|  |
| --- |
|  |
| **EE489 / EE589 Real-Time Embedded Systems Design** |
| Lab #3  *Project Group member names* |
|  |
| *2/7/2018* |

|  |
| --- |
|  |

# Introduction

An idle task is automatically created by the scheduler when *vTaskStartScheduler*() is called. It executes whenever there are no other application tasks ready to run. It also has the lowest possible priority (zero) to make sure the next task which has higher priority pre-empts the idle task entering the Running state. We add application specific functionality directly into the idle task by using an idle task hook function vApplicationIdleHook() which can be called automatically by the idle task once per iteration of the idle task loop. And then it can execute low priority, perform background, measure spare processing capacity, simply to place the processor into a low-power mode.

The scheduler will always select the highest Ready state task as the task to enter the Running state. We can use the *vTaskPrioritySet()* API function to change the priority of two tasks relative to each other. This function has two parameters pxTask which is the handle of the task whose priority is being modified and uxNewPriority which is a priority can be set of a subject task. And then we call the function *uxTaskPriorityGet (xTaskHandle pxTask*) to query the priority of a specific task. So we can demonstrate two tasks at different priorities.

The third part of this lab shows how to delete tasks by using the vTaskDelete(xTaskHandle pxTaskToDelete) API function which can make a task delete itself or any other task.

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

void vApplicationIdleHook();

First, declare a global variable ulIdleCycleCount with the unsigned long type, and initialize it as the value 0UL. Then, increment it by 1 within the vApplicationIdleHook() function.

unsigned long ulIdleCycleCount=0UL;

void vApplicationIdleHook()

{

ulIdleCycleCount++;

}

After creating tasks, we call the vPrintStringAndNumber() library function, and pass it with two parameters: the name of this task and the number of ulIdleCycleCount to print the both a string variable (pcTaskName) and a number variable (ulIdleCycleCount) at the same time.

vPrintStringAndNumber(pcTaskName,ulIdleCycleCount);

Idle task will execute only when no other tasks are running and idle task hook must never attempt to block or suspend. When the application uses the vTaskDelete function, the idle task hook must always return to its caller within a reasonable period to clean up the kernel resources after a task has been deleted.

Declare two global variables xTask1Handle and xTask2Handle with the type xTaskHandle to hold the handle of vTask1 and vTask2 respectively.

xTaskHandle xTask1Handle;

xTaskHandle xTask2Handle;

And when create the tasks in the main function, set the task handle as &xTask1Handle and &xTask2Handle and set them at different priorities.

xTaskCreate(vTask1,"Task1",200,NULL,2,&xTask1Handle);

xTaskCreate(vTask2,"Task2",200,NULL,1,&xTask2Handle);

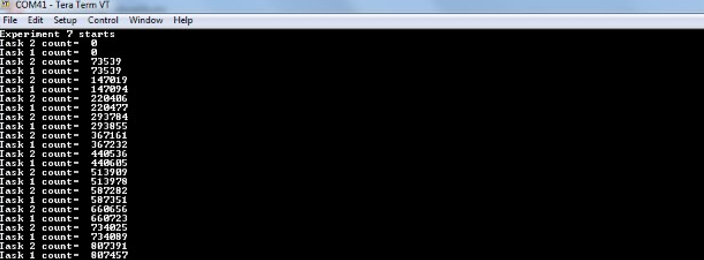
Declare the local variable uxPriority, then call uxTaskPriorityGet() to access its own priority.

unsigned portBASE\_TYPE uxPriority; uxPriority=uxTaskPriorityGet(NULL);

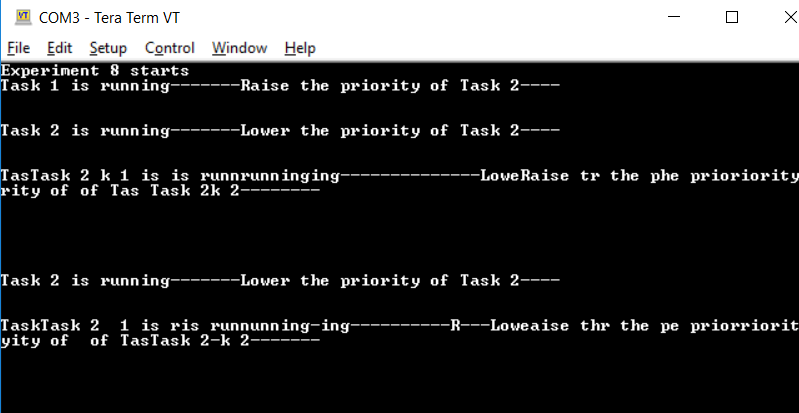
By using the vTaskPrioritySet() API function to change the priority of two tasks relative to each other. To raise the priority of vTask2(): vTaskPrioritySet(xTask2Handle,(uxPriority+1)); To lower the priority of vTask2() itself: vTaskPrioritySet(NULL,(uxPriority-1));

