RTOS and RTX

Content of this Module

- We cover the following topics in this module

  - Why RTOS
  - Compare with Super loop
  - RTX
  - CMSIS

Real-Time Operating Systems

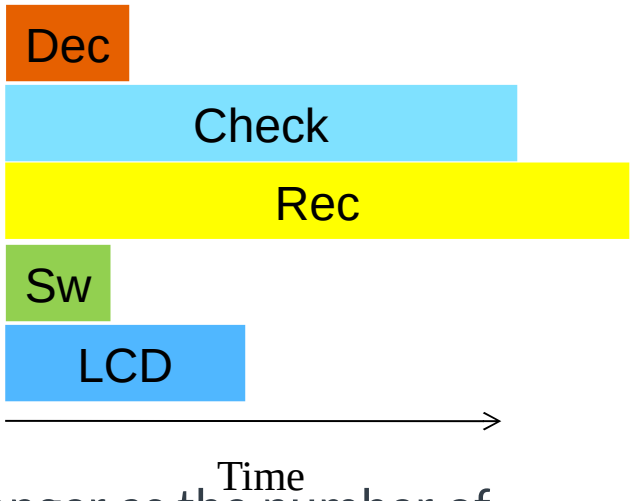
- Real-time operating systems (RTOS) are designed to meet harsh timing constraints
  - Hard real-time – critical tasks which have to be completed on time
  - Soft real-time – may continue finishing the task even missing the deadline
- Industrial applications: robots, aircraft control …
- Key design requirements:
    - Predictability and determinism
    - Speed – fast enough while keeping high predictability and determinism
    - Responsive to user control
    - Fail-safety
  - Advanced scheduling and memory allocation

Why RTOS on embedded systems?

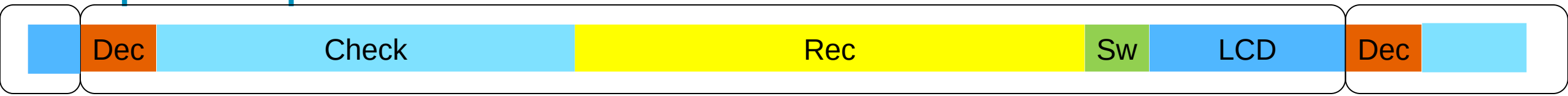
- Although it is possible to implement everything in a huge sequential loop…
  - Uses lengthy interrupt service routine (ISR)
  - Needs to keep synchronization between ISRs
  - Poor predictability (nested ISRs) and extensibility
  - Change of the ISR or the Super-Loop ripple through the entire system
- RTOS: all computation requests are encapsulated into tasks and scheduled on demand
  - Better program flow and event response
  - Multitasking
  - Concise ISRs thus deterministic
  - Better communication
  - Better resource management
  - Easier to develop applications

A case in point

- GPS based Speed Camera Alarm and Moving Map
  - Sounds alarm when approaching a speed camera
  - Display's vehicle position on LCD
  - Also logs driver's position information
  - Hardware: GPS, user switches, speaker, LCD, flash memory
- Tasks:
  - Dec: Decode GPS sentence to find current vehicle position.
  - Check: Check to see if approaching any speed camera locations. Takes longer as the number of cameras increases.
  - Rec: Record position to flash memory. Takes a long time if erasing a block.
  - Sw: Read user input switches.
  - LCD: Update LCD with map.
- How to implement in a super loop?
  - Run tasks in the same order every time?
  - Allow preemption?



Super-loop



- Simple but…
  - Always run the same schedule, regardless of changing conditions and relative importance of tasks.
  - All tasks run at the same rate. Changing rates requires adding extra calls to the function.
  - Maximum delay is the sum of all task run times. Polling/execution rate is equal to 1/maximum delay.
- What if we receive GPS position right after Rec starts running?
- Delays
  - Have to wait for Rec, Sw, LCD before we start decoding position with Dec.
  - Have to wait for Rec, Sw, LCD, Dec, Check before we know if we are approaching a speed camera!

