Application Note

Developing Common Applications With FreeRTOS on TM4C MCUs

Ralph Jacobi and Charles Tsai

ABSTRACT

FreeRTOS is a real-time operating system for embedded systems. It has been widely ported to many architecture platforms due to its compact size and being distributed under free open source licensing. This application report is a continuation of the application report Developing Basic Applications With FreeRTOS on TM4C MCU with additional examples for common peripherals to demonstrate how to use the FreeRTOS kernel on the Texas Instruments TM4C family of Arm® Cortex®-M4F microcontrollers.

The source code for all example projects discussed in this application report can be downloaded from the following link http://www.ti.com/lit/zip/spma086.

Table of Contents

Trademarks

1 Introduction

The microcontroller market continues to see a steady increase in the utilization of embedded real-time operating systems (RTOS) to manage applications. The increased focus and need for RTOS solutions has led to a number of open source RTOS offerings including the widely used FreeRTOS. To address the growing need for FreeRTOS support on the TM4C Arm Cortex-M4F microcontroller series, Developing Basic Applications With FreeRTOS on TM4C MCU was released to provide a set of fundamental application examples. This application report builds off that foundation and expands the application examples to cover many commonly used TM4C peripherals.

Included with this application report are example projects for both the TM4C123x and the TM4C129x device families. These projects are based on the bare metal examples provided in the TivaWare Software Development Kit (SDK). The TivaWare™ SDK includes Driver Library (DriverLib) APIs for all peripherals that are the building blocks for any application on a TM4C microcontroller. The provided example projects demonstrate how to use DriverLib APIs within the FreeRTOS kernel to create simple real-world applications with various device peripherals. The peripherals that are demonstrated in this report are as follows.

- Analog to Digital Converter (ADC)
- Control Area Network (CAN)
- Hardware Timer
- Inter-Integrated Circuit (I^2C)
- Pulse Width Modulator (PWM)
- Synchronous Serial Interface (SSI / SPI)
- Micro Direct Memory Access (µDMA)

2 How to Install

The following steps demonstrate how to download the latest version of TivaWare and FreeRTOS. This section can be skipped if it was done previously.

1. Download the latest TivaWare SDK from here and follow the installer instructions. If using the default installation path, the TivaWare SDK will be installed at `C:\ti\TivaWare_C_Series-2.2.0.295`. For the TivaWare directory structure, see Figure 2-1 (and notice there is already a `third_party/FreeRTOS directory`). The existing FreeRTOS installation in the TivaWare library is version 8.2.3. This needs to be updated to the latest version.

