ECEN 5623

Intro to FreeRTOS

Be Boulder.

University of Colorado **Boulder**

FreeRTOS
What is it?

FreeRTOS is a popular[1] real-time operating system kernel[2][3] for embedded devices, that has been ported to 92 processor families.

It was distributed under the GPL with an additional restriction and optional exception. The restriction forbids benchmarking while the exception permits users' proprietary code to remain closed source while maintaining the kernel itself as open source, thereby facilitating the use of FreeRTOS in proprietary applications.

FreeRTOS is now distributed under the MIT license since managed by Amazon Web Services (AWS).

FreeRTOS

What is it?

OpenRTOS is a commercially licensed version of the FreeRTOS kernel that includes indemnification and dedicated support. FreeRTOS and OpenRTOS share the same code base. OpenRTOS is provided under license from AWS by WITTENSTEIN high integrity systems - an AWS strategic partner.

SafeRTOS is a derivative version of the FreeRTOS kernel that has been analyzed, documented and tested to meet the stringent requirements of industrial (IEC 61508 SIL 3), medical (IEC 62304 and FDA 510(K)) automotive (ISO 26262) and other international safety standards. SafeRTOS includes independently audited safety life cycle documentation artifacts. SafeRTOS is provided by WITTENSTEIN high integrity systems - an AWS strategic partner.

FreeRTOS What is it?

FreeRTOS is designed to be small and simple. The kernel itself consists of only three C files.

To make the code readable, easy to port, and maintainable, it is written mostly in C, but there are a few assembly functions.

FreeRTOS provides methods for multiple threads or tasks, mutexes, semaphores and software timers. A tick-less mode is provided for low power applications. Thread priorities are supported. FreeRTOS applications can be completely statically allocated. Alternatively RTOS objects can be dynamically allocated with five schemes of memory allocation provided:

- allocate only;
- allocate and free with a very simple, fast, algorithm;
- a more complex but fast allocate and free algorithm with memory coalescence;
- an alternative to the more complex scheme that includes memory coalescence that allows a heap to be broken across multiple memory areas.
- and C library allocate and free with some mutual exclusion protection.

https://en.wikipedia.org/wiki/FreeRTOS

What is Memory Coalescence

A What happens to your brain after an RTES exam — 0%

B Loss of brain cells in old people, notably candidates for President — 0%

C An algorithm that groups together global memory to impove memory bandwidth — 0%

D A program to reduce the number of memory blocks accessed by multiple threads — 0%

C and D — 0%

FreeRTOS
What is it?

FreeRTOS has the following standard features:

- Pre-emptive or co-operative operation
- Very flexible task priority assignment
- Flexible, fast and light weight task notification mechanism
- Queues
- Binary semaphores
- Counting semaphores
- Mutexes
- Recursive Mutexes
- Software timers
- Event groups
- Tick hook functions
- Idle hook functions
- Stack overflow checking
- Trace recording
- Task run-time statistics gathering
- Optional commercial licensing and support
- Full interrupt nesting model (for some architectures)
- A tick-less capability for extreme low power applications
- Software managed interrupt stack when appropriate (this can help save RAM)

https://en.wikipedia.org/wiki/FreeRTOS

FreeRTOS
Why use it?

With millions of deployments in all market sectors, blue chip companies trust FreeRTOS because it is professionally developed, strictly quality controlled, robust, supported, free to use in commercial products without a requirement to expose proprietary source code, and has no IP infringement risk.

FreeRTOS is a risk free choice, providing the best of all worlds:
FreeRTOS is truly free and supported, even when used in commercial applications.

