|  |
| --- |
|  |
| **EE489 / EE589 Real-Time Embedded Systems Design** |
| Lab #1  *Dianzhi Yu* |
|  |
| *1/18/2018* |

|  |
| --- |
|  |

# Introduction

This lab project is based on Atmel Studio software and SAM D21 chip. We will the do some examples about task management in RTOS and all tasks are implemented as C functions. First, we create two tasks and then start tasks executing. We will get the results that the tasks simply print out a string periodically by using a null loop to create a period delay. Both tasks are created at the same priority and are identical except for the string that they print. And then we will create a single task and use two instances of the same function to remove the duplication by using the parameter pvParameter to pass into each task instance the string. So far, all the tasks are at the same priority. Then, we change the priority of the two tasks at different levels to observe what happens.

1. **A list of all FreeRTOS API functions being used**

xTaskCreate ( vTask1, “Task 1”, 200, NULL, 1, NULL);

xTaskCreate ( vTask2, “Task 2”, 200, NULL, 1, NULL);

This function, which is for creating tasks, has six arguments. The first one argument means pointer to the function that implements the task. The second one means Text name for the task. The third one is stack depth in words. The fifth argument shows the priority of the task. If no use for the task handle, we set to NULL.

Even though we create two tasks, which are identical except the print of text string, we need to create two instances of a single task implementation vTaskFunction to remove the duplication.

First, we need define two global constant string variables that will be passed in as the task parameters.

const char \*pcTextForTask1 = "Task 1 \r\n";

const char \*pcTextForTask2 = "Task 2 \r\n";

Then create a new task function named as vTaskFunction and change the type of pcTaskName from const char \* to char \* without the value initialization. Add the statement to assign the function input parameter pvParameters to pcTaskName.

static void vTaskFunction( void \*pvParameters );

{

char \*pcTaskName;

volatile unsigned long ul; // The string to print out is passed in via the parameter.

pcTaskName = (char \*)pvParameters;}

In the main function, replace the first and fourth parameter

xTaskCreate( vTaskFunction, "Task 1", 200, (void \*)pcTextForTask1, 1, NULL );

xTaskCreate( vTaskFunction, "Task 2", 200, (void \*)pcTextForTask2, 1, NULL );

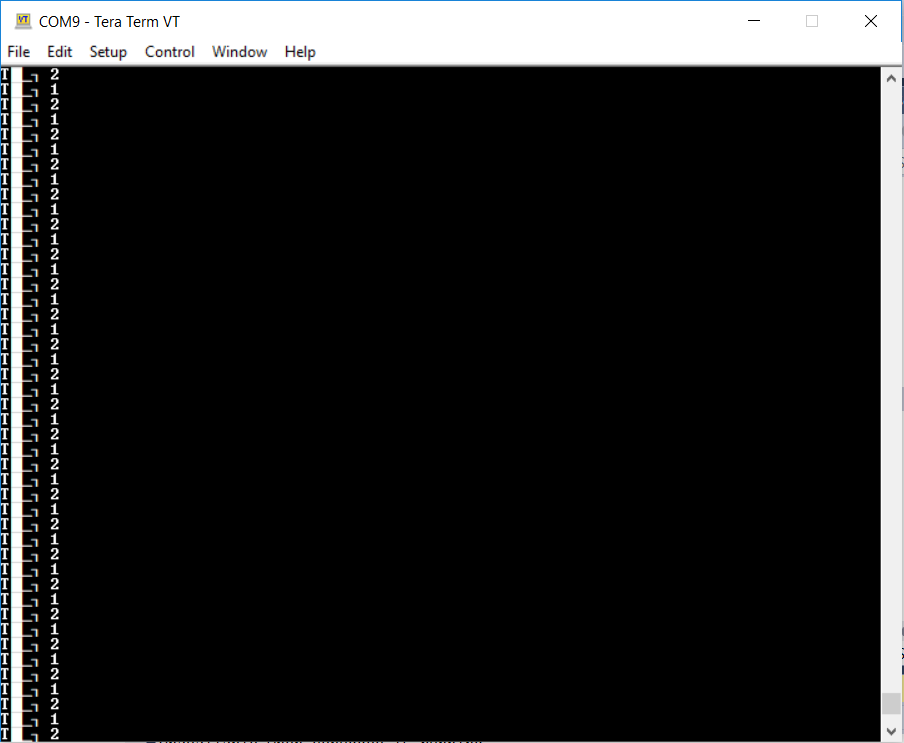
Finally, we can set the fifth parameter to different levels to make two tasks at different priorities.