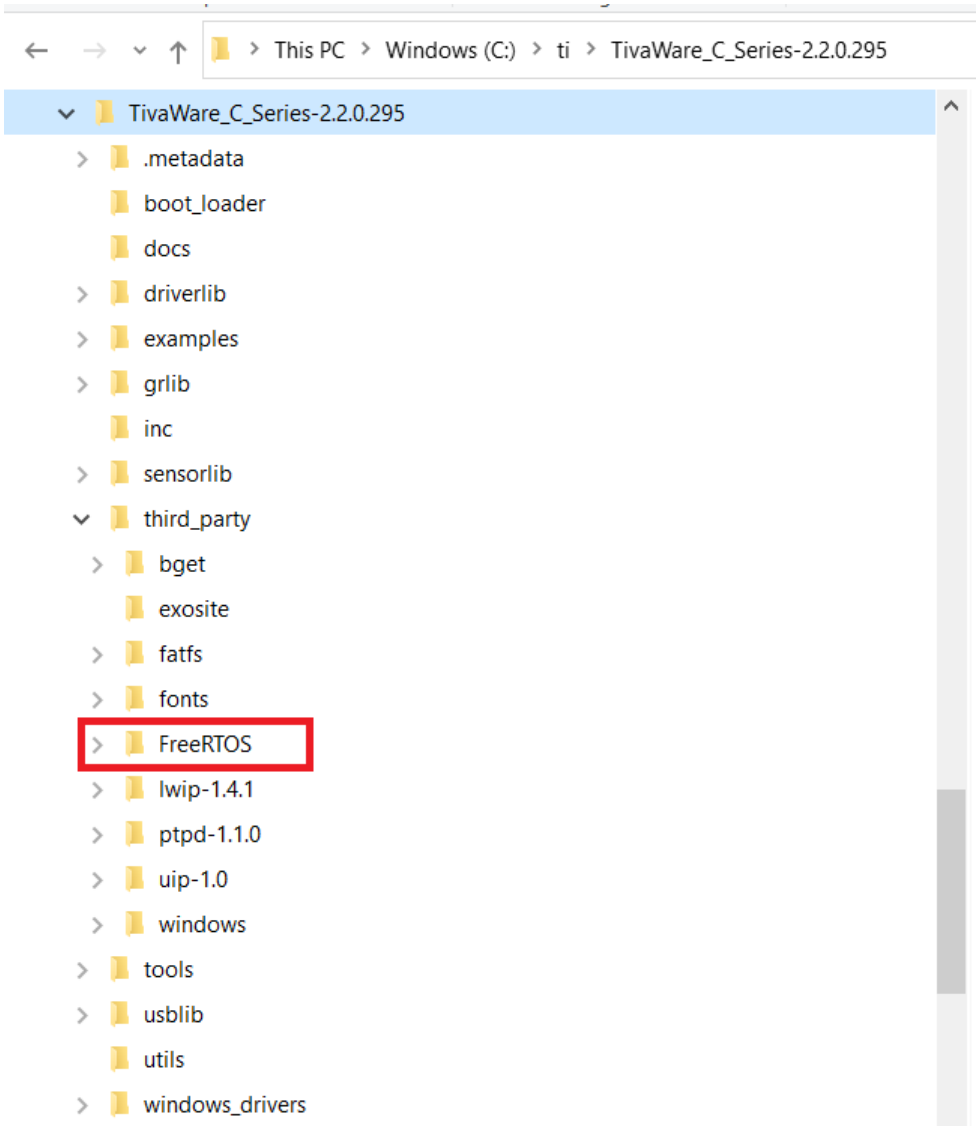

Figure 2-1. TivaWare Directory Structure

2. Download the latest version of FreeRTOS from the official download source here. At the time of the publishing of this application report, the latest version of FreeRTOS is Version 202112.00, as shown in Figure 2-2. All examples provided with this application report have been created and validated using that version.

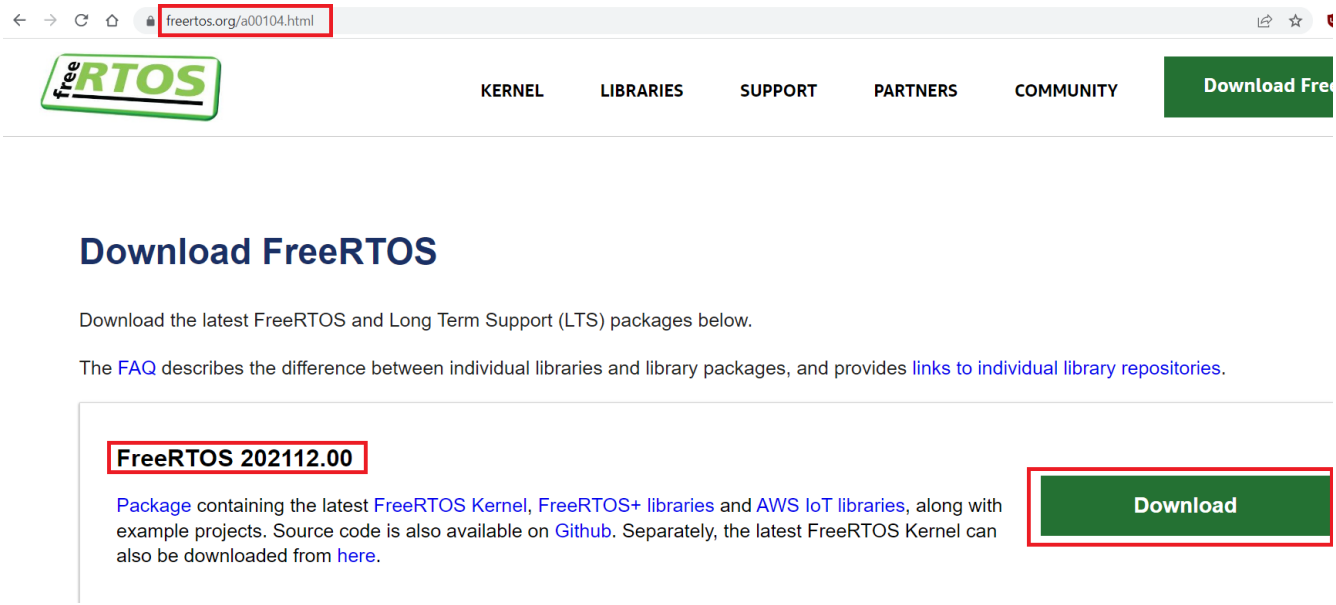

Figure 2-2. FreeRTOS Download Site

3. Unzip the FreeRTOS zip file and install to a temporary directory. View the FreeRTOS directory structure and notice the FreeRTOS folder and other folders and files, as shown in Figure 2-3.

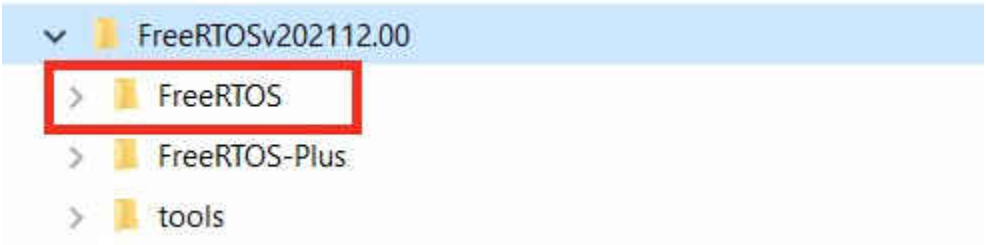

Figure 2-3. FreeRTOS version 202112.00 Directory Structure

2.1 Update the FreeRTOS Version in the TivaWare Directory

The FreeRTOS folder from the new download will need to be either moved or copied to the TivaWare library under `C:\ti\TivaWare_C_Series-2.2.0.295\third_party`. Make sure to first rename or move the existing FreeRTOS folder in the TivaWare library if there is a desire to retain it as a backup as shown in Figure 2-4.

