VORAGO VA416xx FreeRTOS Application Example

November 4, 2020, Version 1.0

VA4162x - VA4163x

Abstract

This document provides information on how to setup FreeRTOS on the PEB1-VA416xx development kit. This tutorial/project help to start using this small footprint but powerful operating system on the VA416xx with its many peripherals and advanced features. The project is a small generic Kiel MDK project. The project creates two tasks, each outputting its task name to the serial port, demonstrating the RTOS functionality.

Table of Contents

1 RTOS Basics

This section covers some of the terminology and structures used in RTOS based systems.

Threads are ongoing tasks implemented as an infinite loop and added to a queue. Thread and task are often used synonymously. A handle is necessary to reference a thread once it is created. Threads may be intended for static use or allowed to have dynamic creation/removal.

While traditional non-RTOS solutions typically have a system of prioritized interrupts that can take control from the main program or lower priority interrupts, an RTOS differs in that active threads of the same priority automatically share CPU control via arbitration.

There is a wide range of arbitration options. The simplest is time-sharing round-robin, where threads are switched back and forth on a regular period. It is preemptive in that a given thread does not yield control when it has completed, rather control is taken by the scheduler.

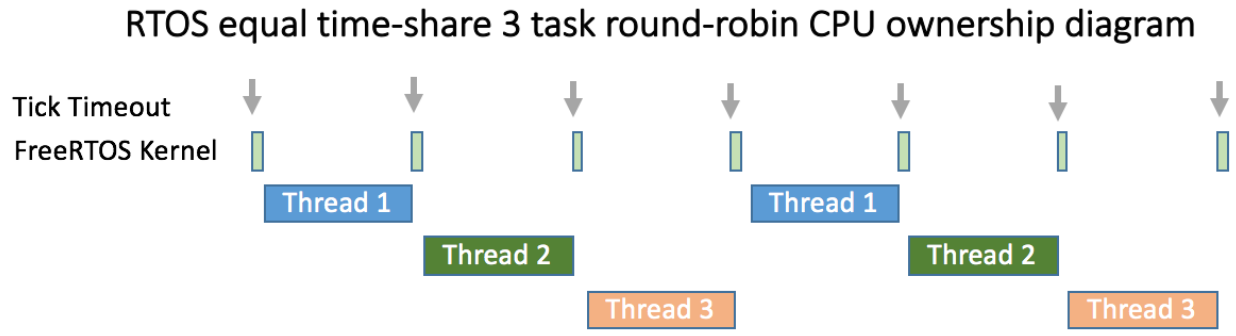

Figure 1-1 - Round-robin CPU time sharing diagram

Multiple tasks can access resources, yet it is often unsafe for a second thread to access a resource while it is already in use. Examples of shared resources could be a UART or a common variable being changed while math operations are being performed. A similar conflict can exist in a non-RTOS system when an interrupt needs to access a resource being used by the main code or a lower priority interrupt.

A straightforward solution to this is masking/disabling interrupts during a critical section to stop arbitration. Control cannot be handed to another thread in that time, but this blocks all other threads entirely, not just attempts to access that resource. A more elegant solution uses semaphores to lock off access to specific resources to prevent other threads from accessing them. Semaphores are software structures that signal the

availability of a resource. A mutex is a binary semaphore that can only be unlocked by the thread that locked it.

See http://www.freertos.org.html for more details

2 RTOS Kernel and Support Files

A standard file structure is used to allow FreeRTOS to be portable between different MCUs. This section reviews the structure which must be followed.

The FreeRTOS kernel itself is contained within 3 essential files:

1. tasks.c,
2. queue.c, and
3. list.c.

Four other support files are typically required:

1. Port.c is essential and contains the architecture-specific code. Since ARM Cortex M4 is a standardized architecture, a port.c made for any Cortex M4 based MCU should work.
2. Heap_1.c is the simplest of 5 memory management options for using a heap. A heap is a preserved area of memory that can be temporarily allocated to a task. For instance, a block of data from a serial bus may be temporarily stored in the heap area. When the data is processed, the allocated heap space is released. Memory management options are described in detail at http://www.freertos.org/a00111.html
3. FreeRTOSConfig.h is essential. It contains options specific to your application and should be located with your project.
4. Timers.c is only necessary if the application uses timers. This application note utilizes the tick counter, which is part of the Cortex M CPU but not peripheral timers.

3 Project Requirements

3.1 Hardware Requirements

Vorago PEB1-VA416X0 development kit (only CPU SBC required)
USB to TTL Serial Cable

3.2 Vorago Evaluation Kit Setup

On the EVK CPU SBC:

- Connect a micro USB cable to the micro USB receptacle J17 on the EVK PEB-1 Core card. This connector provides power and a JLink OBD debug connection to the VA416xx.

- Connect USB to UART cable to J7 for the FreeRTOS example's output. Below is the connector pinout, which is also silkscreened on the PEB-1 Core card.
  PG[0] (UART0_TX→cable RX) on pin J7-1
  PG[1] (cable TX→UART0_RX) on pin J7-2
  GND on pin J7-3

3.3 Software Requirements

Keil µVision MDK Development Software
After installing it like any other Windows application, there should be a shortcut placed on your desktop. Open it, and you should get a blank IDE workspace described further below. It is assumed that the user has gotten a previous project compiling loading and running in Keil MDK.

Tera Term (or Putty)
A serial terminal is required for viewing FreeRTOS output. Configure serial port to 230K 8N1.

4 Starting a New Keil MDK Project

Before getting started with µVision, ARMs Getting started with MDK is an excellent reference for using Keil MDK software. Once the Keil µVision application has been launched for the first time, a window opens with a blank project. On subsequent launches, the µVision IDE opens with the last closed project. A project can be returned

to a blank state by selecting the "Project →Close Project" in the pulldown menus. A blank project can be seen in Figure 4-1.