xTaskCreate( vTaskFunction, "Task 1", 200, (void \*)pcTextForTask1, 1, NULL );

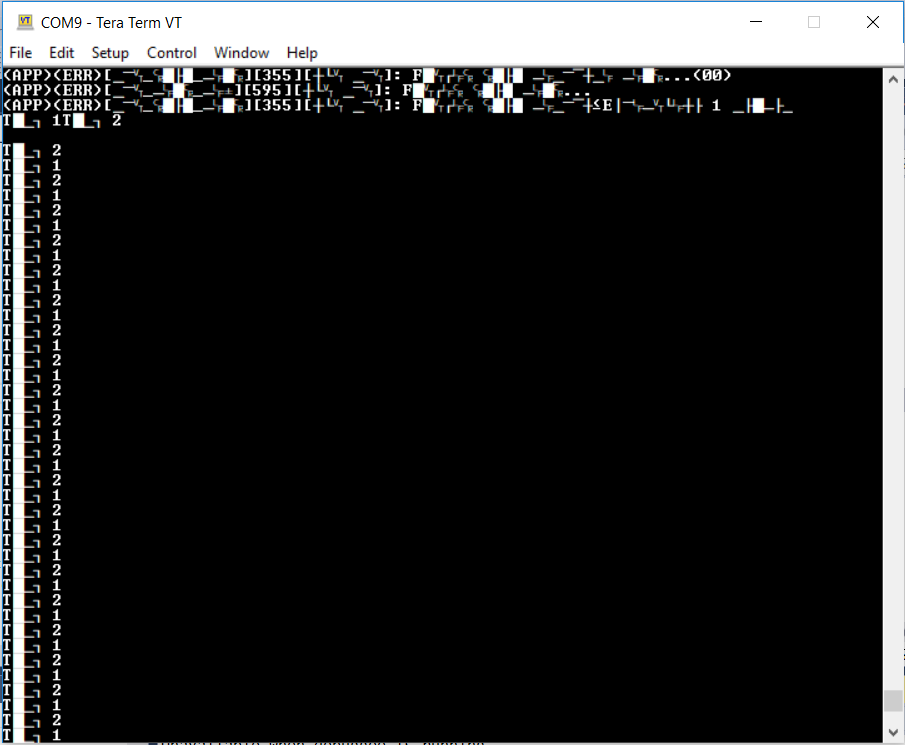
xTaskCreate( vTaskFunction, "Task 2", 200, (void \*)pcTextForTask2, 2, NULL );

According to different priorities, Task 2 is always in the Running state and Task 1 is always in the Not Running state.