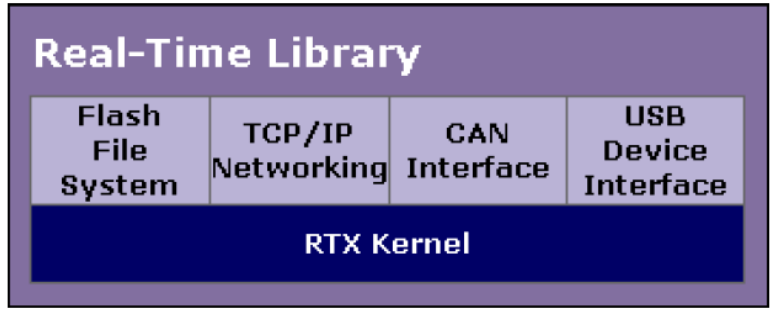
```
while (1){
        Dec();
        Check();
        Rec();
        Sw();
        LCD();
}
```

RTX

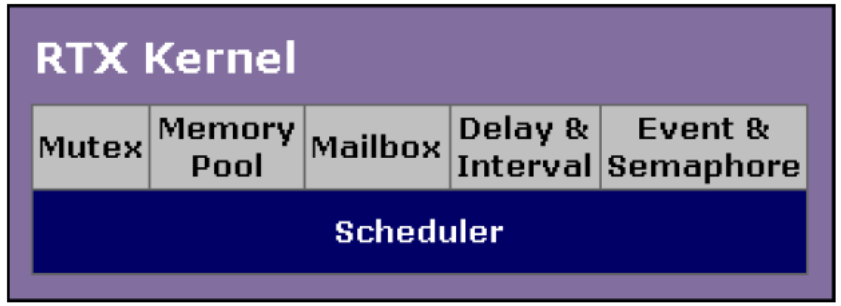
- Royalty-free, deterministic, open source RTOS
- High-Speed real-time operation with low interrupt latency
- Flexible Scheduling: round-robin, pre-emptive, and collaborative
- Small footprint for resource constrained systems
- Compatible with Arm cores (from Arm7, Arm9 to Cortex-M processors) and software tools (Keil MDK-Arm)
- Support for multithreading and thread-safe operation
- Kernel aware debug support in Keil MDK-Arm
- Dialog-based setup using µVision Configuration Wizard

RTX Structure

- Keil Real-Time Library (RTL)
  - RTX Kernel
  - Flash file system
  - Networking
  - CAN interface
  - USB device interface



- RTX Kernel
  - Scheduler is the core of the RTX kernel
  - Supports for mutex, memory pool, mailbox, timing functions, events and semaphores



arm

RTX Files

- Add RTX to your project
  - Set RTX Kernel as OS
  - Include <**RTL.h**>
  - Add **RTX_Conf_CM.c** file (RTX_Conf_CM.c file will also include **RTX_lib.c)**

- RTL.h – defines the basic types and kernel APIs
- RTX_Conf_CM.c – configures the RTX and support for the configuration wizard
- RTX_lib.c – the kernel system configuration, providing an in-code control of the kernel