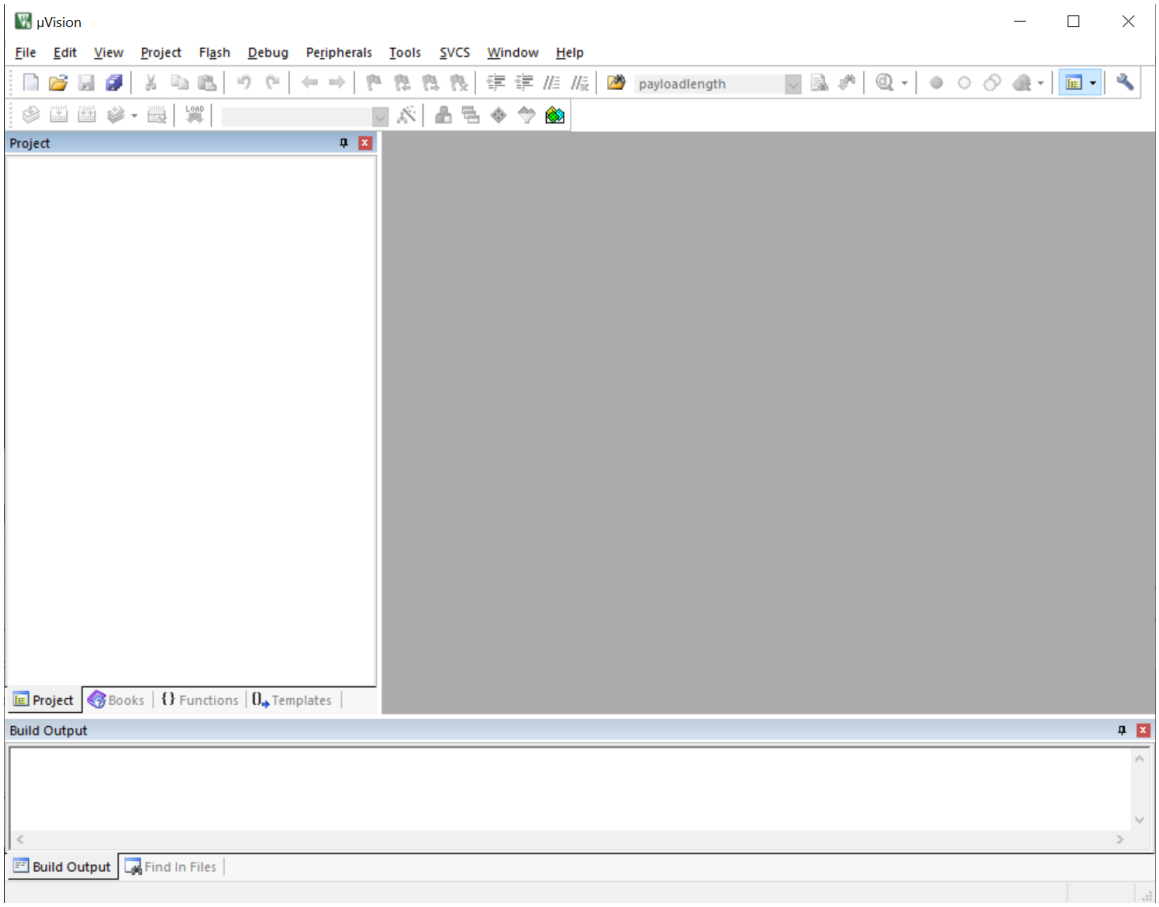

Figure 4-1 Keil MDK Blank Project

5 µVision Software Pack File Installation

Before creating a new project, software pack files may need to be installed. Pack files are a method of adding device information and software drivers into MDK in a standard manner. A pack file is simply a .zip file renamed .pack. During MDK installation, the pact filetype is associated with the Pack Installer tool ( ). Pack files are loaded into the MDK environment with the pack installer. Once installed pack files are then managed in individual projects with the Manage Run-time Environment tool ( ).

Several software pack files are needed to configure FreeRTOS in MDK. Before creating a new project, click on the pack installer tool icon ( ) in the µVision menu bar and wait for the tool to load its main menu.

The left sidebar (Devices) is a list of all the available pack files for all MDK installed microcontrollers. Look for the for VA416xx.. then select Vorago::VA416xx Series::VA416xx. If there is no Vorago::VA416xx Series shown, then the device support package has not been loaded. Visit Voragotech.com, download the VA416xx pack file, and double click on the .pack file to launch the pack installer. When this process completes the Vorago::VA416xx Series should be visible in the Pack Installer tool.

On the right-side panel are various ARM CMSIS, Keil pack files. All of the following pack files are part of the Keil MDK installation and must be shown and have a green pack icon next to them. See Figure 5-1. If any pack shown below is not available, please reload Keil MDK or contact Keil for assistance.