Call the vTaskDelete() function either using a NULL parameter or the handle of xTask2 vTask2Handle: vTaskDelete(xTask2Handle);

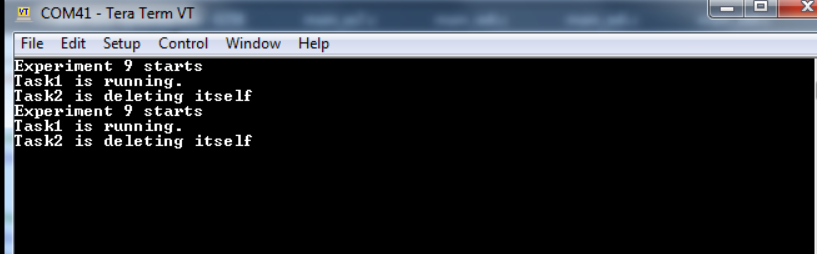
1. **Screen shots of the program execution results or debug windows of Keil µVision**



The output produced by Example 7 is shown below and shows that (on my computer) the idle task hook function is called very approximately 73500 times between each iteration of the application tasks.



Task1 runs first and prints out the strings due to the higher priority. Task2 starts to run as it has the highest relative priority.



Task1 runs at first by created by the main function at priority 1.Then the task2 is created at priority 2. So task 2 runs and does nothing but delete itself.

1. **Conclusion**

Idle task and Idle Task Hook have the lowest priority and call the idle task hook function to make sure there is always a task entering running state. And the scheduler always selects the task with highest priority, by calling vTaskPrioritySet function to change the priority of two tasks relative to each other. And both task either are in either Ready or Running state. Deleted tasks no longer exist and cannot enter the Running state again by calling vTaskDelete() API function. We must note that do not completely starve the idle task of all processing time and any memory or other resource that the application task allocates itself must by be freed explicitly by the application code.

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

EX7/\*\*

\* \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 vPrintStringAndNumber( const char \*pcString, unsigned long ulValue )

{

/\* Print the string, suspending the scheduler as method of mutual

exclusion. \*/

vTaskSuspendAll();

{

printf( "%s %u\r\n", pcString, ulValue );

}

xTaskResumeAll();

}

/\* A variable that is incremented by the idle task hook function. \*/

/\*To-do:

Declare a global variable "ulIdleCycleCount" with the unsigned long type.

And initialize it with the value 0UL. \*/

unsigned long ulIdleCycleCount=0UL;

void vApplicationIdleHook()

{

/\* This hook function does nothing but increment a counter. \*/

/\*To-do:

Only perform the function of incrementing the variable ulIdleCycleCount by 1. \*/

ulIdleCycleCount++;

}

/\*\* 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 )

/\* The task function. \*/

static void vTaskFunction( void \*pvParameters );

/\* Define the strings that will be passed in as the task parameters. These are

defined const and off the stack to ensure they remain valid when the tasks are

executing. \*/

const char \*pcTextForTask1 = "Task 1 count= ";

const char \*pcTextForTask2 = "Task 2 count= ";

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

int main( void )

{

system\_init();

configure\_console();

printf("Experiment 7 starts\r\n");

/\* Create the first task at priority 1... \*/

xTaskCreate( vTaskFunction, "Task 1", 240, (void\*)pcTextForTask1, 1, NULL );

/\* ... and the second task at priority 2. The priority is the second to

last parameter. \*/

xTaskCreate( vTaskFunction, "Task 2", 240, (void\*)pcTextForTask2, 2, 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( ;; );

}

static void vTaskFunction( void \*pvParameters )

{

char \*pcTaskName;

/\* The string to print out is passed in via the parameter. Cast this to a

character pointer. \*/

pcTaskName = ( char \* ) pvParameters;

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

for( ;; )

{

/\*To-do:

Call the vPrintStringAndNumber() library function, and pass it with

two parameters: the name of this task AND the number of ulIdleCycleCount. \*/

vPrintStringAndNumber(pcTaskName,ulIdleCycleCount);

/\* Delay for a period. This time we use a call to vTaskDelay() which

puts the task into the Blocked state until the delay period has expired.

The delay period is specified in 'ticks'. \*/

vTaskDelay( 250 / portTICK\_RATE\_MS );

}

}

EX8

/\*\*

\* \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 )

/\* The task functions. \*/

static void vTask1( void \*pvParameters );

static void vTask2( void \*pvParameters );

/\* Used to hold the handle of Task2. \*/

/\*To-do:

Declare two global variables xTask1Handle and xTask2Handle with the type

xTaskHandle to hold the handle of vTask1 and vTask2 respectively.\*/

xTaskHandle xTask1Handle;

xTaskHandle xTask2Handle;

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

int main( void )

{

system\_init();

configure\_console();

printf("Experiment 8 starts\r\n");

/\* To-do:

Create the task instance Task1 of vTask1 at priority 2.