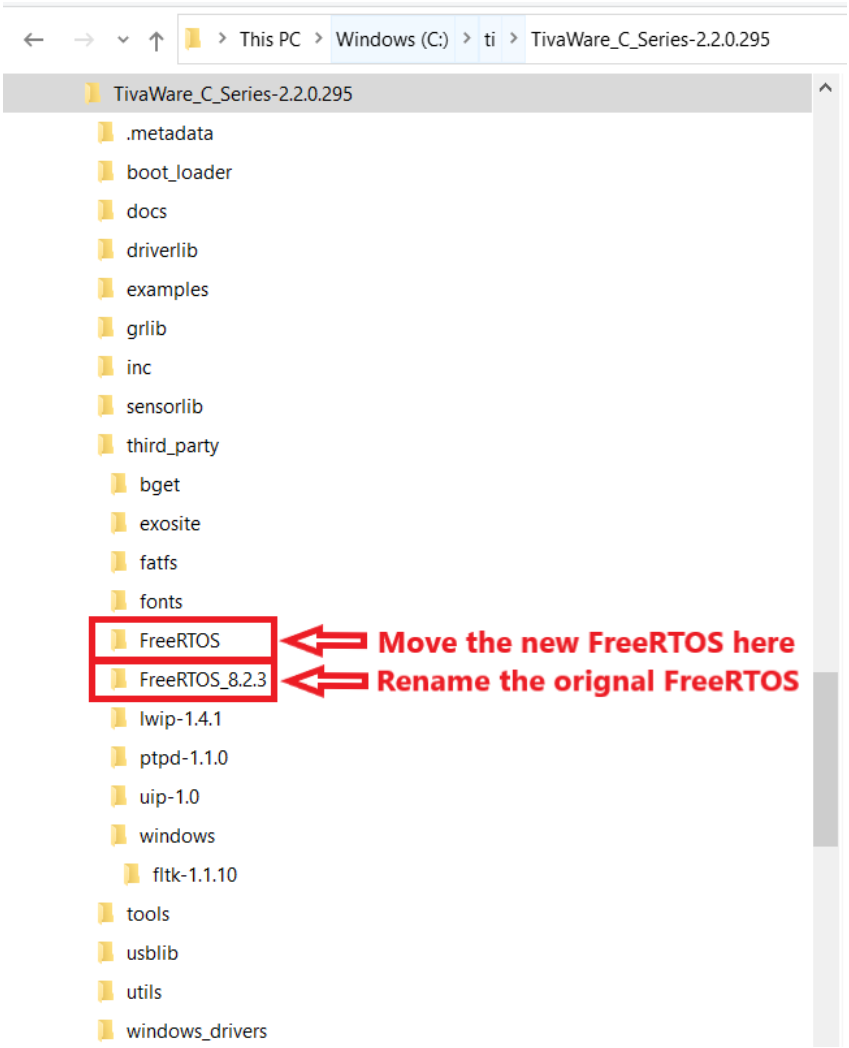

Figure 2-4. Updated TivaWare for New FreeRTOS Installation

2.2 Adding FreeRTOS Hardware Driver Files for TM4C LaunchPads

The collateral provided include new hardware driver files that were created to provide a single source for basic pinout configurations for each LaunchPad Evaluation Kit. The files are located under the `Driver Files` directory. These files are referenced in all examples that use device pins, so this step is required as part of the first time setup.

The *rtos_hw_drivers.c* and *rtos_hw_drivers.h* files for each specific LaunchPad should be copied into the TivaWare folders for that LaunchPad under the `drivers` directory. For EK-TM4C123GXL this path would be `TivaWare_C_Series-2.2.0.295\examples\boards\ek-tm4c123gxl\drivers` and for EK-TM4C1294XL this path would be `TivaWare_C_Series-2.2.0.295\examples\boards\ek-tm4c1294xl\drivers`.

3 Architecture for TM4C FreeRTOS Examples

The provided example projects are designed with the purpose of offering a compact and streamlined foundation that can be used to begin application development. In order to achieve this, the FreeRTOS kernel is used in conjunction with the TivaWare Driver Library (DriverLib) without any additional abstraction layers added. The configuration and handling of specific peripherals are contained within dedicated task files, which allow for easy re-use across multiple projects.

At this time, POSIX is not included as part of the architecture. The FreeRTOS + POSIX solution currently is a partial implementation that is only offered via GitHub. Details about that offering can be read about at this link.

In addition to DriverLib, TivaWare also includes some utilities to complement specific applications such as outputting UART messages over a terminal or using external memory sources. Be aware that these utilities were created for bare metal applications and may include APIs that were not designed to be thread safe. For the simplicity of demonstrations, there are times certain non-thread safe APIs are used in example projects. These are clearly highlighted as non-thread safe, and they are only used to demonstrate what is being executed as part of the example project.

3.1 Proper Clock Configuration

To ensure that the FreeRTOS kernel is operating at the correct tick rate, care must be taken to assign the correct system clock frequency of the TM4C microcontroller to the FreeRTOS kernel. The *configCPU_CLOCK_HZ* variable in `FreeRTOSConfig.h` holds the value of the system clock frequency. This variable must be updated accordingly for any system clock frequency changes. All examples provided are using the maximum supported clock speed for the target microcontroller which is 80 MHz for TM4C123x devices and 120 MHz for TM4C129x devices.

The device-specific data sheets for each TM4C microcontroller provides information about the valid clock speed options under the *Clock Control* section. There are limitations to how precise a frequency can be set based on the clock dividers. If a frequency that cannot be supported is input in a TivaWare clock configuration function, typically the next closest frequency will be configured. Improper clock configurations in terms of not accurately reflecting external crystals or which oscillator should be used can also lock a device until reset to factory condition. Lastly, certain peripherals have requirements for minimum system clock frequencies to be able to operate to specification including the ADC (for high speed sampling), USB, and Ethernet (TM4C129x family only) peripherals.