- Is known to be reliable. Confidence is assured by the activities undertaken by the SafeRTOS sister project.
- Is feature rich and still undergoing continuous active development.
- Has a minimal ROM, RAM and processing overhead. Typically an RTOS kernel binary image will be in the region of 6K to 12K bytes.
- Is well established with a large and ever growing user base.
- Is very scalable, simple and easy to use.

https://en.wikipedia.org/wiki/FreeRTOS

FreeRTOS

Supported architectures

- Altera Nios II & ARM Cortex A9
- ARM architecture
  - ARM7, ARM9
  - ARM Cortex-M
  - ARM Cortex-A
- ~~Atmel~~ Microchip
  - Atmel AVR, AVR32, SAMV7
  - SAM3 / SAM4, SAM7 /SAM9
  - SAMD20 / SAML21/ SAMA5
- Cortus
  - APS1, APS3, APS3R, APS5
  - FPS6, FPS8
- Cypress
  - PSoC5 (ARM M3)
- ~~Fujitsu~~ Spansion
  - FM3, MB91460
  - MB96340
- ~~Freescale~~ NXP
  - Coldfire V1 / V2
  - HCS12, KinetisM4
- IBM
  - PPC404 / PPC405
- Infineon
  - TriCore, XMC4000
- Intel
  - x86 (IA32 only), 8052
- PIC microcontroller
  - PIC18 / PIC24 / dsPIC
  - PIC32
- Microsemi
  - SmartFusion2, RISC-V
- NXP
  - LPC1x00, 2x00, 4300, RISC-V
- Renesas
  - 78K0R, RL78, H8/S
  - RX600, RX200, SuperH
  - V850, RZ/A1 (ARM A9)
- SiFive RISC-V RV32

FreeRTOS

Supported architectures

STMicroelectronics

- STM32 (ARM M0, M3, M4F, M7), STR7

Silicon Labs

- EFM32 Gecko (ARM M3, M4F
- 8051

Texas Instruments

- MSP430, MSP432
- Stellaris, RM48
- Hercules (TMS570LS04 & RM42)
- TIVA (ARM Cortex-M4)

Supported architectures

Xilinx

- MicroBlaze
- Zynq-7000, Ultrascale+ MPSoC
- PPC405, PPC440

Espressif

- ESP8266ex

Contributed Ports

Lattice MICO32, Analog Devices Blackfin, Zilog eZ80, etc.

FreeRTOS – Installation Choices

1. DE1-SoC target
2. TIVA board target
3. Visual Studio emulator
4. Eclipse Emulator

FreeRTOS

FreeRTOS's code breaks down into three main areas: **tasks, communication, and hardware interfacing**.

Tasks: Almost half of FreeRTOS's core code deals with the central concern in many operating systems: tasks. A task is a user-defined C function with a given priority. tasks.c and task.h do all the heavy lifting for creating, scheduling, and maintaining tasks.

Communication: Tasks are good, but tasks that can communicate with each other are even better! queue.c and queue.h handle FreeRTOS communication. Tasks and interrupts use queues to send data to each other and to signal the use of critical resources using semaphores and mutexes.

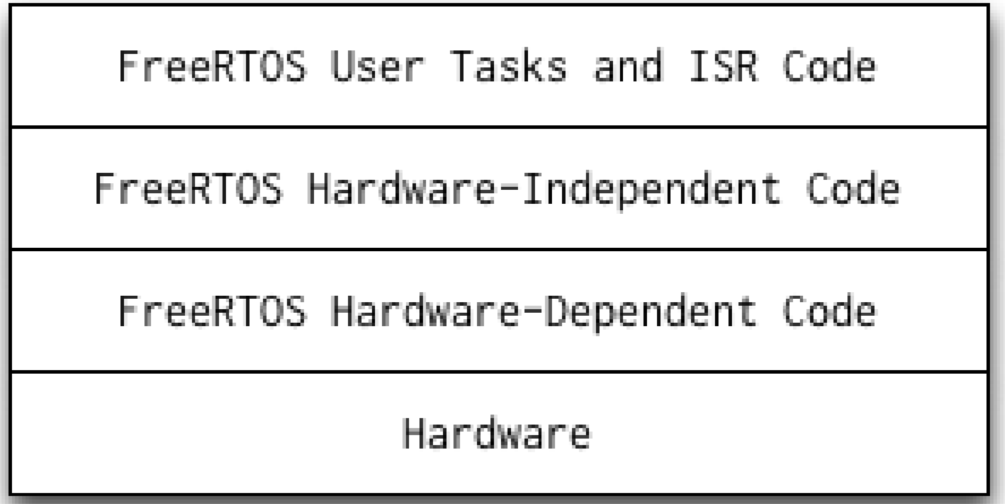
The Hardware Whisperer: The approximately 9000 lines of code that make up the base of FreeRTOS are hardware-independent; the same code runs whether FreeRTOS is running on the humble 8051 or the newest, shiniest ARM core. About 6% of FreeRTOS's core code acts a shim between the hardware-independent FreeRTOS core and the hardware-dependent code.