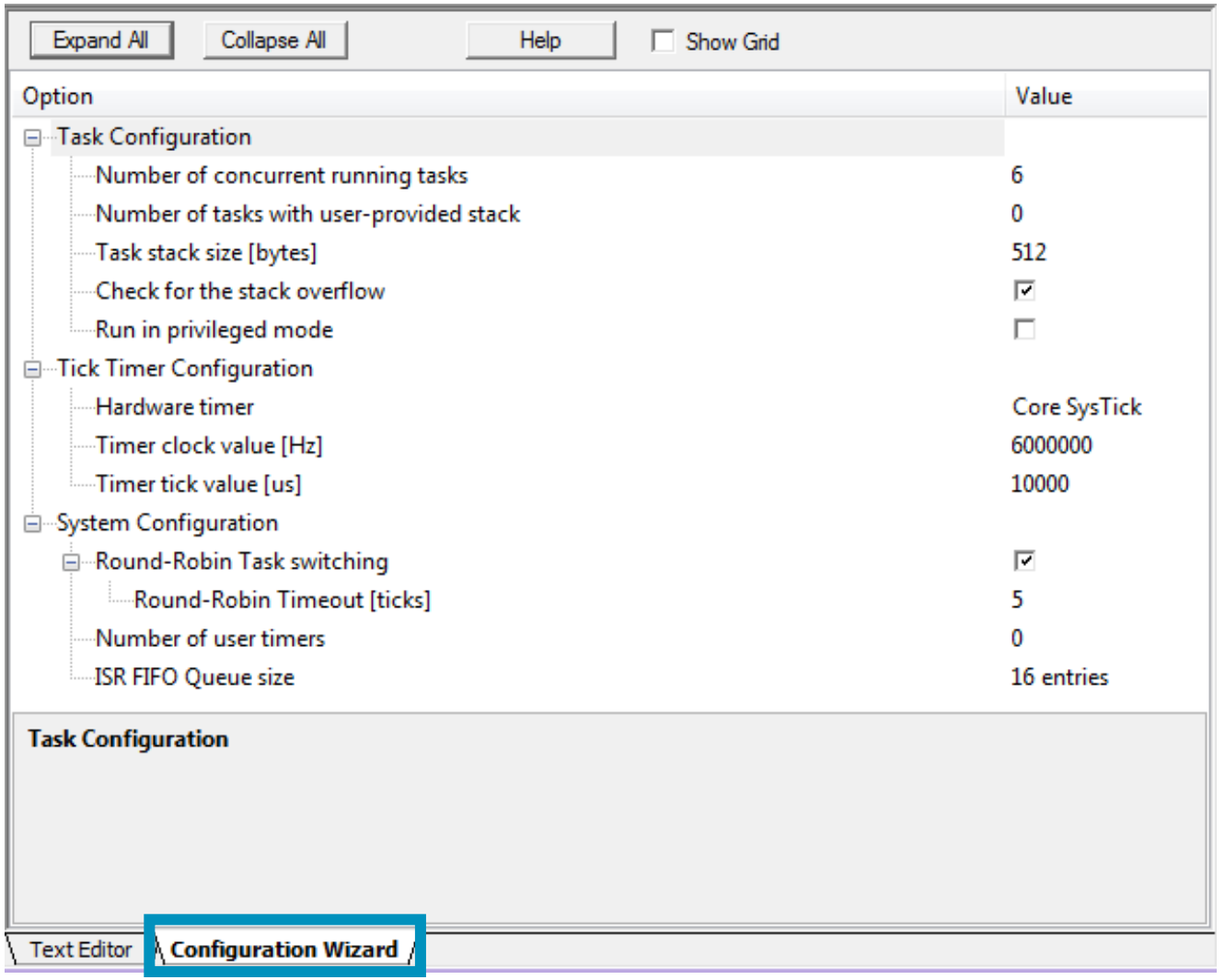
- Source code of the RTX kernel is precompiled and linked by Keil MDK-Arm

RTL.h

- Type defines
- Supports Cortex-M architecture, with the following features:
  - Task management
  - Event flag management
  - Semaphore management
  - Mailbox management
  - Mutex management
  - Time management
  - User Timer management
  - System functions
  - Fixed memory block management functions

RTX_Conf_CM.c

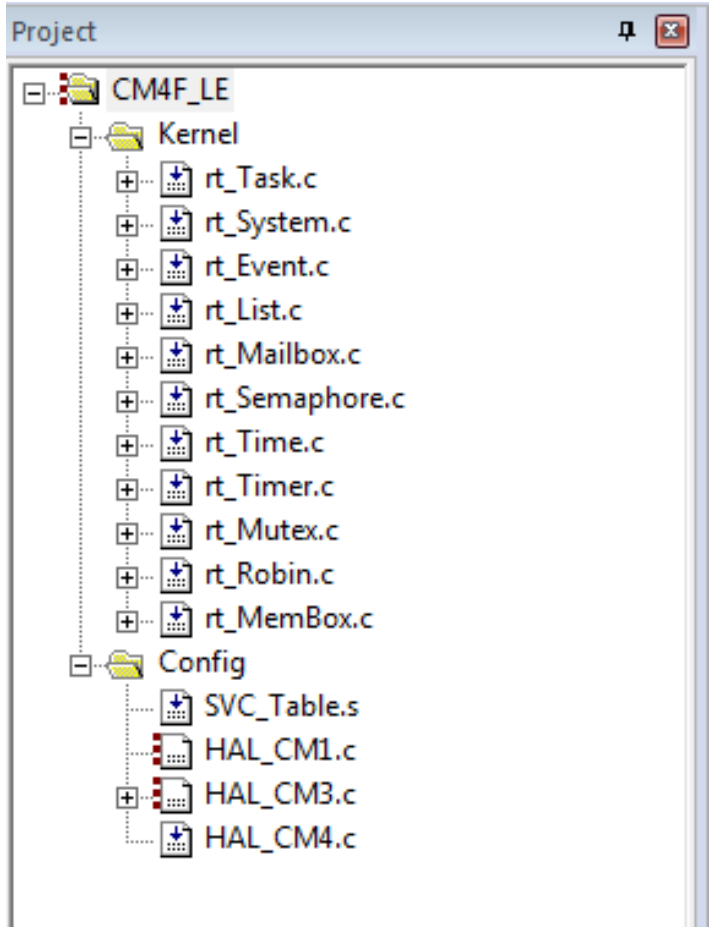
- “<h> <o> <i> <q>” are the annotations for Keil configuration wizard
- Configuration wizard provides an easy-to-use GUI (see picture on the right)
- Can also change the configuration by directly modifying the file



