|  |
| --- |
|  |
| **EE489 / EE589 Real-Time Embedded Systems Design** |
| Lab #2  *Dianzhi Yu* |
|  |
| *1/26/2018* |

|  |
| --- |
|  |

# Introduction

We design the delay instructions to make the periodic tasks remain in the ready state by calling the vTaskDelay(portTickTYpe xTicksToDelay) library function which replaces the former null loop. The input parameter xTicksToDelay is the number of tick interrupts that the calling task should remain in the blocked state before being transitioned back into the Ready state. For example, if a task called vTaskDelay(100) while the tick count was 10,000, then it would immediately enter the blocked state and remain there until the tick count reached 10, 100.

But it has a potential problem by using the vTaskDelay() library function because this function does not make sure the frequency at which the tasks run is fixed. We use the vTaskDelayUntil() library function to solve the problem.

At last the example, we design an execution sequence to make tasks at different priorities. The task1 and task2 at same priority and print out the string continuously. The task3 is at priority 2 to print out the string periodically. We combine some functions at the last examples to observe the output.

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

vTaskDelay(250/portTICK\_RATE\_MS);

The input parameter is the number of tick interrupts that calling task should remain in the blocked state before being transitioned back into the ready state. So “250/portTICK\_RATE\_MS” can convert milliseconds into ticks.

vTaskDelayUntil(xLastWakeTime, xTimeIncrement);

We call this function to replace vTaskDelay() to generate delay. This function can be used on the condition that the task is periodical with a fixed frequency. And both of these parameters is defined as portTickType. First, we have the only chance to initialize the variable xLastWakeTime with the current tick count, because this parameter is managed by the function automatically. Then the second parameter is specified in ‘ticks’, so it is set as 250/portTICK\_RATE\_MS.

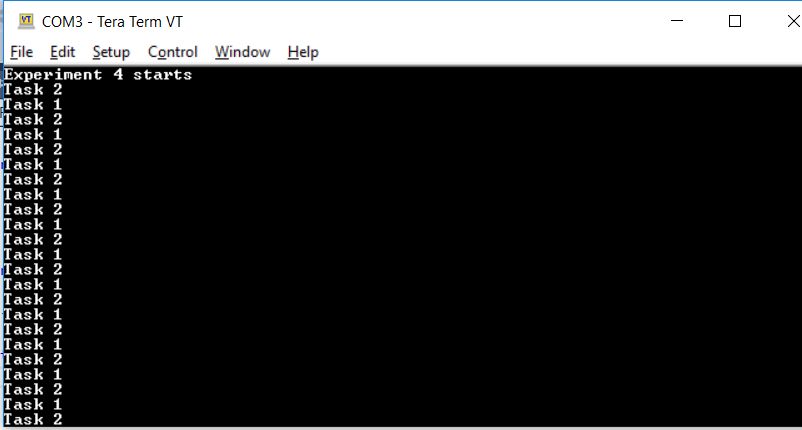
vContinuousProcessingTask(void \*pvParameters);

Declare the prototype of the task function vContinuousProcessingTask, then call the xTaskCreate function to create task1 and task2 at priority 1.

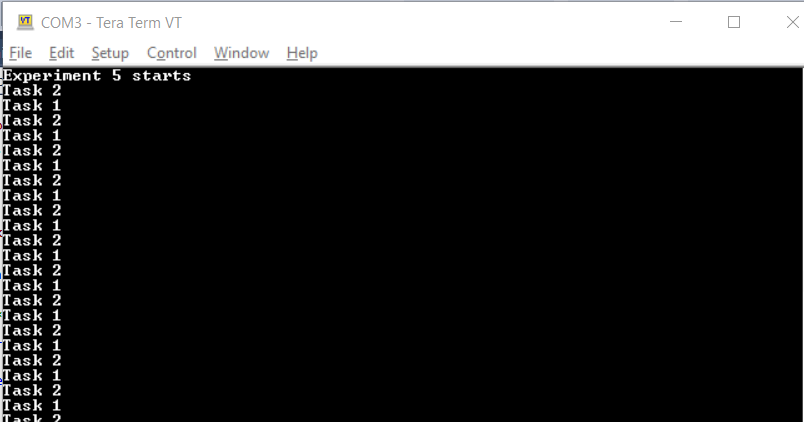
vPeriodicTask(void \*pvParameters);

Create the task3 as a periodic task and then call the vTaskDelay function to place itself into the blocked state between each print iteration.