There are differences in how to handle the clock configuration between the two TM4C device families.

For TM4C123x devices, the system clock frequency is set with the `SysCtlClockSet` API and is determined only by the divider that is passed with the function call. The actual rate of the system clock can be then acquired with the `SysCtlClockGet` API. However, since the `FreeRTOSConfig.h` file requires hardcoded definitions of the clock rate, any changes to the system clock frequency will need to be validated without the kernel running before plugging in the new value for *configCPU_CLOCK_HZ*.

For TM4C129x devices, the system clock frequency is set with the `SysCtlClockFreqSet` API, which requires an input for the desired system clock frequency. It then returns the actual system clock frequency that was set based on what is the closest setting to the requested value. In each example provided, that value is stored in *g_ui32SysClock*, which can be used to verify that the correct frequency has been set. The clock frequency input provided to `SysCtlClockFreqSet` is the *configCPU_CLOCK_HZ* variable. Therefore, as long as a valid TM4C129x clock frequency is defined, only one change is needed for the project.

3.2 How to use Hardware Interrupts Alongside the FreeRTOS Kernel

An important element of managing an RTOS on a microcontroller is correctly handling the use of hardware interrupts from the microcontroller without disrupting the RTOS kernel and scheduler. Because FreeRTOS aims to be chip and compiler agonistic, there is no unique plug-in for device specific hardware interrupts. Instead, hardware interrupt processing occurs outside of the kernel and scheduler and must be handled by using very lean interrupt service routines (ISRs). Minimizing the cycles spent executing the ISR will allow the scheduler to take control of the application quicker.

Because most ISRs require the processing of a specific event that has been triggered, the FreeRTOS kernel provides methods to defer the processing of events from an ISR to a task that is part of the kernel and runs according to the scheduler. These deferred tasks have configured priorities that allow developers the same flexibility as ISR processing where priorities can be set to ensure the most important interrupts are processed first. With this method, the hardware ISR processing can be minimalized while still retaining priorities as required by an application.

To learn more details about deferred interrupt processing as well as FreeRTOS interrupt safe API and how context switching is performed by the kernel, see the *Interrupt Management* chapter in Mastering the FreeRTOS Real Time Kernel.

From a TM4C microcontroller perspective, configuring interrupts includes mapping the ISR to the interrupt vector table so that the ISR is executed whenever the hardware interrupt is triggered. There are two methods to correctly register a new interrupt to the interrupt vector table. These are both documented in full detail in the *TivaWare Vector Tables and IntDefaultHandler* section of the *Getting Started with TivaWare™ User's Guide*.

4 Example Project Walkthroughs

This application report includes a zip file containing example projects for the EK-TM4C123GXL LaunchPad Evaluation Kit and EK-TM4C1294XL LaunchPad Evaluation Kit. . The Code Composer Studio™ projects demonstrate how to create applications on TM4C microcontrollers by using the TivaWare SDK with the FreeRTOS kernel. They are focused on demonstrating fundamental FreeRTOS features and basic Arm microcontroller peripherals. Table 4-1 shows a list of examples.

Table 4-1. FreeRTOS Application Examples

TM4C129	TM4C123
adc_udma_pingpong	adc_udma_pingpong
adc_udma_two_channels	adc_udma_two_channels
can_multi_rx	can_multi_rx
can_multi_tx	can_multi_tx
i2c_simple_loopback	i2c_simple_loopback
i2c_tmp117_demo	i2c_tmp117_demo
pwm_interrupt	pwm_interrupt
pwm_invert	pwm_invert
ssi_quad_mode	ssi_simple_xfer
ssi_simple_xfer	
timer_edge_capture	timer_edge_count
udma_mem_transfer	udma_mem_transfer

4.1 Download and Import the Examples

The collateral provided with this application report can either be unzipped into a local directory or kept in the zip format. Both formats can be imported to the CCS.