ARM::CMSIS
ARM::CMSIS-Driver
ARM::CMSIS-FreeRTOS
Keil::ARM_Compiler

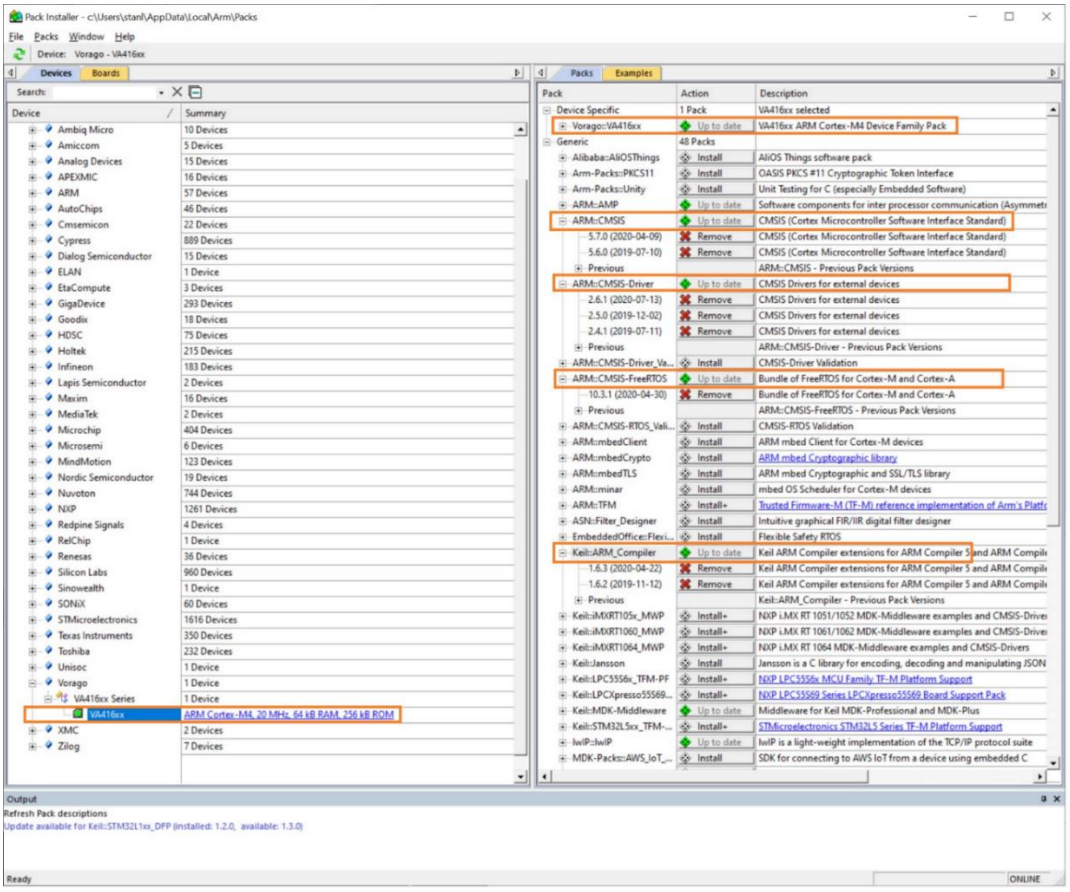

Figure 5-1 Keil MDK Pack Installer Setup

Once the required pack files are visible into the KEIL MDK environment, a new project can be started.

6 µVision Project Setup

To create a new project, click on the "Project tab→New µVision Project." Before creating your project, the µVision IDE doesn't automatically create a subdirectory. Since it is best to have all source files in one place, create a directory got the project first. For this example, name the folder "Vorago VA416xx FreeRTOS "and the project "Vorago VA416xx FreeRTOS." Reference Figure 6-1.

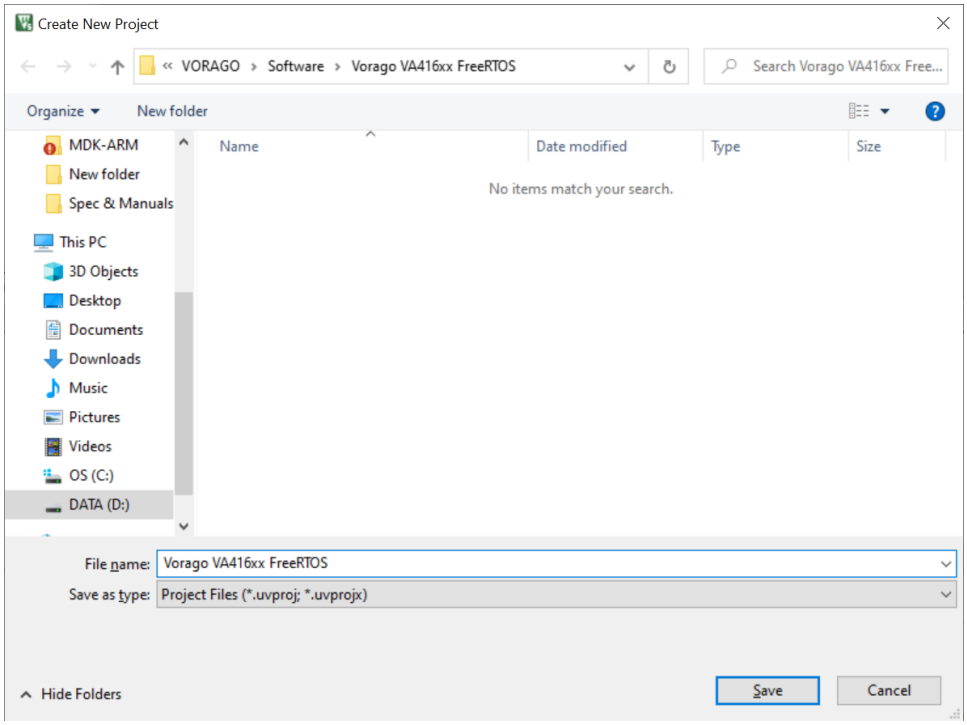

Figure 6-1 Setting the Project Directory and Name

Once the project directory and name are created, the "Save" button opens the target selection dialog box to define the project's target microcontroller selection. Select the Vorago::VA416xx Series::VA416xx device, as shown in Figure 6-2.