More about the Keil configuration wizard: http://www.keil.com/support/man/docs/uv4/uv4_ut_configwizard.htm

Source code of RTX

- By default C:\Keil_v5\Arm\RL\RTX\SRC\CM (Keil uVision 5)
- Or through the project file RTX_Lib_CM located in C:\Keil_v5\Arm\RL\RTX

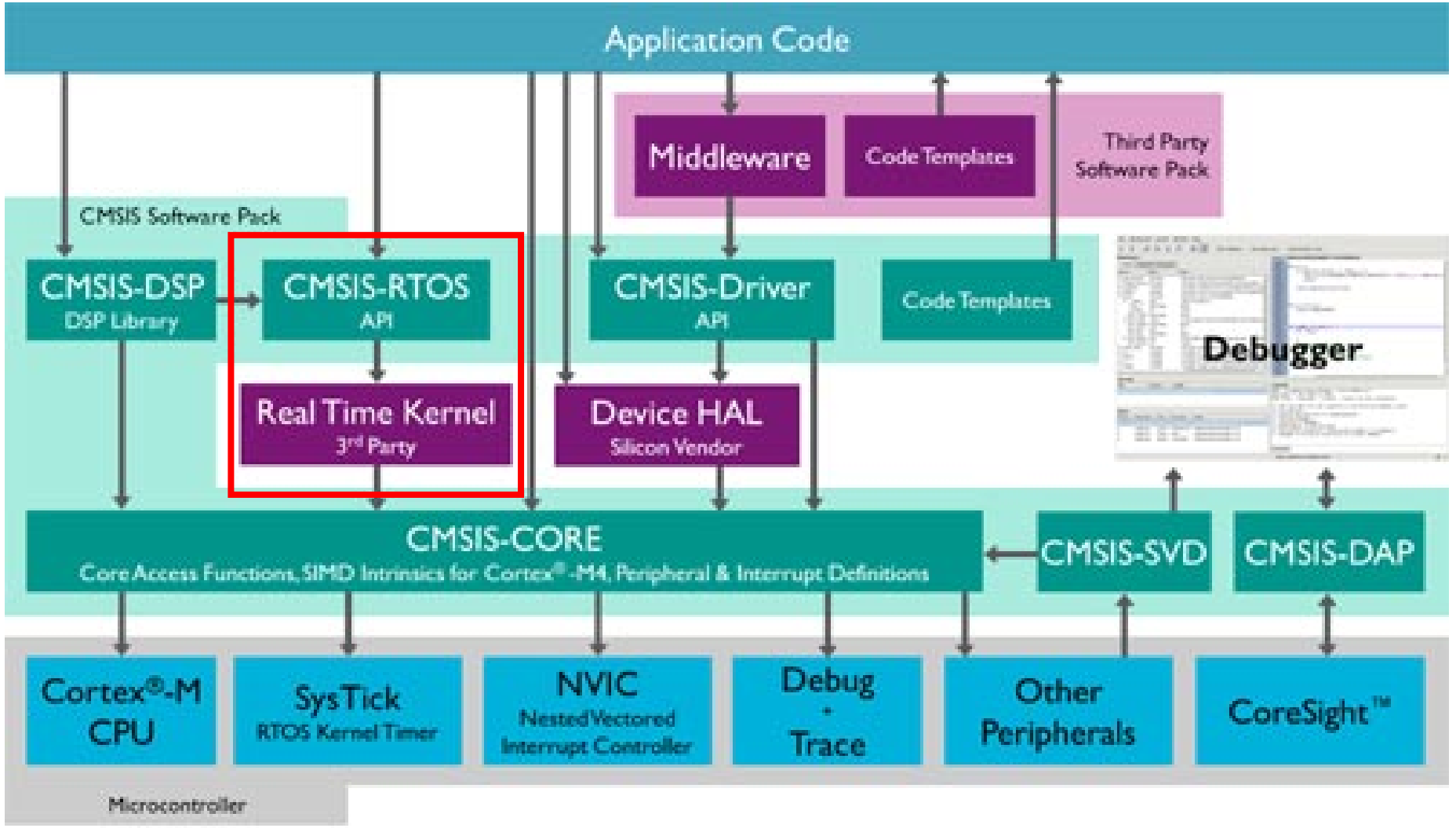


arm

CMSIS and CMSIS-RTOS

- Arm Cortex Microcontroller Software Interface Standard (CMSIS)
  - Vendor-independent hardware abstraction layer
  - Used for Cortex-M processor series
  - Specifies debugger interfaces
- The CMSIS-RTOS API
  - Generic RTOS interface for Cortex-M processor-based devices