1. To import the project into CCS, first select “File” -> ”Import”.

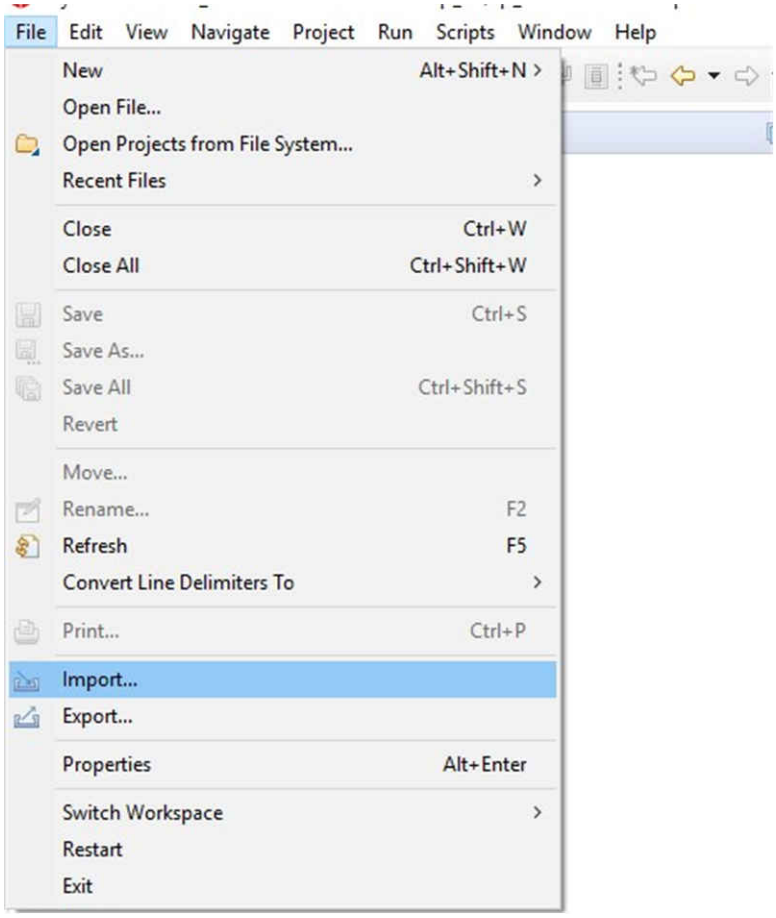

Figure 4-1. Import CCS Projects Step 1

2. Select “CCS Projects” to import the examples and then click “Next”.

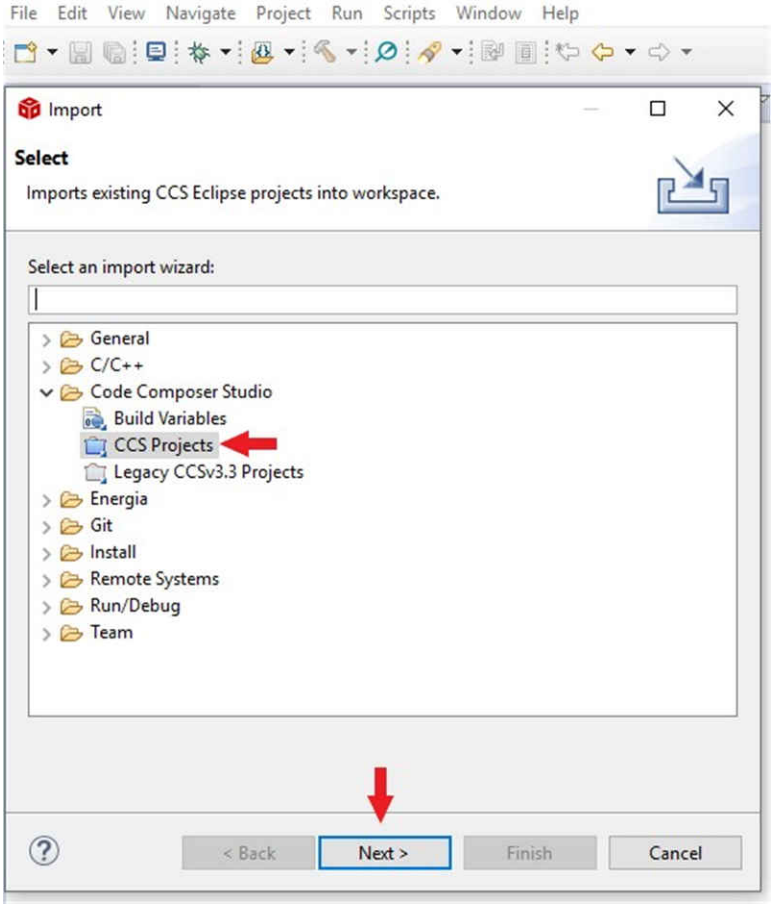

Figure 4-2. Import CCS Projects Step 2

3. Next, provide the path to either the unzipped project by selecting the first radio button or import the zip file directly by selecting the second radio button. Click the “Copy projects into workspace”.