From http://www.aosabook.org/en/freertos.html by Christopher Svec

FreeRTOS

FreeRTOS's code breaks down into three main areas: **tasks, communication, and hardware interfacing**.

The hardware-independent FreeRTOS layer sits on top of a hardware-dependent layer. This hardware-dependent layer knows how to talk to whatever chip architecture you choose that is supported.



FreeRTOS software layers

From http://www.aosabook.org/en/freertos.html
by Christopher Svec

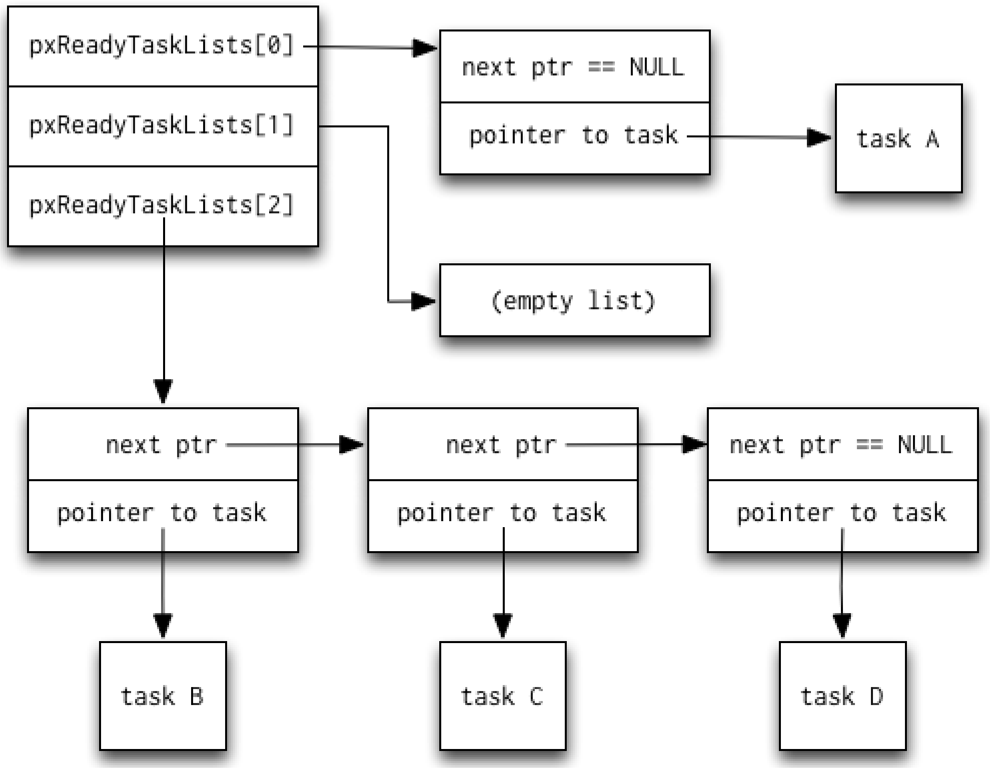
FreeRTOS

The heartbeat of a FreeRTOS system is called the system tick. FreeRTOS configures the system to generate a periodic tick interrupt. The user can configure the tick interrupt frequency, which is typically in the millisecond range. Every time the tick interrupt fires, the vTaskSwitchContext() function is called. vTaskSwitchContext() selects the highest-priority ready task and puts it in the pxCurrentTCB variable:

```
/* Find the highest-priority queue that contains ready tasks. */
while( listLIST_IS_EMPTY( &( pxReadyTasksLists[ uxTopReadyPriority ] ) ) )
{
    configASSERT( uxTopReadyPriority );
    --uxTopReadyPriority;
}

/* listGET_OWNER_OF_NEXT_ENTRY walks through the list, so the tasks of the same
priority get an equal share of the processor time. */
listGET_OWNER_OF_NEXT_ENTRY( pxCurrentTCB, &( pxReadyTasksLists[
uxTopReadyPriority ] ) );
```

FreeRTOS



