FreeRTOS Preface

Norman McEntire
norman.mcentire@gmail.com

Textbook Reference

- Mastering the FreeRTOS Real Time Kernel by Richard Barry
  - Preface

Outline

- About FreeRTOS
- Value Proposition
- A Note About Terminology
- Why Use a Real-Time Kernel?
- FreeRTOS Features

About FreeRTOS

- FreeRTOS is aimed at embedded real-time microcomputer-based applications
- Current Version is 10.1.1
- These applications include two types of real-time
    - Soft Real-Time Requirements
    - Hard Real-Time Requirements

Soft Real-Time Requirements

- Soft Real-Time Requirements state a time deadline
  - But breaching the dead-line does not render the system useless
    - Example: Keystroke responding too slowly

Hard Real-Time Requirements

- Hard Real-Time Requirements state a time deadline
  - Breaching the dead-line renders the system useless
    - Example: Airbag deploys too late

FreeRTOS
Soft and Hard Real-Time

- FreeRTOS supports Soft and Hard Real-Time
- Organize applications as a collection of independent tasks
    - Higher Priority Tasks can be dedicated to Hard Real-Time
    - Lower Priority Tasks can be dedicated to Soft Real-Time

Value Proposition for FreeRTOS

- Professionally developed
- Strict Quality Control
- Free/Open Source
- Written in portable C
  - Minimal kernel is only three C source files!
- Small memory footprint, low overhead, fast execution

FreeRTOS Tasks

- FreeRTOS uses the concept of Tasks
  - A thread of execution is a Task
  - In FreeRTOS, you organize your solution into Tasks
    - Much more on creating Tasks later in this course

Key FreeRTOS Features

- Open Source with MIT License
- Static and Dynamic Memory Allocation
- Tasks (with priorities)
- Software Timers (one shot and periodic)
- Queues
- Synchronization
  - Semaphores (Binary and Counting)
  - Mutexes
- Tick-less Mode for low-power

