral registers definitions
System_LPC17xx.c
system_<device>.c
System functions including initialization
Startup_LPC17xx.s
Startup code files
Different startup code for different tool chain
Other header files
Device peripheral access layer and additional access functions

# Creating Tasks with RTX

```
#include <stdio.h>
#include "LPC17xx.h"
#include <RTL.h>

long global_c1 = 0, global_c2 = 0;

__task void task1(void){
    for(;;){
           global_c1 += 3;
    }
}
__task void task2(void){
    for(;;){
           global_c2 += 2;
    }
}
```

```
int main(void){
    SystemInit();
    os_tsk_create(task1, 1);
    os_tsk_create(task2, 1);
    os_tsk_delete_self();

    os_sys_init(task1);
}
```