From http://www.aosabook.org/en/freertos.html
by Christopher Svec

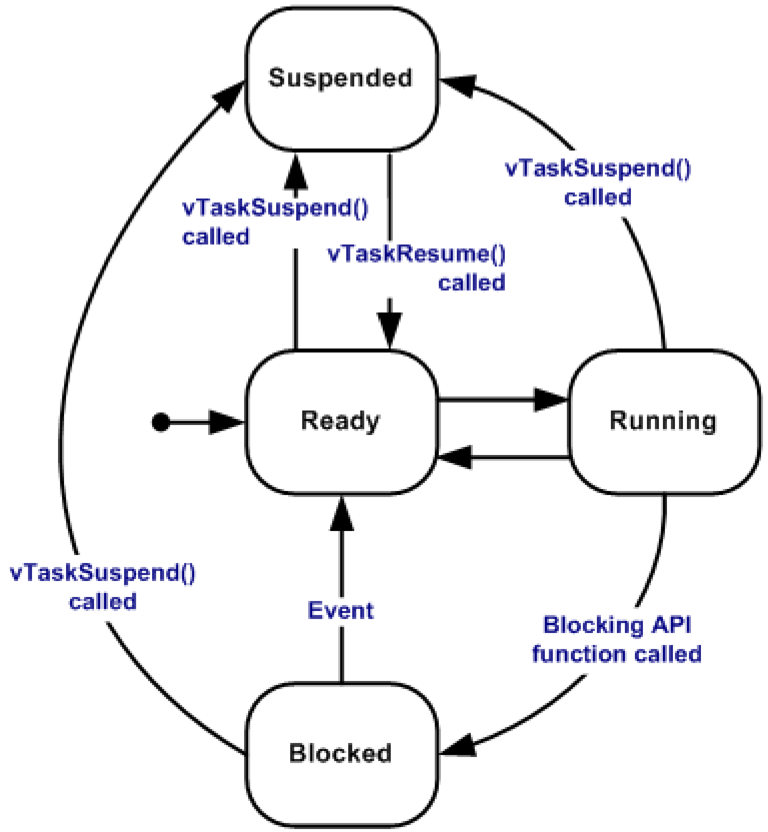
FreeRTOS

A task can exist in one of the following states:

•Running
•Ready
•Blocked
•Suspended

A task should have the following structure:

```
void vATaskFunction( void *pvParameters )
{
        for( ;; )
   {
        -- Task application code here. --
   }
}
```



How to Create a FreeRTOS Project

Adapting One of the Supplied Demo Projects

Every FreeRTOS port comes with at least one pre-configured demo application that should build with no errors or warnings. It is recommended that new projects are created by adapting one of these existing projects; this will allow the project to have the correct files included, the correct interrupt handlers installed, and the correct compiler options set.

To start a new application from an existing demo project:

1. Open the supplied demo project and ensure that it builds and executes as expected.
2. Remove the source files that define the demo tasks. Any file that is located within the Demo/Common directory can be removed from the project.
3. Delete all the function calls within main(), except prvSetupHardware() and vTaskStartScheduler(), as shown in Listing 1.
4. Check the project still builds.

Following these steps will create a project that includes the correct FreeRTOS source files, but does not define any functionality.