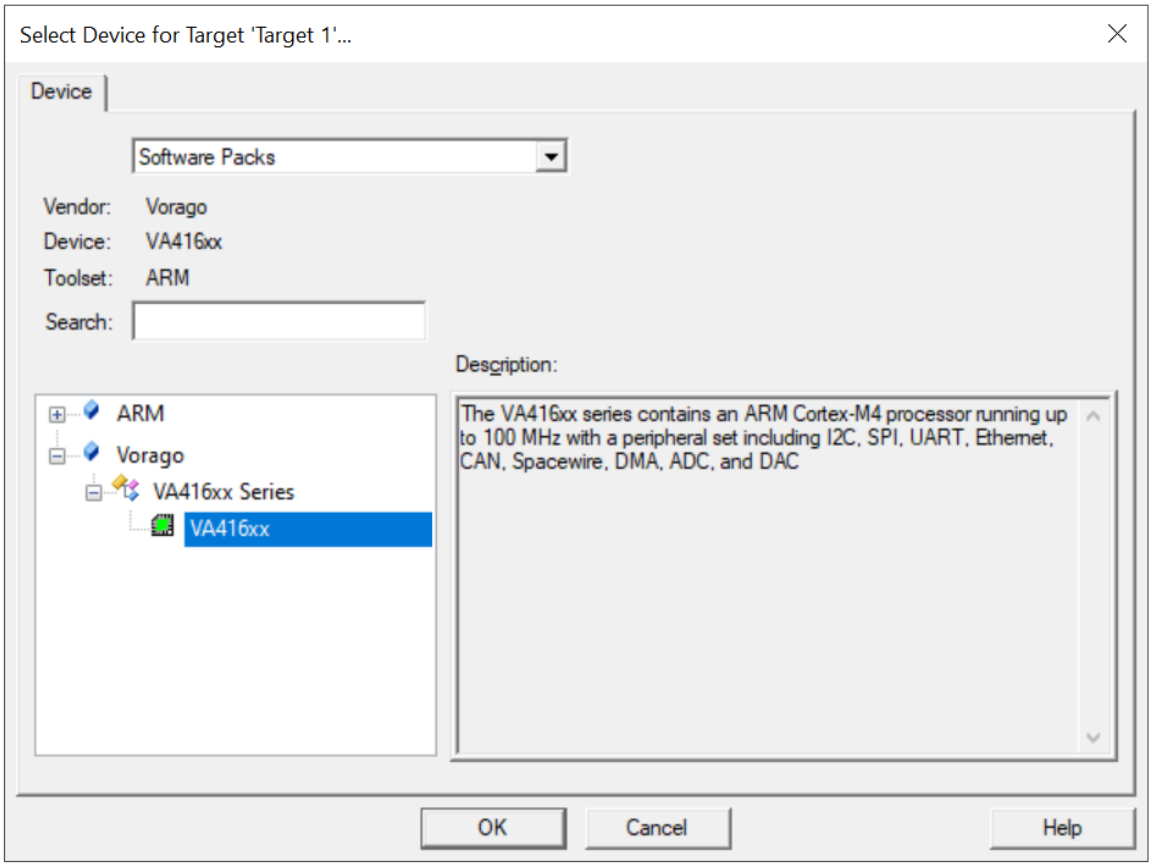

Figure 6-2 Setting Project Microcontroller Target

After these steps, we have to select what pack files are to be used in the project. The pack file selection is accomplished with the Manage Run-time Environment (). The blank dialog box is shown in Figure 6-3

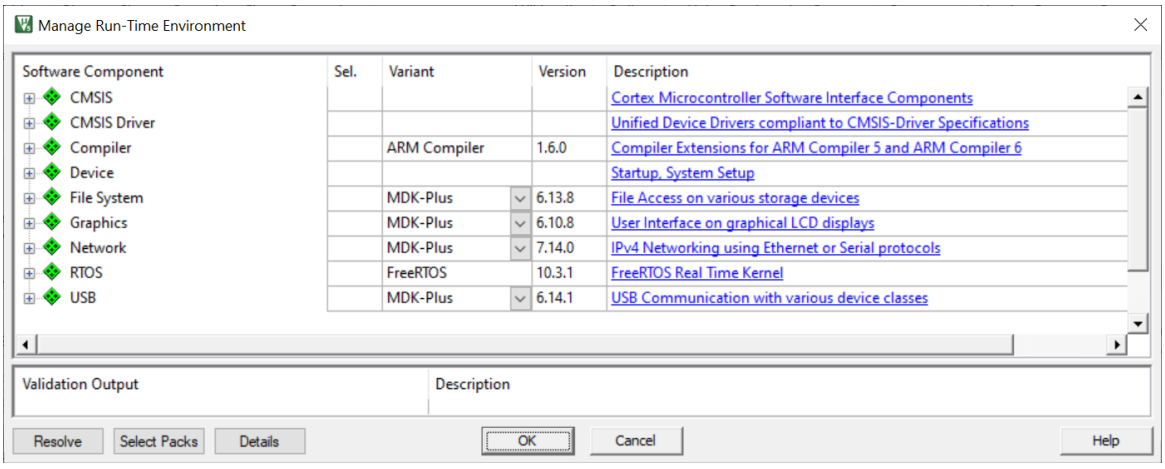

Figure 6-3 Default Manage Run-time Environment

Some pack files are dependent on others. During selection, if any checkboxes turn orange or warnings are shown in the validation output dialog box, click on the "Resolve" button, and the required packages are added to the µVision project. Note the two additional pulldown selections required. Figure 6.4 shows all the required pack files for the FreeRTOS example.