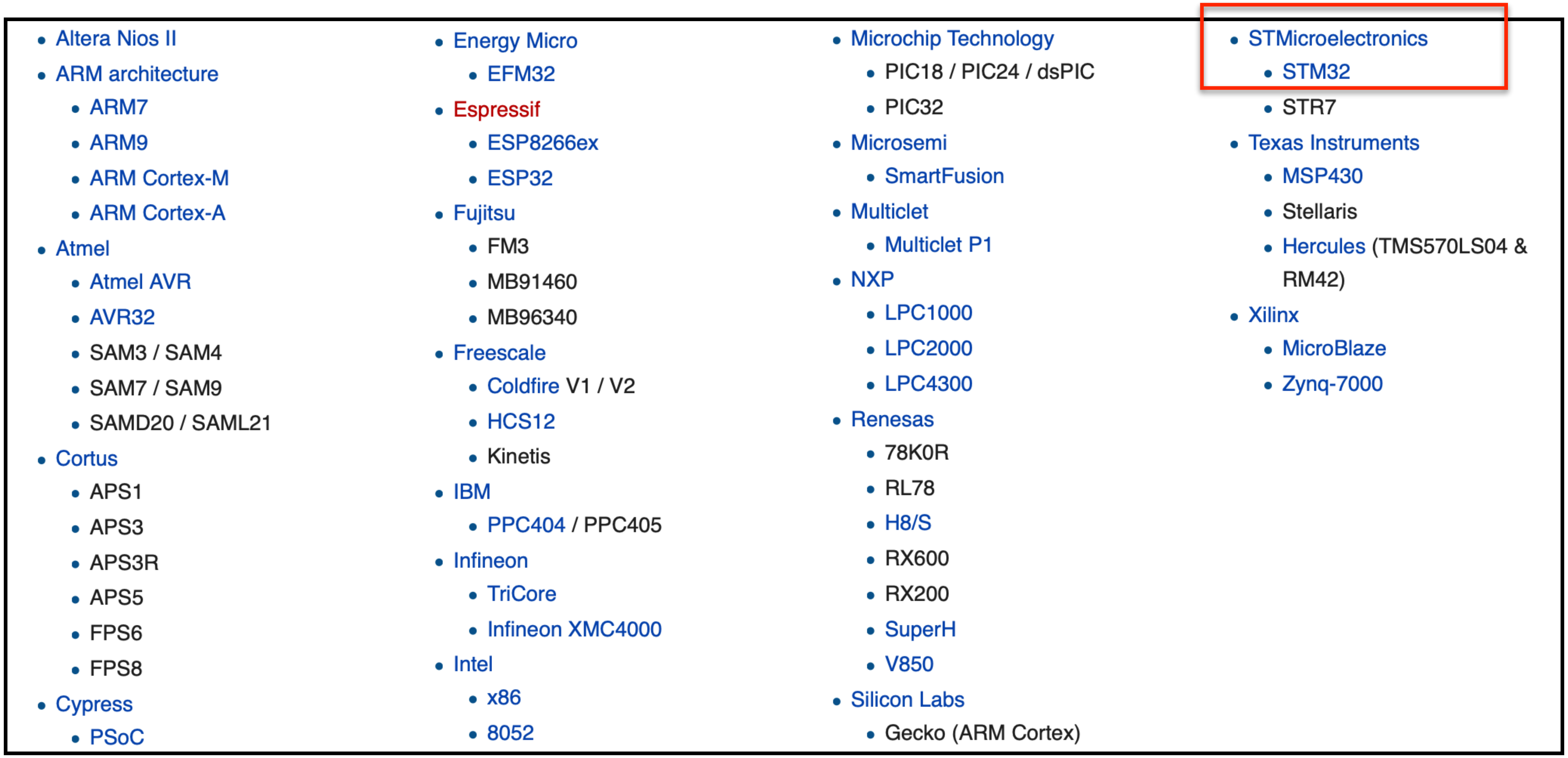
FreeRTOS Licenses

- The FreeRTOS License
    - Can be used in commercial applications
    - Freely available to everybody
    - Users retain ownership in their IP
        - IP = Intellectual Property
- See more info on next slide

FreeRTOS Versions

- FreeRTOS - free license - We use this in the course
- OpenRTOS - commercial license
- SafeRTOS - For Safety-critical implementations
- Amazon FreeRTOS
    - Adds libraries for IoT Support for AWS
        - AWS = Amazon Web Services

Supported Architectures (30+ Architectures!)



- Altera Nios II
- ARM architecture
  - ARM7
  - ARM9
  - ARM Cortex-M
  - ARM Cortex-A
- Atmel
  - Atmel AVR
  - AVR32
  - SAM3 / SAM4
  - SAM7 / SAM9
  - SAMD20 / SAML21
- Cortus
  - APS1
  - APS3
  - APS3R
  - APS5
  - FPS6
  - FPS8
- Cypress
  - PSoC
- Energy Micro
  - EFM32
- Espressif
  - ESP8266ex
  - ESP32
- Fujitsu
  - FM3
  - MB91460
  - MB96340
- Freescale
  - Coldfire V1 / V2
  - HCS12
  - Kinetis
- IBM
  - PPC404 / PPC405
- Infineon
  - TriCore
  - Infineon XMC4000
- Intel
  - x86
  - 8052
- Microchip Technology
  - PIC18 / PIC24 / dsPIC
  - PIC32
- Microsemi
  - SmartFusion
- Multiclet
  - Multiclet P1
- NXP
  - LPC1000
  - LPC2000
  - LPC4300
- Renesas
  - 78K0R
  - RL78
  - H8/S
  - RX600
  - RX200
  - SuperH
  - V850
- Silicon Labs
  - Gecko (ARM Cortex)
- STMicroelectronics
  - STM32
  - STR7
- Texas Instruments
  - MSP430
  - Stellaris
  - Hercules (TMS570LS04 & RM42)
- Xilinx
  - MicroBlaze
  - Zynq-7000

Why use RTOS?

- Ability to create multiple tasks with different priorities
- Abstracting away timing info - let RTOS handle timing
- Maintainability / Extensibility - RTOS handles details
- Modularity - Each task separate
- Team Development - well-defined modules/ interfaces
- Easier testing - well-defined modules / interfaces
- Code Reuse - multiple projects, controllers, etc.
- Improved Efficiency - software is fully event driven
- Idle time - scheduler handles idle time
- Power Management - can spend more time in low-power mode
- Flexible interrupt scheduling - can use deferred processing on RTOS daemon task

FreeRTOS on Windows

- FreeRTOS Port on Windows
- Useful for learning FreeRTOS APIs
    - But not really real-time on Windows
- Windows version built with free Express Edition of Visual Studio

FreeRTOS Distribution

The FreeRTOS Distribution

- FreeRTOS Distributed as single ZIP file
    - Contains all official FreeRTOS Ports
    - Contains large number of pre-configured demos

FreeRTOS Port

- FreeRTOS is build for…
  - …20 different compilers….
  - …30 different architectures…
- Each compiler/architecture is a different Port
  - …./compiler/architecture/….