  - Standardized API for software components that require RTOS functionality, gives serious benefits to the users and the software industry

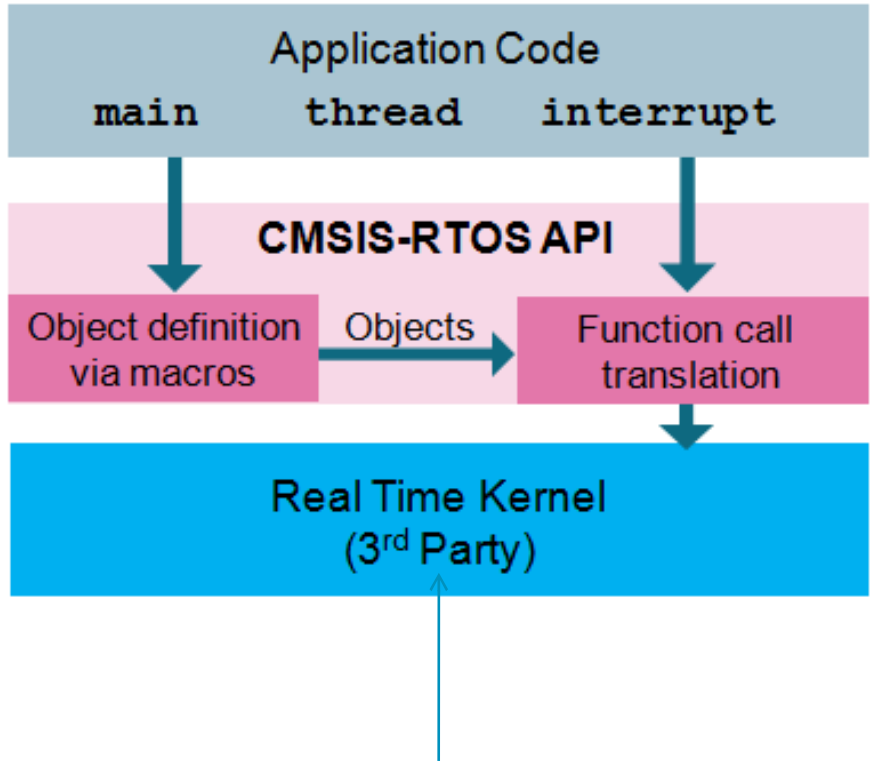
CMSIS and CMSIS-RTOS



arm

CMSIS-RTOS and RTX

- CMSIS-RTOS API – an abstracted layer on top of RTX kernel or other 3rd party RTOS kernels
- Application programs can be easily ported to other 3rd party RTOS kernels (such as FreeRTOS)
- In this set of teaching materials, we will focus on the RTX kernel rather than the CMSIS-RTOS API
  - Same concept
  - More low-level details



arm

To setup RTX

- In the main function

os_sys_init(first_task);

- That's it!