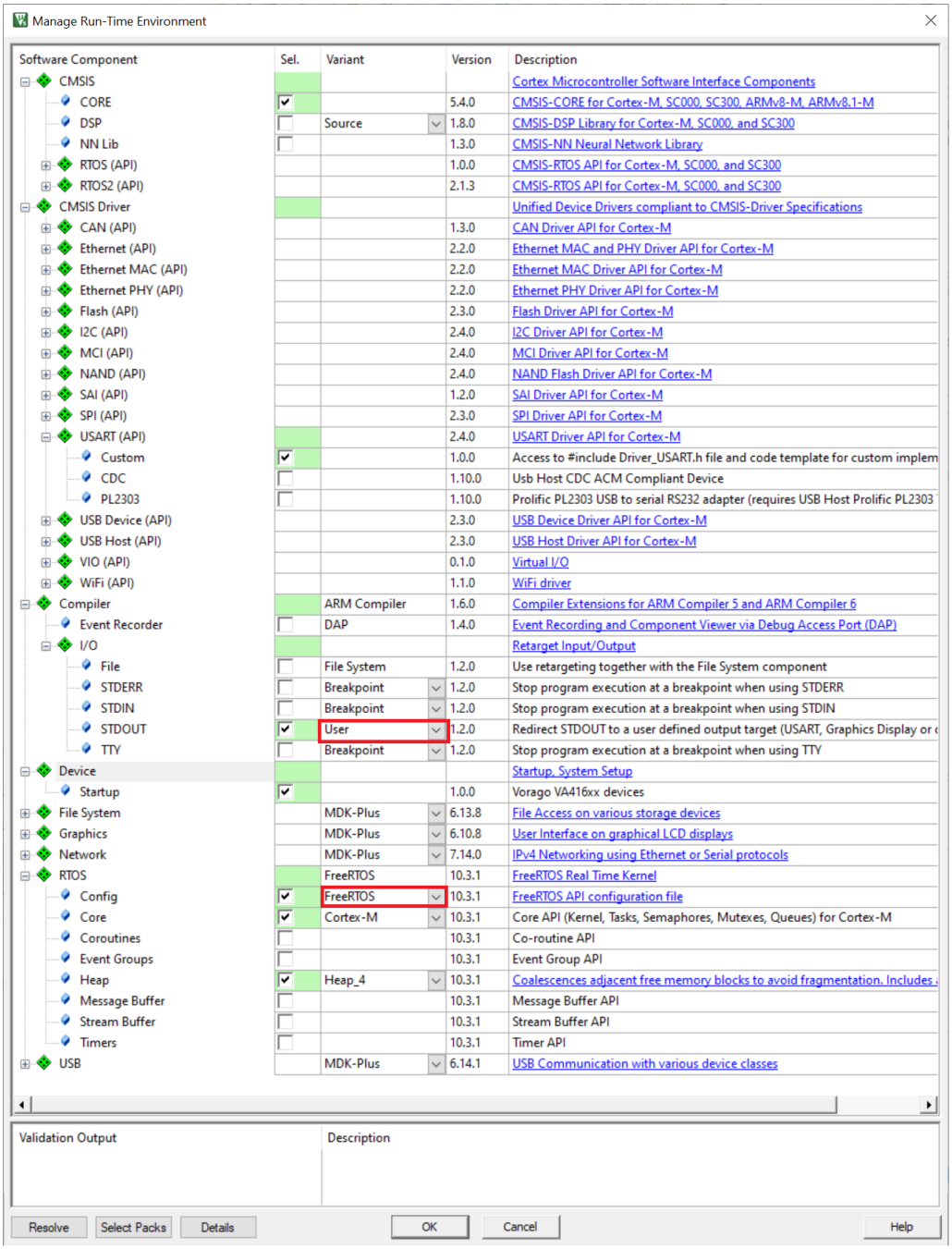

Figure 6-4 Full FreeRTOS Pack File List

Exiting the Manage Run-time Environment and returning to the µVision project, many files have been added to the project (see Figure 6-5). These files are the Vorago device-specific files and files needed for the RTOS. If any selections are not available, return to Section 5 µVision Software Pack File Installation and ensure all proper packs are installed.

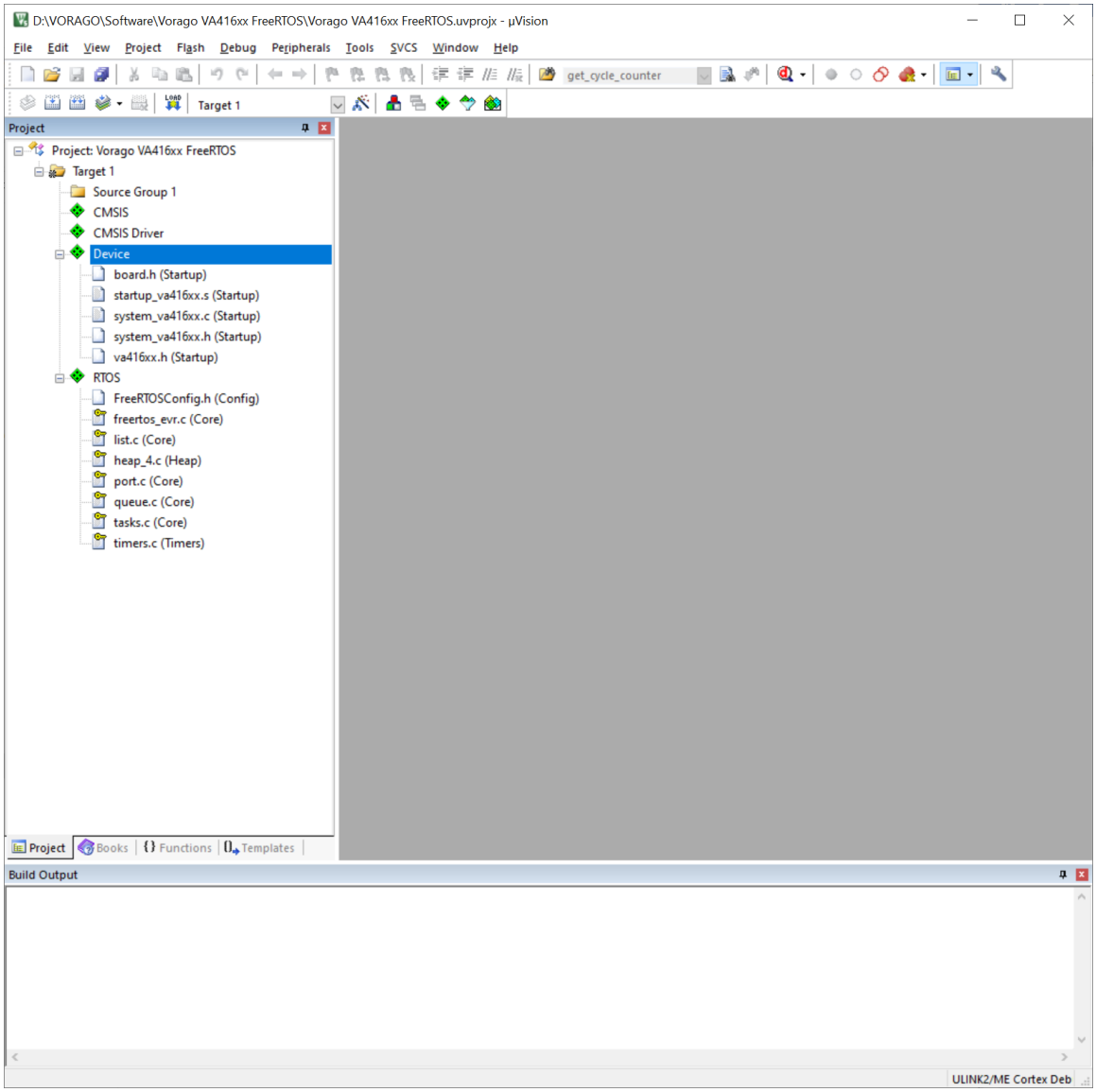

Figure 6-5 Example Project Imported Pack Files

For character output, STDOUT needs to be redirected. For printf() to use the device UART, MDK provides a file named retarget_io.c. Device-specific user functions that connect putchar()of the STDOUT to the USART driver is required.