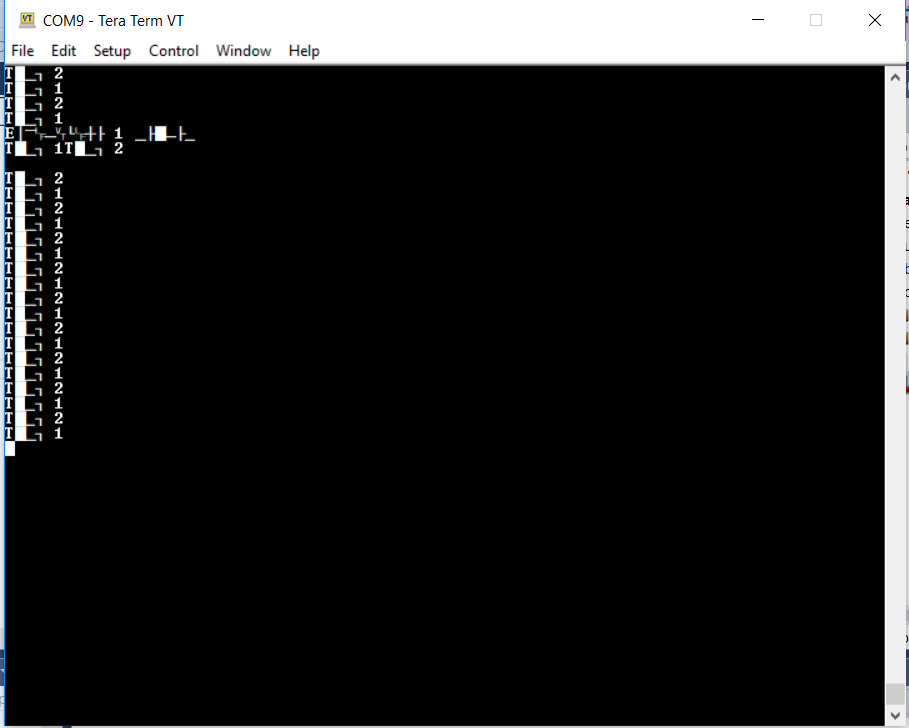
1. **Screen shots of the program execution results or debug windows of Keil µVision**



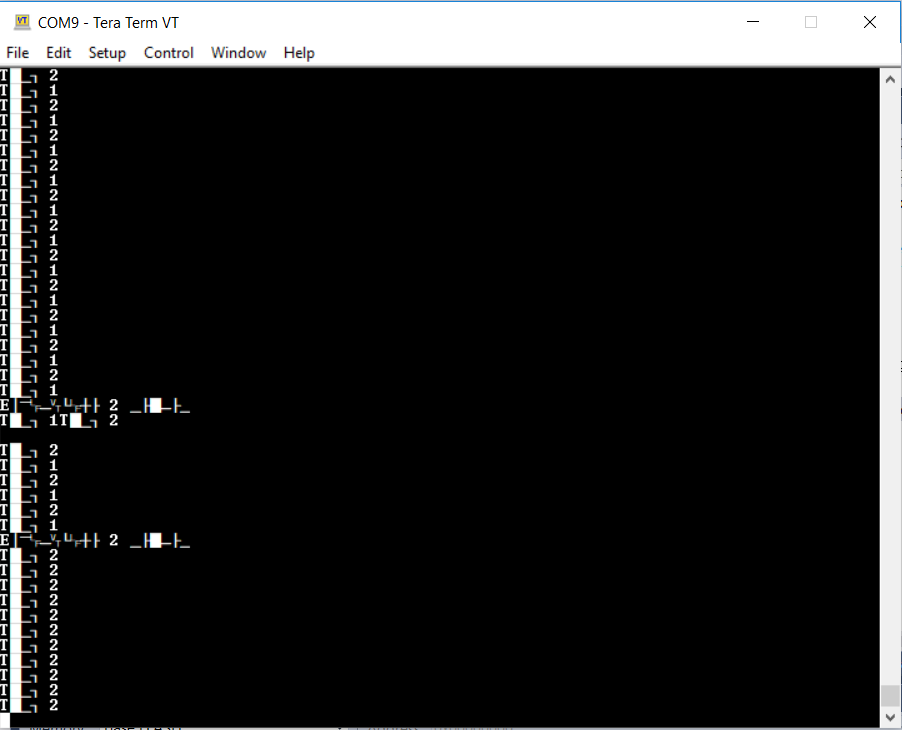
Both tasks are rapidly entering and exiting the running state, but only one task can run at any one time.

.

The output is still same by creating task1 from main(), then creating task2 from within task1.



It shows the redution of duplication by creating two instances of a single task implementation vTaskFunction. Both tasks entered and exited the running state in turn due to the same priority. It has a delay as well by creating the null for loop.