How to Create a FreeRTOS Project

```
int main( void )
{
/* Perform any hardware setup necessary. */
prvSetupHardware();
/* --- APPLICATION TASKS CAN BE CREATED HERE --- */
/* Start the created tasks running. */
vTaskStartScheduler();
/* Execution will only reach here if there was insufficient heap to
start the scheduler. */
for( ;; );
return 0;
}
```

Listing 1. The template for a new main() function

FreeRTOS Identifier Conventions

Variable Names

Variables are prefixed with their type: ‘c’ for char, ‘s’ for int16_t (short), ‘l’ int32_t (long), and ‘x’ for BaseType_t and any other non-standard types (structures, task handles, queue handles, etc.).

If a variable is unsigned, it is also prefixed with a ‘u’. If a variable is a pointer, it is also prefixed with a ‘p’. For example, a variable of type uint8_t will be prefixed with ‘uc’, and a variable of type pointer to char will be prefixed with ‘pc’.

Function Names

Functions are prefixed with both the type they return, and the file they are defined within. For example:

☐ v**Task**PrioritySet() returns a void and is defined within **task**.c.

☐ x**Queue**Receive() returns a variable of type BaseType_t and is defined within **queue**.c.

☐ pv**Timer**GetTimerID() returns a pointer to void and is defined within **timers**.c.

File scope (private) functions are prefixed with ‘prv’.

FreeRTOS – Installation Choices

1. DE1-SoC target
2. TIVA board target
3. Visual Studio emulator
4. Eclipse Emulator

FreeRTOS –

1. DE1-SoC target

https://www.freertos.org/RTOS_Altera_SoC_ARM_Cortex-A9.html

CU Sites | Tools | OnlineCourses | CU Course pages | Boards | Organization | Family | India | ARM | Technical | HDL | Software | FPGA

The FreeRTOS kernel is now an MIT licensed AWS open source project, and these pages are being updated acc

Quality RTOS & Embedded Software

About | Contact | Support | FAQ | Download

Quick Start | Supported MCUs | PDF Books | Trace Tools | Ecosystem | Email List

- Home
- MIT License
- FreeRTOS Books and Manuals
- FreeRTOS
- FreeRTOS Interactive!

Quick Start Guide

Download Source

FreeRTOS+ Ecosystem

FreeRTOS+TCP: Thread safe TCP/IP stack

SafeRTOS: TUV certified RTOS

OpenRTOS: Commercial Licensed RTOS

Fail Safe File System: Ensures data integrity

FreeRTOS BSPs: 3rd party driver packages

UDP/IP: Thread aware UDP stack

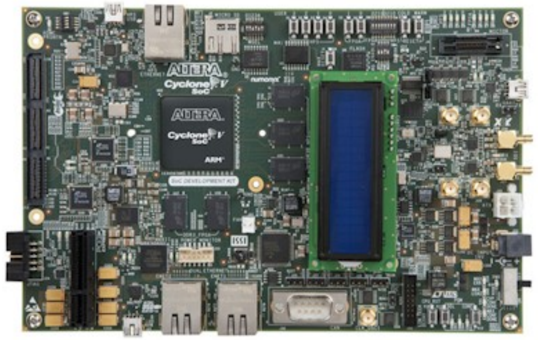
Trace & Visualisation: Tracealyzer for FreeRTOS

CLI: Command line interface

Altera Cyclone V SoC RTOS Demo

(ARM Cortex-A9)

[RTOS Ports]



Introduction

This page documents a FreeRTOS demo application for a Cortex-A9 core in the Altera Cyclone V SoC Hard Processing System (HPS).

The project builds using the free Altera edition of the ARM DS-5 Eclipse based IDE and the GCC compiler, both of which come as part of the Altera Embedded Development Suite (EDS). Note only the DS-5 and compiler components of the EDS are used - it is not necessary to install any FPGA tools to build or use this RTOS demo.