Figure 6-6 shows how to create a user code template (stdout_USART.c) with information from the pack files and MDK to perform the redirection.

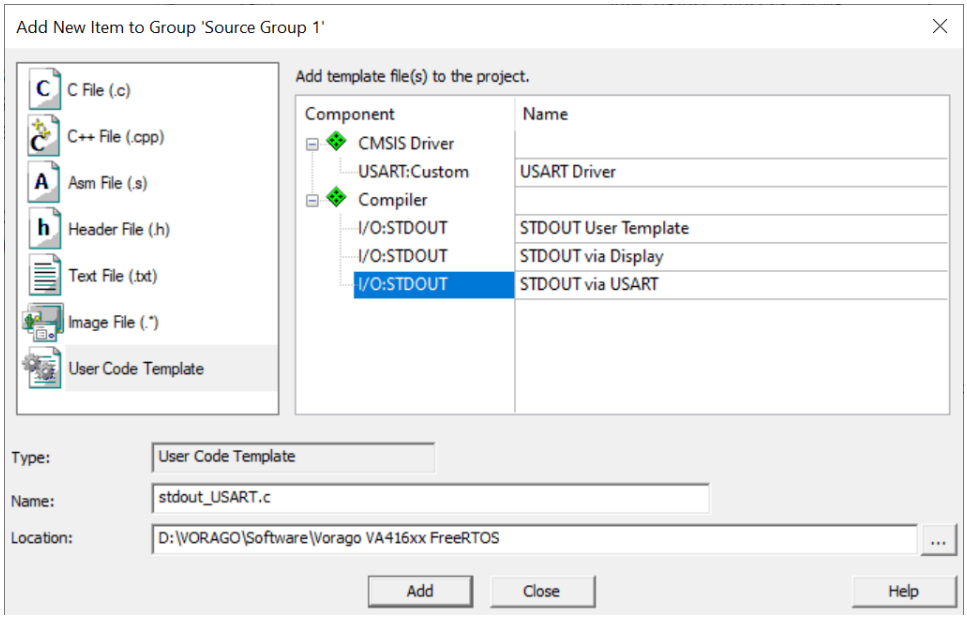

Figure 6-6 Creating stdout File

Using the User Code Template function in MDK pulls in header support files like Driver_USART.h, Driver_Common.h, and any other header files as required. Using the Add New Item function also adds the stdout_USART.c and headers to the project file.

Returning to the µVision project, open stdout_USART.c and delete the file's entire text. Replace the deleted code with the code from Appendix A.

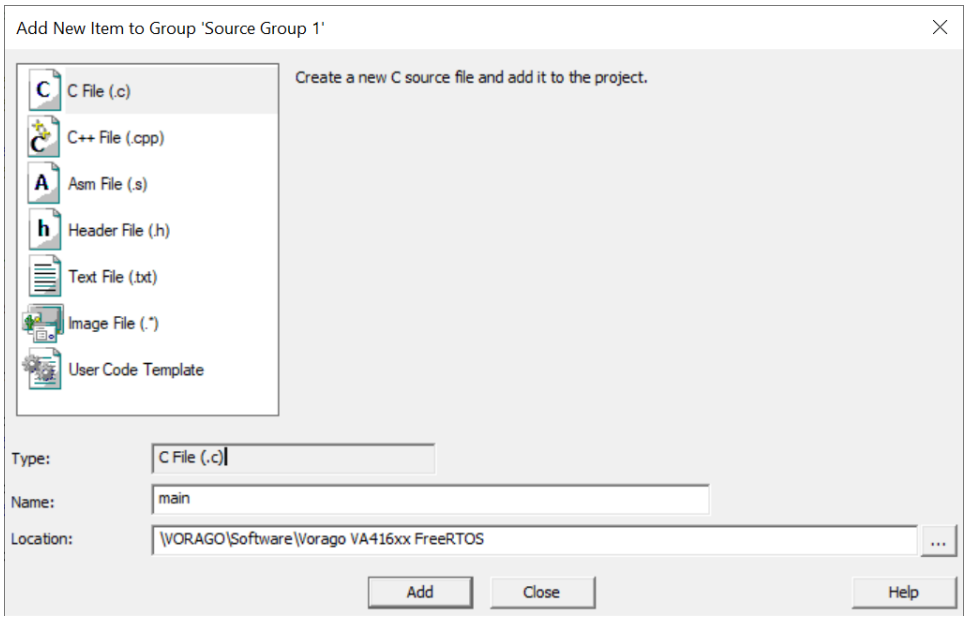

Figure 6-7 Main.c Creation

Figure 6-7 shows the creation of the main.c file, it is created similar to the creation of the stdout_USART file. Since it is created with the C File selection and not the User Code Template, no additional files are created or included but, the file main.c is added to the project.

Open the empty main.c file and insert the code from Appendix B.

7 Running the Project

Compile the project (F7) and download flash (F8) to load and run the project.

In main.c, the following happens:

- VA416xx is initialized using SystemInit() call
- Peripherals are enabled by writing to VOR_SYSCONFIG
- The system clock is set by writing to VOR_CLKGEN
- UART0 is configured by the stdout_init() call.
- Two tasks are created, each using the xTaskCreate() FreeRTOS call
- A binary semaphore is created with the xSemaphoreCreateBinary() FreeRTOS call
- The FreeRTOS task scheduler is started with the vTaskStartScheduler() FreeRTOS call

Each task will:

- Check for the printf semaphore
- Use printf functions
- Release the printf semaphore
- Delay

Once the FreeRTOS scheduler is running:

- The two tasks run with the same priority, so they have equal execution time.
- The binary semaphore is used to prevent the two tasks from competing for the UART resource.