It shows the delay because of the null for loop. But changing the priority of each task causes the scheduler selects the same task. Task 2 is always in running state and task 1 is always in the not running state.

1. **Conclusion**

I have learnt lots of knowledge during my first FreeRTOS task. Not only do I know how to use the Atmel Studio by building the project and configuring the serial port, but also I have familiar with some basic implemented functions, especially the xTaskCreate() function. It is the most fundamental component in a multitasking system. And it is very flexible of this function to change the type of parameters. It is very clear to see how different priorities tasks work by running the project. From the overall view of programming, I noticed it is important to declare lobal variables as well, which can make sure each created instance will execute independently under the control of the freertos scheduler.

1. **Appendix: The source code (main.c) with sufficient comments.**

Example1

/\*\*

\* \mainpage User Application template doxygen documentation

\*

\* \par Empty user application template

\*

\* Bare minimum empty user application template

\*

\* \par Content

\*

\* -# Include the ASF header files (through asf.h)

\* -# "Insert system clock initialization code here" comment

\* -# Minimal main function that starts with a call to board\_init()

\* -# "Insert application code here" comment

\*

\*/

/\* Include header files for all drivers that have been imported from

\* Atmel Software Framework (ASF).

\*/

#include <asf.h>

void vApplicationIdleHook()

{

while(1);

}

/\*\* UART module for debug. \*/

static struct usart\_module cdc\_uart\_module;

/\*\*

\* \brief Configure UART console.

\*/

static void configure\_console(void)

{

struct usart\_config usart\_conf;

usart\_get\_config\_defaults(&usart\_conf);

usart\_conf.mux\_setting = EDBG\_CDC\_SERCOM\_MUX\_SETTING;

usart\_conf.pinmux\_pad0 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD0;

usart\_conf.pinmux\_pad1 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD1;

usart\_conf.pinmux\_pad2 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD2;

usart\_conf.pinmux\_pad3 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD3;

usart\_conf.baudrate = 115200;

stdio\_serial\_init(&cdc\_uart\_module, EDBG\_CDC\_MODULE, &usart\_conf);

usart\_enable(&cdc\_uart\_module);

}

/\* Used as a loop counter to create a very crude delay. \*/

#define mainDELAY\_LOOP\_COUNT ( 0xfffff )//( 0xffffff )

/\* The task functions. \*/

static void vTask1( void \*pvParameters );

static void vTask2( void \*pvParameters );

/\*-----------------------------------------------------------\*/

int main( void )

{

system\_init();

configure\_console();

printf("Experiment 1 starts\r\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. \*/

200, /\* Stack depth in words. \*/

NULL, /\* We are not using the task parameter. \*/

1, /\* This task will run at priority 1. \*/

NULL ); /\* We are not using the task handle. \*/

xTaskCreate(vTask2, "Task 2", 200, NULL, 1, NULL);

/\* Start the scheduler so our tasks start executing. \*/

vTaskStartScheduler();

/\* If all is well we will never reach here as the scheduler will now be

running. If we do reach here then it is likely that there was insufficient

heap available for the idle task to be created. \*/

for( ;; );

}

/\*-----------------------------------------------------------\*/

void vTask1( void \*pvParameters )

{

const char \*pcTaskName = "Task 1\r\n";

volatile unsigned long ul;

/\*To-do:

Move the commented task2 creation statement from the main() to here.\*/

/\* As per most tasks, this task is implemented in an infinite loop. \*/

for( ;; )

{

printf( pcTaskName ); // Print out the name of this task.