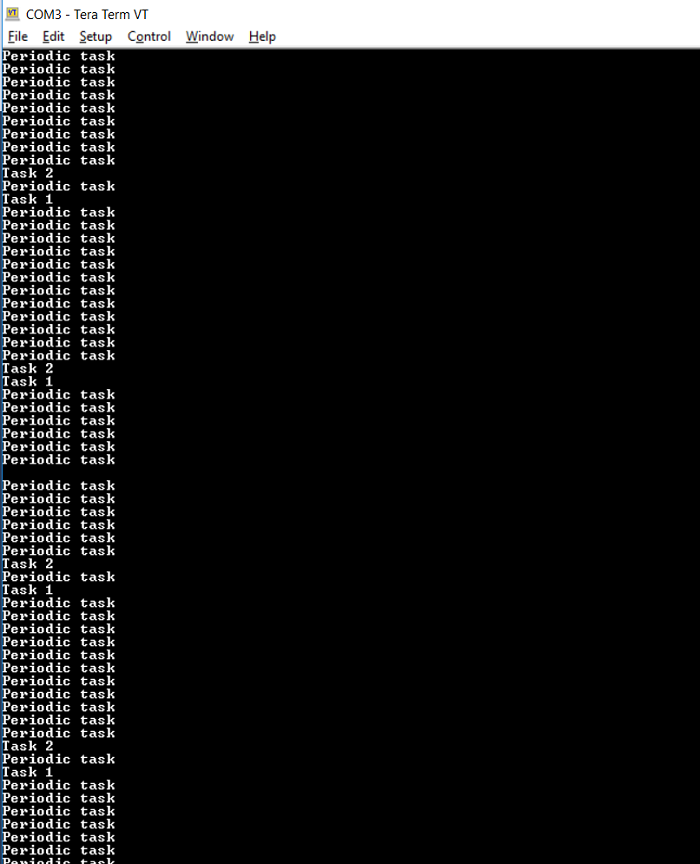
1. **Screen shots of the program execution results or debug windows of Keil µVision**



Task2 has higher priority than task1 so runs first. It prints out its string then calls vTaskDelay and in so doing enters the blocked state, which allows the task1 to execute. And then task1 prints out its string. It also enters the blocked state by calling vTaskDelay.



These two tasks use the vTaskDelayUntil when the tasks enter the blocked state to ensure that the frequency at which they run is fixed. So the result shows same as the output from the last one.



The periodic task is the highest priority task so it enters the running state immediately. When the scheduler chooses a task into a running state, the periodic task will enter the blocked state. The continuous task 1 runs for a complete tick period and it could print out its string many times. Then the tick interrupt occurs during the scheduler select a new task to run. As both continuous tasks have same priority, both are able to run. So continuous task 2 can also enter the running state when it remains for the entire tick period because both tasks which are at same priority can run the scheduler.

1. **Conclusion**

The vTaskDelay function uses the blocked state to create a delay to ensure all tasks can remain in the ready state. And the blocked state task doesn’t use any processing time, so processing time is consumed only when there is a work to be done. Each time when the tasks leave the blocked state, they execute for a fraction of a tick period before reentering the blocked state. Then convert the example task to use vTaskDelayUntil function to in case that the tasks are not periodic. And combining the blocking and non-blocking tasks will be more flexible based on the former examples.

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

EX4:/\*\*

\* \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 )

/\* To-do:

DECLARE the prototype of a task function "vTaskFunction". This function is

defined after the "main" function.