As each task is placed in the run mode, it waits for the semaphore, uses printf(), and then releases the semaphore back to be used by another task. If a task does not have

the semaphore, it becomes blocked and waits for the semaphore to be released, and ownership is taken to execute its printf().

There is a short delay in each task to ease reading the serial output. Without delay, serial output is rapid and continuous.

Figure 7-1 Project Sample OutputShows the terminal output of the running project.

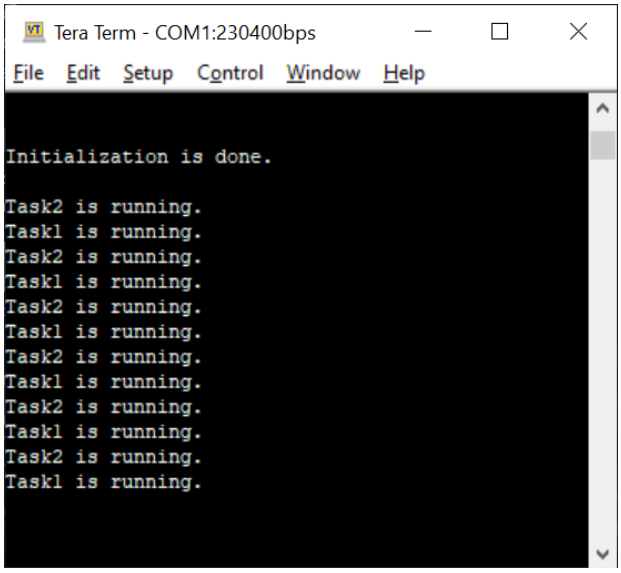

Figure 7-1 Project Sample Output

8 Next Steps

The FreeRTOS application, along with the Vorago VA416xx, creates a powerful system. An abundance of peripheral and peripheral modes in the VA416xx can be integrated into the many functions within FreeRTOS. Explore and expand your knowledge of both hardware and software by building on this simple example. Additional projects and idea can be found on the FreeRTOS website: https://www.freertos.org/

Appendix A Driver_USART.c code:

```
/*-----------------------------------------------------------------------------
 * Name:    stdout_USART.c
 * Purpose: STDOUT USART Template
 * Rev.:    1.0.0
 *-----------------------------------------------------------------------------*/

/* Copyright (c) 2013 - 2015 ARM LIMITED

   All rights reserved.
   Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions are met:
   - Redistributions of source code must retain the above copyright
     notice this list of conditions and the following disclaimer.
   - Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions and the following disclaimer in the
     documentation and/or other materials provided with the distribution.
   - Neither the name of ARM nor the names of its contributors may be used
     to endorse or promote products derived from this software without
     specific prior written permission.
   *
   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
   AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
   ARE DISCLAIMED. IN NO EVENT SHALL COPYRIGHT HOLDERS AND CONTRIBUTORS BE
   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
   CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
   SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
   INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
   CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
   ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
   POSSIBILITY OF SUCH DAMAGE.
   ---------------------------------------------------------------------------*/
/*******************************************************************************************
 * @file     stdout_USART.c
 * @version  V0.1
 * @date     12 October 2020
 *
 * @note
 * VORAGO Technologies
 *
 * @note
 * Copyright (c) 2013-2020 VORAGO Technologies.
 *
 * @par
 * BY DOWNLOADING, INSTALLING OR USING THIS SOFTWARE, YOU AGREE TO BE BOUND BY
 * ALL THE TERMS AND CONDITIONS OF THE VORAGO TECHNOLOGIES END USER LICENSE AGREEMENT.
 * THIS SOFTWARE IS PROVIDED "AS IS." NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 * OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. VORAGO TECHNOLOGIES
 * SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
 * DAMAGES, FOR ANY REASON WHATSOEVER.
 *
 *******************************************************************************************/
```

```
#include "Driver_USART.h"
#include "va416xx.h"
#define UART_CALC_CLOCKSCALE(_scc,_baud) ((_scc / (_baud * 16)) << \

UART_CLKSCALE_INT_Pos) |  \
                                 (((((_scc % (_baud * 16)) * \
                           (_baud * 8)) / \
                           (_baud * 16))) << \
                           UART_CLKSCALE_FRAC_Pos)

//-------- <<< Use Configuration Wizard in Context Menu >>> --------------------
// <h>STDOUT USART Interface

//   <o>Connect to hardware via Driver_USART# <0-255>
//   <i>Select driver control block for USART interface
#define USART_DRV_NUM           0

//   <o>Baudrate
#define USART_BAUDRATE          230400
// </h>
#define _USART_Driver_(n)  Driver_USART##n
#define  USART_Driver_(n) _USART_Driver_(n)

extern ARM_DRIVER_USART  USART_Driver_(USART_DRV_NUM);
#define ptrUSART       (&USART_Driver_(USART_DRV_NUM))

/*****************************************************************************
 ** Start of Serial IO function.
 ** @brief
 *****************************************************************************/
int stdout_init(void)
{
  VOR_SYSCONFIG->PERIPHERAL_CLK_ENABLE |= CLK_ENABLE_UART0 | CLK_ENABLE_IOCONFIG | CLK_ENABLE_PORTG;
  VOR_SYSCONFIG->PERIPHERAL_RESET &= ~SYSCONFIG_PERIPHERAL_RESET_UART0_Msk;
  __NOP();
  __NOP();
  VOR_SYSCONFIG->PERIPHERAL_RESET |= SYSCONFIG_PERIPHERAL_RESET_UART0_Msk;

// initialize port G for UART0
  VOR_IOCONFIG->PORTG[0] |= 0x00002000;   // PORTG.0 is UART0 Tx.
  VOR_IOCONFIG->PORTG[1] |= 0x00002000;   // PORTG.1 is UART0 Rx.
// initialize UART0
  VOR_UART0->IRQ_ENB     = 0x00000001;
  VOR_UART0->CLKSCALE    = 0x000001B2;
  VOR_UART0->CLKSCALE      = UART_CALC_CLOCKSCALE(SystemCoreClock/4, 230400); // APB2 divide by 4
  VOR_UART0->ENABLE      = 0x00000003;
return (0);
}


/**
  Put a character to the stdout
  \param[in]   ch  Character to output
  \return          The character written, or -1 on write error.
*/
int stdout_putchar (int ch) {
  uint32_t timeout = 100000;
  uint8_t buf[1];

  buf[0] = ch;
```

```
  // Block until there is room on the FIFO to transmit a byte
    while (( VOR_UART0->TXSTATUS & UART_TXSTATUS_WRRDY_Msk) == 0){  // wait for Tx ready
      timeout--;
      if(timeout == 0)
      {
        return(0xffffffff);   // return -1, compiler won't complain
      }
    }
    VOR_UART0->DATA = *buf;
  return (ch);
}
```