/\* Delay for a period. \*/

for( ul = 0; ul < mainDELAY\_LOOP\_COUNT; ul++ )

{

/\* This loop is just a very crude delay implementation. There is

nothing to do in here. Later exercises will replace this crude

loop with a proper delay/sleep function. \*/

}

}

}

/\*-----------------------------------------------------------\*/

void vTask2( void \*pvParameters )

{

const char \*pcTaskName = "Task 2\r\n";

volatile unsigned long ul;

/\* As per most tasks, this task is implemented in an infinite loop. \*/

for( ;; )

{

printf( pcTaskName );// print out the name of this task

/\* Delay for a period. \*/

for( ul = 0; ul < mainDELAY\_LOOP\_COUNT; ul++ )

{

/\* This loop is just a very crude delay implementation. There is

nothing to do in here. Later exercises will replace this crude

loop with a proper delay/sleep function. \*/

}

}

}

Example 2

/\*\*

\* \mainpage User Application template doxygen documentation

\*

\* \par Empty user application template

\*

\* Bare minimum empty user application template

\*

\* \par Content

\*

\* -# Include the ASF header files (through asf.h)

\* -# "Insert system clock initialization code here" comment

\* -# Minimal main function that starts with a call to board\_init()

\* -# "Insert application code here" comment

\*

\*/

/\*

\* Include header files for all drivers that have been imported from

\* Atmel Software Framework (ASF).

\*/

#include <asf.h>

void vApplicationIdleHook()

{

while(1);

}

/\*\* UART module for debug. \*/

static struct usart\_module cdc\_uart\_module;

/\*\*

\* \brief Configure UART console.

\*/

static void configure\_console(void)

{

struct usart\_config usart\_conf;

usart\_get\_config\_defaults(&usart\_conf);

usart\_conf.mux\_setting = EDBG\_CDC\_SERCOM\_MUX\_SETTING;

usart\_conf.pinmux\_pad0 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD0;

usart\_conf.pinmux\_pad1 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD1;

usart\_conf.pinmux\_pad2 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD2;

usart\_conf.pinmux\_pad3 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD3;

usart\_conf.baudrate = 115200;

stdio\_serial\_init(&cdc\_uart\_module, EDBG\_CDC\_MODULE, &usart\_conf);

usart\_enable(&cdc\_uart\_module);

}

/\* Used as a loop counter to create a very crude delay. \*/

#define mainDELAY\_LOOP\_COUNT ( 0x11ffff )//( 0xffffff )

static void vTaskFunction( void \*pvParameters );

// DECLARE the prototype of a task function "vTaskFunction". This function is defined after the "main" function.

const char \*pcTextForTask1 = "Task 1 \r\n";

// DECLARE a global constant string variable "pcTextForTask1" that will be passed in as the task parameter and initialize it as "Task 1 \r\n".

const char \*pcTextForTask2 = "Task 2 \r\n";

//DECLARE another global constant string variable "pcTextForTask2" that will also be passed in as the task parameter and initialize it as "Task 2 \r\n".

/\*-----------------------------------------------------------\*/

int main( void )

{

system\_init();

configure\_console();

printf("Experiment 2 starts\r\n");

xTaskCreate( vTaskFunction, "Task 1", 200, (void \*)pcTextForTask1, 1, NULL );

xTaskCreate( vTaskFunction, "Task 2", 200, (void \*)pcTextForTask2, 2, NULL );

// change the priority of Task 2 from 1 to 2

/\* Start the scheduler so our tasks start executing. \*/

xTaskStartScheduler();

/\* If all is well we will never reach here as the scheduler will now be

running. If we do reach here then it is likely that there was insufficient

heap available for the idle task to be created. \*/

for( ;; );

}

/\*-----------------------------------------------------------\*/

static void vTaskFunction( void \*pvParameters )

{

char \*pcTaskName;

// Declare a string variable "pcTaskName" WITHOUT initialization

volatile unsigned long ul;

// The string to print out is passed in via the parameter.

pcTaskName = (char \*)pvParameters;

//Cast the input parameter "pvParameters" to character pointer (i.e., char \*) and assign it to the variable "pcTaskName".