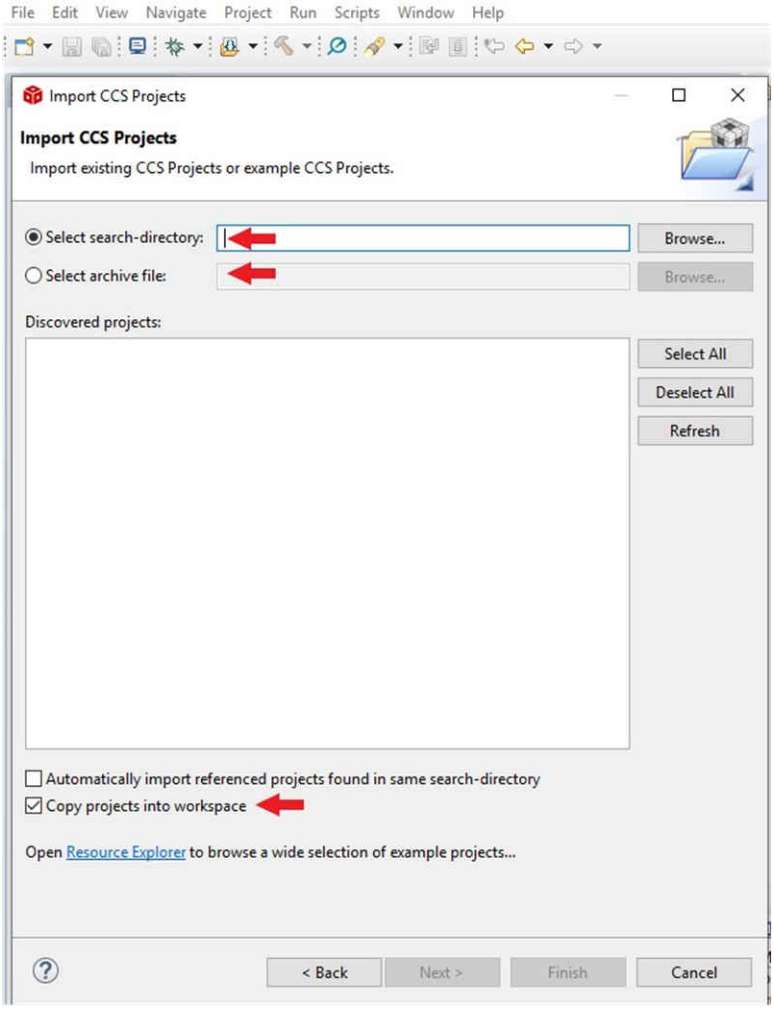

Figure 4-3. Import CCS Projects Step 3

4. After the project path is provided, discovered projects by CCS will show up. First click the “Select All” button and then click the “Finish” button to complete the import.

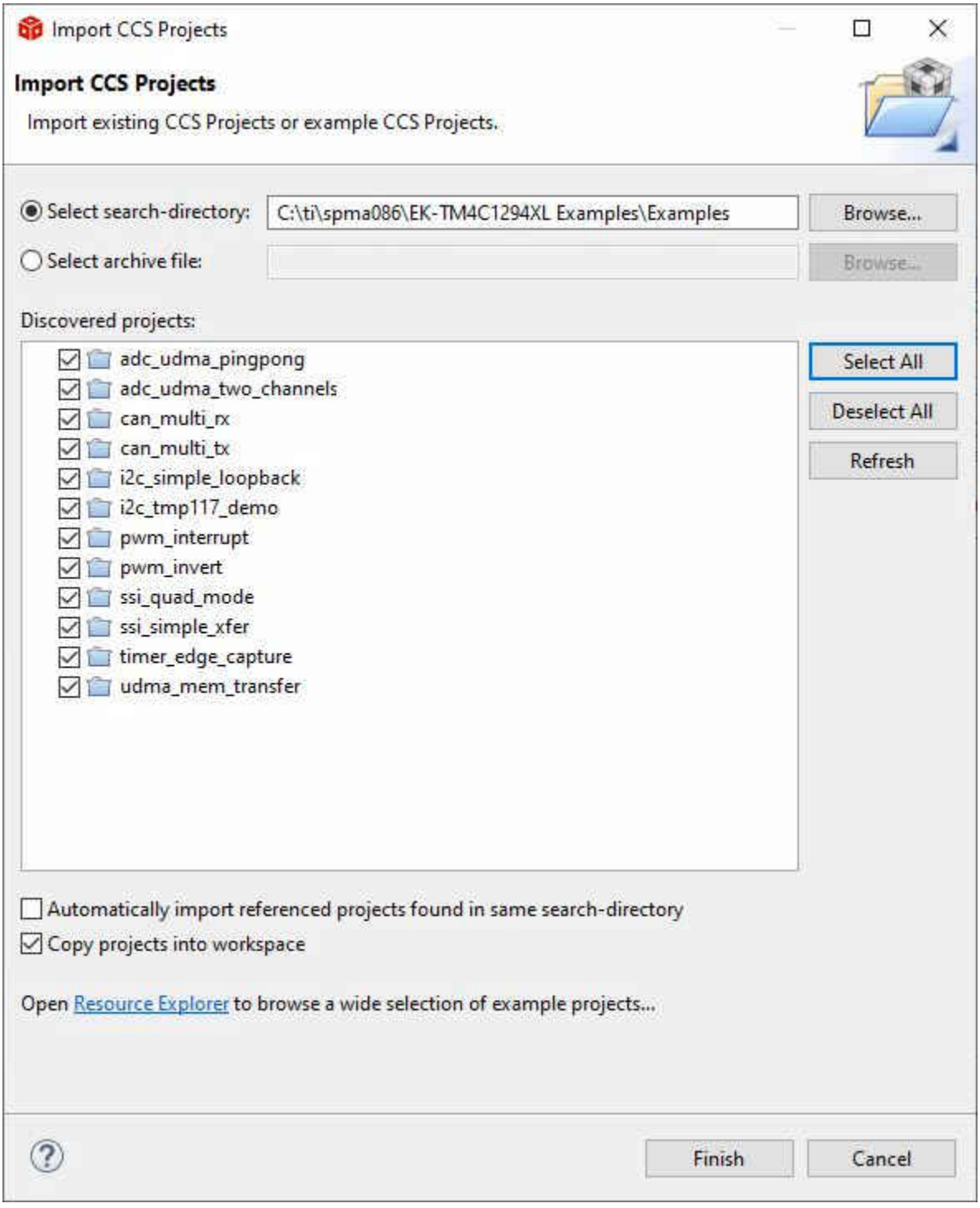

Figure 4-4. Import CCS Projects Step 4

4.2 CAN Examples

All TM4C micrcontrollers come with at least one Controller Area Network (CAN) module and this application report provides two examples that should be used in conjunction with one another to demonstrate a robust CAN application using multiple message objects. To run these examples on either TM4C LaunchPad™ Evaluation Kit, it is required to connect CAN transceivers to the BoosterPack™ headers for the associated CAN signals.