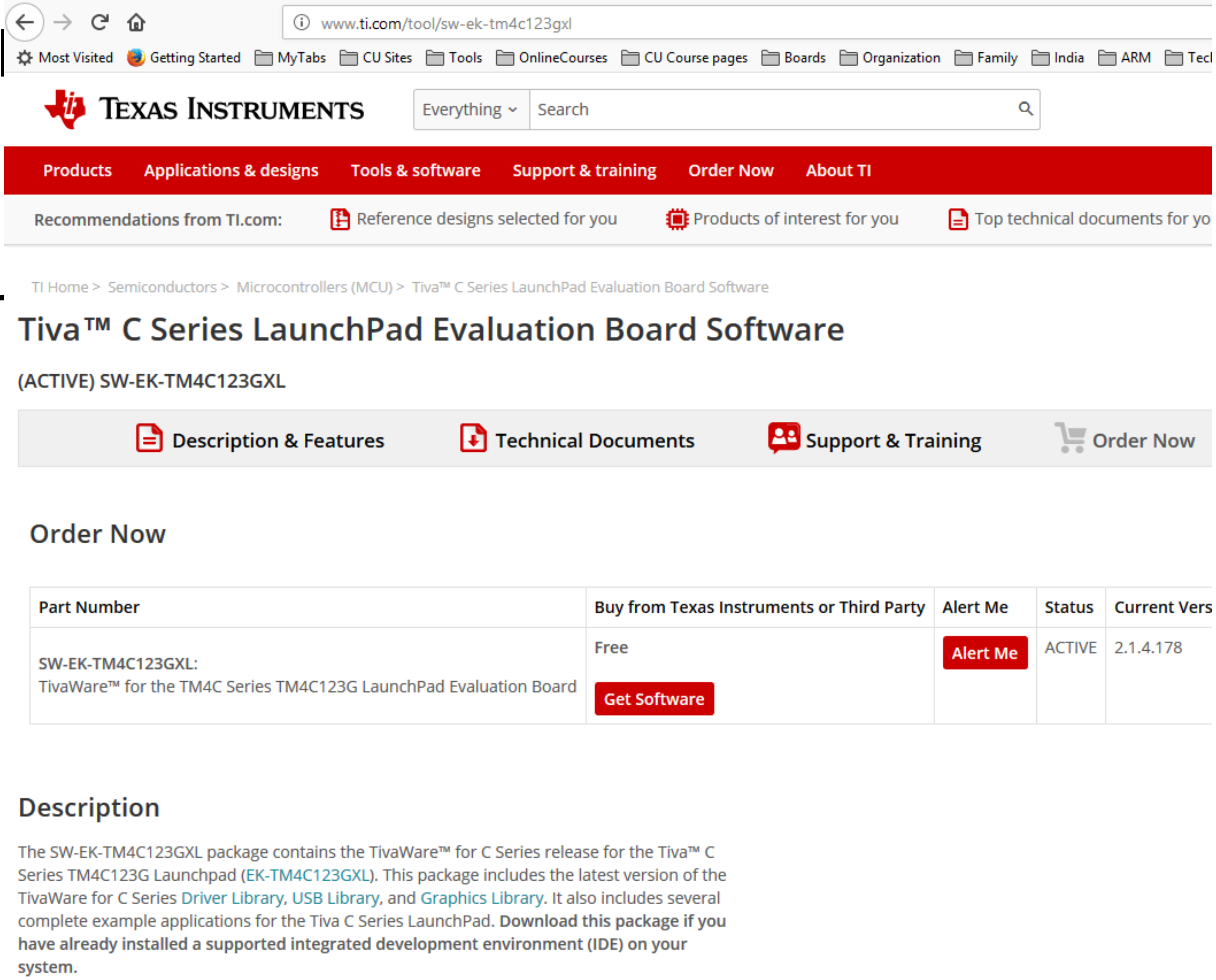
The project is pre-configured to execute on the Cyclone V SoC Development Kit hardware.

FreeRTOS – I

1. TIVA board target

Get Code Composer

TivaWare from TI. FreeRTOS demo is in Tivaware.



FreeRTOS – Inst

1. Visual Studio emulator