/\* As per most tasks, this task is implemented in an infinite loop

which is exactly SAME as the loop body of vTask1 in Example 1.\*/

for( ;; )

{

printf(pcTaskName);// print the name of this task

/\* Delay for a period. \*/

for( ul = 0; ul < mainDELAY\_LOOP\_COUNT; ul++ )

{

/\* This loop is just a very crude delay implementation. There is

nothing to do in here. Later exercises will replace this crude

loop with a proper delay/sleep function. \*/

}

}

}

Example 3

Example 2

/\*\*

\* \mainpage User Application template doxygen documentation

\*

\* \par Empty user application template

\*

\* Bare minimum empty user application template

\*

\* \par Content

\*

\* -# Include the ASF header files (through asf.h)

\* -# "Insert system clock initialization code here" comment

\* -# Minimal main function that starts with a call to board\_init()

\* -# "Insert application code here" comment

\*

\*/

/\*

\* Include header files for all drivers that have been imported from

\* Atmel Software Framework (ASF).

\*/

#include <asf.h>

void vApplicationIdleHook()

{

while(1);

}

/\*\* UART module for debug. \*/

static struct usart\_module cdc\_uart\_module;

/\*\*

\* \brief Configure UART console.

\*/

static void configure\_console(void)

{

struct usart\_config usart\_conf;

usart\_get\_config\_defaults(&usart\_conf);

usart\_conf.mux\_setting = EDBG\_CDC\_SERCOM\_MUX\_SETTING;

usart\_conf.pinmux\_pad0 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD0;

usart\_conf.pinmux\_pad1 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD1;

usart\_conf.pinmux\_pad2 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD2;

usart\_conf.pinmux\_pad3 = EDBG\_CDC\_SERCOM\_PINMUX\_PAD3;

usart\_conf.baudrate = 115200;

stdio\_serial\_init(&cdc\_uart\_module, EDBG\_CDC\_MODULE, &usart\_conf);

usart\_enable(&cdc\_uart\_module);

}

/\* Used as a loop counter to create a very crude delay. \*/

#define mainDELAY\_LOOP\_COUNT ( 0x11ffff )//( 0xffffff )

static void vTaskFunction( void \*pvParameters );

// DECLARE the prototype of a task function "vTaskFunction". This function is defined after the "main" function.

const char \*pcTextForTask1 = "Task 1 \r\n";

// DECLARE a global constant string variable "pcTextForTask1" that will be passed in as the task parameter and initialize it as "Task 1 \r\n".

const char \*pcTextForTask2 = "Task 2 \r\n";

//DECLARE another global constant string variable "pcTextForTask2" that will also be passed in as the task parameter and initialize it as "Task 2 \r\n".

/\*-----------------------------------------------------------\*/

int main( void )

{

system\_init();

configure\_console();

printf("Experiment 2 starts\r\n");

xTaskCreate( vTaskFunction, "Task 1", 200, (void \*)pcTextForTask1, 1, NULL );

xTaskCreate( vTaskFunction, "Task 2", 200, (void \*)pcTextForTask2, 1, NULL );

/\* Start the scheduler so our tasks start executing. \*/

xTaskStartScheduler();

/\* If all is well we will never reach here as the scheduler will now be

running. If we do reach here then it is likely that there was insufficient

heap available for the idle task to be created. \*/

for( ;; );

}

/\*-----------------------------------------------------------\*/

static void vTaskFunction( void \*pvParameters )

{

char \*pcTaskName;

// Declare a string variable "pcTaskName" WITHOUT initialization

volatile unsigned long ul;

// The string to print out is passed in via the parameter.

pcTaskName = (char \*)pvParameters;

//Cast the input parameter "pvParameters" to character pointer (i.e., char \*) and assign it to the variable "pcTaskName".

/\* As per most tasks, this task is implemented in an infinite loop

which is exactly SAME as the loop body of vTask1 in Example 1.\*/

for( ;; )

{

printf(pcTaskName);// print the name of this task

/\* Delay for a period. \*/

for( ul = 0; ul < mainDELAY\_LOOP\_COUNT; ul++ )

{

/\* This loop is just a very crude delay implementation. There is

nothing to do in here. Later exercises will replace this crude

loop with a proper delay/sleep function. \*/

}

}

}