Appendix B main.c code:

```
/*-----------------------------------------------------------------------------
 * Name:    main.c
 * Purpose: main Template
 * Rev.:    1.0.0
 *----------------------------------------------------------------------------*/
/* Copyright (c) 2013 - 2015 ARM LIMITED

   All rights reserved.
   Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions are met:
   - Redistributions of source code must retain the above copyright
     notice this list of conditions and the following disclaimer.
   - Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions and the following disclaimer in the
     documentation and/or other materials provided with the distribution.
   - Neither the name of ARM nor the names of its contributors may be used
     to endorse or promote products derived from this software without
     specific prior written permission.
   *
   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
   AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
   ARE DISCLAIMED. IN NO EVENT SHALL COPYRIGHT HOLDERS AND CONTRIBUTORS BE
   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
   CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
   SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
   INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
   CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
   ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
   POSSIBILITY OF SUCH DAMAGE.
   ----------------------------------------------------------------------------*/

/* Include files */
#include "va416xx.h"
#include <stdio.h>
#include <stdlib.h>
#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"
/* The task functions prototype*/
void vTask1( void *pvParameters );
void vTask2( void *pvParameters );
/* Task parameter to be sent to the task function */
const char *pvTask1  = "Task1 is running.";
const char *pvTask2  = "Task2 is running.";
/* Extern functions */
extern void SystemInit(void);
extern void SystemCoreClockUpdate(void);
extern int stdout_init (void);
/*-----------------------------------------------------------*/
/* Global semaphore variable */
SemaphoreHandle_t xSemaphore = NULL;
int main( void )
{
/* Board initializations */
        SystemInit();
/* Initializes the MCU clock, PLL will be used to generate main MCU clock */
        VOR_SYSCONFIG->PERIPHERAL_CLK_ENABLE |= CLK_ENABLE_UART0 | CLK_ENABLE_IOCONFIG | CLK_ENABLE_PORTG;
        VOR_SYSCONFIG->PERIPHERAL_CLK_ENABLE |= CLK_ENABLE_CLKGEN | CLK_ENABLE_UART0 | CLK_ENABLE_IOCONFIG |
CLK_ENABLE_PORTG;
        // initialize clock to maximum (100MHz)
        VOR_CLKGEN->CTRL0 = 0x87ECBB1A;
        VOR_CLKGEN->CTRL1 = 0X00000010;
        SystemCoreClockUpdate();

/* Initialize the serial I/O(console ), making standard output to be send to USART1 */
        stdout_init();
        printf("\033[0H\033[2JInitialization is done.\r\n\n");
        /* Create one of the two tasks. */
        xTaskCreate(vTask1,                  /* Pointer to the function that implements the task. */
                    "Task 1",                /* Text name for the task.  This is to facilitate debugging only. */
                    configMINIMAL_STACK_SIZE, /* Stack depth in words. */
                    (void*)pvTask1,          /* We are not using the task parameter. */
                    1,                       /* This task will run at priority 1. */
                    NULL );                  /* We are not using the task handle. */
```

```
        /* Create the other task in exactly the same way. */
        xTaskCreate( vTask2, "Task 2", configMINIMAL_STACK_SIZE, (void*)pvTask2, 1, NULL );
        /* Create a binary semaphore */
        xSemaphore = xSemaphoreCreateBinary();
        /* make the semaphore token available for the first time */
        xSemaphoreGive( xSemaphore);
        /* Start the scheduler so our tasks start executing. */
        vTaskStartScheduler();

        /* If all is well, we never reach here as the scheduler is be
        running.  If we reach here, then it is likely that there was insufficient
        heap available for the idle task to be created. */
        for( ;; );
}
/*-----------------------------------------------------------*/
void vTask1( void *pvParameters )
{
char *pcTaskName = (char *) pvParameters;
        /* Task is implemented in an infinite loop. */
        for( ;; )
        {
                /* Take semaphore */
                xSemaphoreTake(xSemaphore,(TickType_t) portMAX_DELAY);
                /* Print out the name of this task. */
                printf( "%s\r\n",pcTaskName );
                /* Give semaphore */
                xSemaphoreGive(xSemaphore);
                /* Delay for a period. */
                vTaskDelay( 2000 / portTICK_PERIOD_MS );
        }
}
/*-----------------------------------------------------------*/
void vTask2( void *pvParameters )
{
char *pcTaskName = (char *) pvParameters;
        /* Task is implemented in an infinite loop. */
        for( ;; )
        {
                /* Take semaphore */
                xSemaphoreTake(xSemaphore,(TickType_t) portMAX_DELAY);
                /* Print out the name of this task. */
                printf( "%s\r\n",pcTaskName );
                /* Give semaphore */
                xSemaphoreGive(xSemaphore);
                /* Delay for a period. */
                vTaskDelay( 2000 / portTICK_PERIOD_MS );
        }
}
```

9 Revision History

Date	Version	Sections	Description
11/04/2020	1.00	All	Initial draft

The use of this product is subject to the manufacturer's standard terms and conditions available on the manufacturer's website here.

VORAGO Technologies
1501 S MoPac Expressway, Suite #350
Austin, TX 78746
www.voragotech.com

Email: info@voragotech.com
Phone: (512) 347-1800