Building FreeRTOS

- FreeRTOS can be thought of as a library that provides multi-tasking capabilities
- Supplied as a set of C source files
    - Some files common to all ports
    - Some file unique to a given port
- Demo applications are included - can be built “right out the box”

FreeRTOSConfig.h

- FreeRTOS is configured with FreeRTOSConfig.h file
    - Example: configUSE_PREEMPTION
- Every demo application has a separate FreeRTOSConfig.h
- NOTE: Find an existing FreeRTOSConfig.h file and edit it to match your needs

FreeRTOS Directory Organization

- **FreeRTOS** - base RTOS code
  - Source - RTOS source files
  - Demo - Pre-configured demos
- **FreeRTOS-Plus** - Ecosystem Code
  - Source
  - Demo

FreeRTOS Files Common to All Projects

- Only Two files required
  - FreeRTOS/Source/**tasks.c**
  - FreeRTOS/Source/**list.c**
- Other optional files include the following
  - FreeRTOS/Source/**timers.c**
  - FreeRTOS/Source/**queue.c**
  - FreeRTOS/Source/**event_groups.c**

FreeRTOS Files Specific to a Port

- FreeRTOS/Source/**portable**/compiler/architecture
  - Example
    - FreeRTOS/Source/portable/**GCC**/**ARM_CM4F**
- Memory Management is part of the porting layer
  - Example
    - FreeRTOS/Source/portable/**MemMang**/heap_4.c

Include Directories

- FreeRTOS requires three directories to be included in the compiler's include path
    - FreeRTOS/Source/include
    - FreeRTOS/Source/portable/[compiler]/[arch]
    - FreeRTOSConfig.h

Header Files

- Header Files Must follow in this order
    - **FreeRTOS.h**
    - task.h
    - timers.h
    - queue.h
    - semphr.h
    - event_groups.h

main.c Example

-
  ```
  int main(void) {
      …
      prvSetupHardware();
      …
      vTaskStartScheduler(); //v = void
      …
      // Should never get here
      for(;;);
      return 0;
  }
  ```

Data Types and Coding Style

Data Types - Part 1

- Each port of FreeRTOS has a unique portmacro.h
    - Defines two port specific ports
        - **BaseType_t**
            - The most efficient data type for the given architecture
            - Typically used for return types
            - Also used for pdTRUE and pdFALSE Booleans
            - Typically 32-bits
        - **TickType_t**
            - Used to hold tick values
            - Typically 32-bits

Data Types - Part 2

- Related to char
  - Some compilers make unqualified char as unsigned
    - Other compilers make unqualified char as signed
- For this reason **FreeRTOS always qualifies a char**
  - unsigned char
  - signed char

Data Types - Part 3

- Plain “int” types are never used
- Instead use these
    - uint8_t, uint16_t, uint32_t
    - int8_t, int16_t, int32_t

Variable Names

- Variable names are prefixed with their type
  - c for char / int8
  - s for short / int16
  - l for long / int32
  - x for BaseType_t and any other non-standard types
  - u for unsigned (e.g. ul for uint32_t)
  - p for pointer (e.g. pc for *int8_t)

Function Names

- Function names are prefixed with…
    - The return type
    - AND
    - The file they are in (e.g. task.c, queue.c)
- Examples
    - vTaskPriroitySet() - returns void and defined in task.c
    - xQueueReceive() - returns BaseType_t and defined in queue.c
    - pvTimerGetTimerID() - returns pointer to void and defined in timers.c
- File scope (private) functions are prefixed with prv
    - prv = private

Formatting

- One tab is always equal to 4 spaces

Macro Names

- Most macros written in UPPERCASE and prefixed with lowercase that indicate where macro defined
  - portMAX_DELAY() - portmacro.h
  - taskENTER_CRITICAL() - task.h
  - pdTRUE - projectdefs.h
  - configUSE_PREEMPTION - FreeRTOSConfig.h
  - errQUEUE_FULL - projdefs.h

Macros Used Throughout Code

- pdTRUE 1
- pdFALSE 0
- pdPASS 1
- pdFAIL 0

Why Excessive Casting?

- FreeRTOS compiled on many compilers
    - All of which generate warnings differently
- Different compilers want casting to be used in different ways
    - Hence FreeRTOS uses more casting to satisfy a wide range of compilers

Summary

- FreeRTOS Introduction
    - FreeRTOS Features and Benefits
    - FreeRTOS Distribution as single ZIP file
    - FreeRTOS Data Types and Naming