It is important that when running the CAN examples on EK-TM4C1294XL LaunchPad Evaluation Kit, the JP4 and JP5 jumpers are changed to their vertical positions. JP4 and JP5 are used to configure CAN signals to the BoosterPack 2 interface. In the default horizontal configuration, CAN is not present on the BoosterPack. UART2 goes to the BoosterPack and UART0 is connected to the virtual COM port of the ICDI. In the vertical CAN-enabled configuration, UART2 goes to the ICDI virtual serial port and CAN signals are available on the BoosterPack. For more details, see the *Tiva™ C Series TM4C1294 Connected LaunchPad Evaluation Kit - EK-TM4C1294XL*.

4.2.1 multi_can_rx

This example shows how to set up the CAN0 to receive multiple CAN messages using separate message objects for different messages, and using CAN ID filtering to control which messages are received. Three message objects are set up to receive the CAN message IDs that are used by the can_multi_tx example. Filtering is demonstrated by only receiving three of the four messages sent by the multi_tx example. As messages are received the contents are printed to the serial console.

In this example, only one task is created which processes all the data received on different message objects. The concept of event group is also used in this example where multiple events (e.g. notifications from different interrupt sources) can unblock a BLOCKed task. Event groups are another FreeRTOS feature that allows events to be communicated to tasks. Event groups allow a task to wait in the BLOCKed state for a combination of one of more events to occur. This feature is particularly useful for the CAN example because interrupts received by different message objects are handled in the same ISR. The message objects are associated with unique event IDs and form an event group. Any events in the event group can unblock the task that is in BLOCKed state. Once unblocked, the task can use the unique event ID to handle the associated message object.

4.2.2 multi_can_tx

The CAN peripheral is configured to send messages with four different CAN IDs. Two of the messages (with different CAN IDs) are sent using a shared message object. This shows how to reuse a message object for multiple messages. The other two messages are sent using their own message objects. All four messages are transmitted at random intervals. The content of each message is a test pattern. A CAN interrupt handler is used to confirm message transmission and count the number of messages that have been sent.

In this example, a total of five tasks are created. The first two tasks are mapped to message object 1 and message object 2. A changing test pattern is sent to its message object at a random interval. Once the data is written to the message object, the data is sent on the physical CAN bus. Two other tasks share the same message object 3. These two tasks do not write to message object 3 directly but rather send them through a queue. It is the task prvQueueReceiveTask1 that receives the data from the two tasks through the queue and then write to message object 3. Using queues prevents the two tasks from writing to message object 3 asynchronously to each other. The queue serves as a mutex or a gatekeeper.

4.3 Hardware Timer Examples

The concept of using hardware timers over software timers is introduced in the Developing Basic Applications With FreeRTOS on TM4C MCU and included a few simple timer examples to show fundamental configurations. The two examples provided here are taking those simple concepts and creating more advanced applications that leverage the use of edge detection on capture compare inputs.

4.3.1 timer_edge_capture

This example is specific to the EK-TM4C1294XL LaunchPad and demonstrates the use of a general purpose timer in capture compare mode to measure the duration of a high pulse. Timer 0 is configured as a split pair where both Timer A and Timer B are used to monitor for edge events on input pins. On detecting such events, the interrupts will store a capture of the current time and use that to calculate the length of the pulse.

In this example, only one task is created that handles processing the timer values that are captured by the timer interrupt service routine (ISR). The timer ISR is entered whenever a rising edge occurs on PL4 or a falling edge occurs on PL5. When entered, it stores a snapshot of the current time. On the falling edge event, a notification is sent to the processing task to indicate it an calculate the latest duration of a high pulse.

4.3.2 timer_edge_count

This example is specific to the EK-TM4C123GXL LaunchPad and demonstrates the use of a general purpose timer in down edge count mode. Timer 0 is configured to decrement each time a rising edge is seen on PF0 which maps to the capture compare Pin 0 for the timer peripheral. PF0 is connected to the SW2 switch on the EK-TM4C123GXL LaunchPad and pressing it will trigger a rising edge. The count is initialized to 10 and the match is set to 0 with interrupts enabled. That will prompt an interrupt to be triggered after 10 edges are detected on the CCP pin.

In this example, two tasks are created. The first provides a simple update on the current count so the user can see it decrement after each button press. The second handles processing the timer interrupt after 10 edges have detected by updating the interrupt counter and displaying the latest interrupt count.

4.4 I²C Examples

All TM4C microcontrollers come with multiple I²C modules and this application report provides two example projects demonstrating how to use them for basic applications. For the i2c_tmp117_demo, this example was developed using the BP-BASSENSORSMKII Sensors BoosterPack™ plug-in module.

4.4.1 i2c_simple_loopback

This example shows how to configure the I2C 0 module to send and receive data on the I²C bus. It uses loopback mode, which allows for the omission of any pull-up resistors on the I²C lines by instead using the internal bus to transfer the data.

In this example, two tasks are created. The transmit task sends data to the I²C shift register every 10 milliseconds. After the data is written, the transmit task sends a notification to the receive task. When the data is sent, a notification is asserted to the receiver task which reads the data on the I²C receiver and display it on the terminal window.

4.4.2 i2c_tmp117_demo

This example shows how to configure the I²C module to read a TMP117 High Accuracy Temperature Sensor. The I2C 2 module is configured to run at 100 kHz and read data from the TMP117 every second before calculating the temperature and displaying it over UART.