\*/

static void vTaskFunction( void \*pvParameters );

/\* To-do:

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

/\* To-do:

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". \*/

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

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

int main( void )

{

system\_init();

configure\_console();

printf("Experiment 4 starts\r\n");

/\* To-do:

Create the first instance of the task "vTaskFunction" by modifying the 1th parameter to "vTaskFunction" and passing

"pcTextForTask1" to the task instance in the 4th parameter, other parameters are same as lab1 \*/

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

/\* To-do:

Create the second task instance in the same way. but passing

a different value as the 4th parameter "pcTextForTask2". \*/

xTaskCreate( vTaskFunction, "Task 2", 200, (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 )

{

/\* To-do:

Declare a string variable "pcTaskName" WITHOUT initialization. \*/

char \*pcTaskName;

volatile unsigned long ul;

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

/\* To-do:

Cast the input parameter "pvParameters" to character pointer (i.e., char \*)

and assign it to the variable "pcTaskName". \*/

pcTaskName = (char \*)pvParameters;

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

{

vTaskDelay(250/portTICK\_RATE\_MS);

/\* To-do:

Call printf() function to display the name of this task at the terminal. \*/

printf(pcTaskName);

/\* 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.

}

\*/

}

}

EX5

/\*\*

\* \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 )

/\* Declare the task function prototype. \*/

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 \r\n";

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

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

int main( void )

{

system\_init();

configure\_console();

printf("Experiment 5 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;

/\* To-do:

Declare a variable named as xLastWakeTime with the type portTickType.

\*/

portTickType xLastWakeTime ;

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

character pointer. \*/

pcTaskName = ( char \* ) pvParameters;

/\* To-do:

Assign xLastWakeTime with the value returned from the library

function xTaskGetTickCount(). \*/

xLastWakeTime= xTaskGetTickCount();

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

for( ;; )

{

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

printf( pcTaskName );

/\* To-do:

Call the library function vTaskDelayUntil() to generate periodic delay.

It requires two parameters, 1th parameter is the pointer to the xLastWakeTime;

2nd parameter is the relative delay time which could be (250 / portTICK\_RATE\_MS).

\*/

vTaskDelayUntil(&xLastWakeTime, 250/portTICK\_RATE\_MS);

}

}

EX6

/\*\*

\* \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 )

/\* To-do:

Declare the prototype of the task function vContinuousProcessingTask \*/

static void vContinuousProcessingTask( void \*pvParameters );

/\* To-do:

Declare the prototype of the task function vPeriodicTask \*/

static void vPeriodicTask( 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 \r\n";

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

const char \*pcTextForPeriodicTask = "Periodic task \r\n";

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

int main( void )

{

system\_init();

configure\_console();

printf("Experiment 6 starts\r\n");

/\* To-do:

Create the first instance of the continuous task vContinuousProcessingTask

at priority 1. Pass the string pcTexForTask1 as the 4th argument \*/

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

/\* To-do:

Also create the second instance of the continuous task vContinuousProcessingTask

at priority 1. Pass the string pcTexForTask2 as the 4th argument \*/

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

/\* To-do:

Create one instance of the periodic task vPeriodicTask

at priority 2. Pass the string pcTexForPeriodicTask as the 4th argument \*/

xTaskCreate(vPeriodicTask, "Periodic Task 1", 200, (void \*)pcTextForPeriodicTask, 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 vContinuousProcessingTask( void \*pvParameters )

{

/\* To-do:

Copy the function implementation of vTaskFunction() in Example 3 here. \*/

char \*pcTaskName;

volatile unsigned long ul;

/\* 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( ;; )

{

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

printf( pcTaskName );

/\* 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. \*/

}

}

}

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

static void vPeriodicTask( void \*pvParameters )

{

/\* To-do:

Copy the function implementation of vTaskFunction() in Example 5 here. \*/

char \*pcTaskName;

/\* To-do:

Declare a variable named as xLastWakeTime with the type portTickType. \*/

portTickType xLastWakeTime;

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

character pointer. \*/

pcTaskName = ( char \* ) pvParameters;

/\* To-do:

Assign xLastWakeTime with the value returned from the library

function xTaskGetTickCount(). \*/

xLastWakeTime = xTaskGetTickCount();

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

for( ;; )

{

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

printf( pcTaskName );

/\* To-do:

Call the library function vTaskDelayUntil() to generate periodic delay.

It requires two parameters, 1th parameter is the pointer to the xLastWakeTime;

2nd parameter is the relative delay time which could be (250 / portTICK\_RATE\_MS).

\*/

vTaskDelayUntil(&xLastWakeTime, (250 / portTICK\_RATE\_MS));

}

}