The last parameter is set as &xTask1Hanle to obtain a handle to the

created task1 \*/

xTaskCreate(vTask1,"Task1",200,NULL,2,&xTask1Handle);

/\* To-do:

Create the task instance Task2 of vTask2 at priority 1 - which is lower

than the priority of Task1. The last parameter is set as &xTask2Hanle to

obtain a handle to the created task2 \*/

xTaskCreate(vTask2,"Task2",200,NULL,1,&xTask2Handle);

/\* 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( ;; );

}

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

static void vTask1( void \*pvParameters )

{

volatile unsigned long ul;

/\*To-do:

Declare a local variable uxPriority with the type unsigned portBASE\_TYPE \*/

unsigned portBASE\_TYPE uxPriority;

/\* This task will always run before Task2 as it has the higher priority.

Neither Task1 nor Task2 ever block so both will always be in either the

Running or the Ready state.

Query the priority at which this task is running - passing in NULL means

"return our own priority". \*/

/\*To-do:

Assign the variable uxPriority with the priority of Task 1 itself,

which is returned from the function uxTaskPriorityGet().

The parameter passed to this function is NULL. \*/

uxPriority=uxTaskPriorityGet(NULL);

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

for( ;; )

{

/\* Print out the name of this task. \*/

printf( "Task 1 is running-------Raise the priority of Task 2----\r\n\r\n" );

printf( "\r\n" );

/\* 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. \*/

}

/\* Setting the Task2 priority above the Task1 priority will cause

Task2 to immediately start running (as then Task2 will have the higher

priority among two created tasks). \*/

/\*To-do:

Call the function vTaskPrioritySet() with two paramters. The first one

is the handle of task 2; the second one is uxPriority + 1. \*/

vTaskPrioritySet(xTask2Handle,(uxPriority+1));

/\* Task1 will only run when it has a priority higher than Task2.

Therefore, for this task to reach this point Task2 must already have

executed and set its priority back down to lower than Task1. \*/

}

}

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

static void vTask2( void \*pvParameters )

{

unsigned portBASE\_TYPE uxPriority;

volatile unsigned long ul;

/\*To-do:

Assign uxPriority with the priority of Task 1, which is returned from

uxTaskPriorityGet(). The parameter passed to this function is the

handle to the created task1. \*/

uxPriority =uxTaskPriorityGet(NULL);

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

for( ;; )

{

/\* Print out the name of this task. \*/

printf( "Task 2 is running-------Lower the priority of Task 2----\r\n\r\n" );

printf( "\r\n" );

/\* 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. \*/

}

/\* Set our priority back down to its original value. Passing in NULL

as the task handle means "change our own priority". Setting the

priority below that of Task1 will cause Task1 to immediately start

running again. \*/

/\*To-do:

Call the function vTaskPrioritySet() with two parameters. The first one

is the handle of task 2 itself which is NULL; the second one could be

uxPriority - 1. \*/

vTaskPrioritySet(NULL,(uxPriority-1));

}

}

EX9

/\*\*

\* \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 )

/\* The two task functions. \*/

static void vTask1( void \*pvParameters );

static void vTask2( void \*pvParameters );

/\* To-do:

Create a global variable xTask2Handle with the type xTaskHandle.

It is used to hold the handle of Task2. \*/

xTaskHandle xTask2Handle;

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

int main( void )

{

system\_init();

configure\_console();

printf("Experiment 9 starts\r\n");

/\* To-do:

Create the first task instance of vTask1 at priority 1. The

task parameter (4th parameter) is not used and is set to NULL.

The task handle (last parameter) is also not used and set to NULL. \*/

xTaskCreate(vTask1,"Task 1",240,NULL,1,NULL);

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

vTaskStartScheduler();

for( ;; );

}

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

static void vTask1( void \*pvParameters )

{

const portTickType xDelay100ms = 100 / portTICK\_RATE\_MS;

for( ;; )

{

/\* Print out the name of this task. \*/

printf( "Task1 is running.\r\n" );

/\* To-do:

Create task 2 at a higher priority (i.e., 2).

Again the task parameter is not used, so is set to NULL. BUT we want to

obtain a handle to the task, so pass in the address of the xTask2Handle variable

as the last parameter of xTaskCreate() API function. \*/

xTaskCreate(vTask2,"Task 2",240,NULL,2,&xTask2Handle);

/\* Task2 has/had the higher priority, so for Task1 to reach here Task2

must have already executed and deleted itself. Delay for 100ms. \*/

vTaskDelay( xDelay100ms );

}

}

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

static void vTask2( void \*pvParameters )

{

/\* Task2 does nothing but delete itself. \*/

printf( "Task2 is deleting itself\r\n" );

/\*To-do:

Call the vTaskDelete() function either using a NULL parameter

or the handle of xTask2 vTask2Handle. \*/

vTaskDelete(NULL);

}