To read and convert the data from the TMP117 sensor, ASC-STUDIO was used to generate the source code which handles all register reads, writes, and delays. TivaWare I²C APIs were inserted into the *mcu_i2cTransfer* function to handle all communications with the sensor. From there simple read and data conversion APIs can be used to read / display the temperature.

In this example, only one task is created which takes a temperature reading every millisecond. The sensor data is received as a raw value that needs to be converted, so three different conversion functions are demonstrated to translate and output the result over UART as both an integer and a decimal down to a millidegree Celsius.

4.5 PWM Examples

Many TM4C microcontrollers come with a dedicated PWM module that offers more features and flexibility compared to running a hardware timer in PWM mode. The two PWM examples provided will demonstrate how to configure the PWM peripheral for output and how to use some basic features for adjusting the output.

4.5.1 pwm_interrupt

This example demonstrates how to use the PWM peripheral with interrupts to dynamically change the PWM duty cycle at a timed interval. PWM 0 is configured to output at 250Hz with an initial duty cycle of 25%. The PWM load interrupt is configured to trigger on every PWM reload.

In this example, only one task is created and it is blocked until the PWM interrupt is triggered to send a notification that unblocks it. Each time it runs, the task will increase the duty cycle by 0.1% until the duty cycle becomes 75%. The subsequent interrupt at that point causes the PWM task to reset the duty cycle to 25%.

4.5.2 pwm_invert

This example demonstrates how to use the PWM peripheral in inverted output mode. This feature allows the MCU to invert the polarity of the PWM output. The PWM task configures PWM 0 to output at 250Hz with a 25% duty cycle.

In this example, only one task is created that handles inverting that PWM output every two seconds using VtaskDelay. This will toggle the duty cycle between 25% to 75%.

4.6 SSI Examples

All TM4C microcontrollers come with multiple Synchronous Serial Interface (SSI) modules and TM4C129x devices offer the Quad-SSI (QSSI) interface as well. Synchronous Serial Interface is the TM4C-specific name for Serial Peripheral Interface (SPI). One example demonstrating a simple SSI transfer is provided for both device families, and an additional example highlights using the QSSI interface on TM4C129x devices.

4.6.1 ssi_simple_xfer

This example applications demonstrates how to configure the SSI interface to both send and receive data. The SSI interface is configured for 2 MHz clock speed, 8-bit data size, and data output per SSI Mode 0.

In this example, two tasks are created. The first is the SSI transit task that prepares data to be sent from the SSI 1 interface before initiating the full-duplex transfer by sending over the SSI 0 interface, which generates the clocks required for both data transfers. The second is the SSI receive task that receives the sent data for both SSI interfaces and prints the results over UART. Both tasks self-terminate after completion.

4.6.2 ssi_quad_mode

This example is specific to the EK-TM4C1294XL LaunchPad and demonstrates how to configure SSI to both send and receive data using the SSI quad-mode feature unique to the TM4C129x family. The SSI 0 interface sends four characters to the SSI 1 interface using the advanced Quad mode. In Quad-SSI mode, four bits are sent on each SSI Clock pulse. Once the SSI 1 interface receives the four characters in its RX FIFO, it will generate an interrupt.

In this example, two tasks are created. The transmit task transmits four bytes and then self-terminates. The receive task waits for the notification from the interrupt signaling the data has been received in its receive FIFO. Once the receive task is unblocked, it processes the data by means of sending the received bytes to the terminal window before self-terminating as well.

4.7 µDMA Examples

The Micro Direct Memory Access (µDMA) controller allows for data transfer tasks to be offloaded from the microcontroller core which results in more efficient use of the processor and memory bus. This enables programmers to minimize CPU cycles for certain tasks like peripheral-to-memory and memory-to-memory transfers. These benefits can greatly help to optimize a system using an RTOS. Three examples are provided to show how to use the µDMA in various modes as well as demonstrate both types of memory transfers.

4.7.1 adc_udma_pingpong

This example demonstrates how to use the ADC peripheral with both the µDMA and hardware timer peripherals to optimize the ADC sampling process for a single channel using the µDMA in Ping-Pong mode. One ADC channel is configured to convert an ADC input and the result is transferred via the µDMA to a buffer. Using Ping-Pong mode allows for the smooth swapping of buffers to ensure data is always being stored while also offering the application time to process a filled buffer. The timer is used to trigger the ADC measurements at a set sampling frequency of 16 kHz.

In this example, only one task is created that handles the processing of ADC conversion data. When µDMA completes the block transfer of ADC data to the buffers, it sends a notification to the processing task. The processing task takes the average of all samples and display on the terminal window. Afterwards, it signals to the µDMA controller that the buffer is ready for the next swap.

4.7.2 udma_mem_transfer

This example application demonstrates the use of the µDMA controller to transfer data between memory buffers repeatedly. It also tracks the CPU utilization and reports both CPU usage and how many memory transfer are occurring per second. The µDMA configuration includes initiating a memory-to-memory transfer that runs endlessly.

In this example, one task and one software timer is created. The software timer is configured to timeout at a one second interval. Upon timeout, a callback function is used to get a snapshot of the current CPU usage by using the CPU usage monitor utility provided as part of the TivaWare SDK. The callback function then sends a notification to the task that handles calculating the CPU's overall utilization and displaying those results on a terminal window. After displaying the results, the task puts the CPU to sleep. After ten seconds, the task self-terminates.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265