**Date Submitted: 10/30/2018**

**Assignment Youtube Playlist:** **https://www.youtube.com/playlist?list=PL4oTyvRrubXffjGq6W9WsJFOVMQHGAZCz**

# Task 01: Replacing the contents of hello.c

For Task 01, I have replaced the contents of hello.c with the provided code detailed below. No changes to XGCONF were made. As this assignment covers how to show stack usage and use ROV and RTOS Analyzer, those elements will be covered in the tasks relevant. Similarly, YouTube videos will be included for portions of this lab that involve physical reactions of the device or measurements, as many tasks are simply programming changes.

**Youtube Link:** **https://youtu.be/ZxyOo0O9L6s**

**Original Code (added comments):**

/\*

\* Copyright (c) 2015-2017, Texas Instruments Incorporated

\* 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 Texas Instruments Incorporated 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 THE COPYRIGHT OWNER OR

\* 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.

\*/

/\*

\* ======== hello.c ========

\*/

/\* TI-RTOS Header files \*/

**#include** <xdc/std.h>

**#include** <ti/sysbios/BIOS.h>

**#include** <ti/sysbios/knl/Task.h>

**#include** <ti/drivers/GPIO.h>

/\* Example/Board Header files \*/

**#include** "Board.h"

**void** **myDelay**(**int** count);

/\* Could be anything, like computing primes \*/

**#define** FakeBlockingSlowWork() myDelay(12000000)

**#define** FakeBlockingFastWork() myDelay(2000000)

Task\_Struct workTask;

/\* Make sure we have nice 8-byte alignment on the stack to avoid wasting memory \*/

**#pragma** DATA\_ALIGN(workTaskStack, 8)

**#define** STACKSIZE 1024

**static** uint8\_t workTaskStack[STACKSIZE];

**void** **doUrgentWork**(**void**)

{

**GPIO\_write**(Board\_GPIO\_LED1, Board\_GPIO\_LED\_OFF);

FakeBlockingFastWork(); /\* Pretend to do something useful but time-consuming \*/

**GPIO\_write**(Board\_GPIO\_LED1, Board\_GPIO\_LED\_ON);

}

**void** **doWork**(**void**)

{

**GPIO\_write**(Board\_GPIO\_LED0, Board\_GPIO\_LED\_OFF);

FakeBlockingSlowWork(); /\* Pretend to do something useful but time-consuming \*/

**GPIO\_write**(Board\_GPIO\_LED0, Board\_GPIO\_LED\_ON);

}

Void **workTaskFunc**(UArg arg0, UArg arg1)

{

**while** (1) {

/\* Do work \*/

doWork();

/\* Wait a while, because doWork should be a periodic thing, not continuous.\*/

myDelay(24000000);

}

}

/\*

\*

\*======== main ========

\*

\*/

**int** **main**(**void**)

{

Board\_initGeneral();

**GPIO\_init**();

/\* Set up the led task \*/

Task\_Params workTaskParams;

Task\_Params\_init(&workTaskParams);

workTaskParams.stackSize = STACKSIZE;

workTaskParams.priority = 2;

workTaskParams.stack = &workTaskStack;

Task\_construct(&workTask, workTaskFunc, &workTaskParams, NULL);

/\* Start kernel. \*/

BIOS\_start();

**return** (0);

}

/\*

\* ======== myDelay ========

\* Assembly function to delay. Decrements the count until it is zero

\* The exact duration depends on the processor speed.

\*/

**\_\_asm**(" .sect \".text:myDelay\"\n"

" .clink\n"

" .thumbfunc myDelay\n"

" .thumb\n"

" .global myDelay\n"

"myDelay:\n"

" subs r0, #1\n"

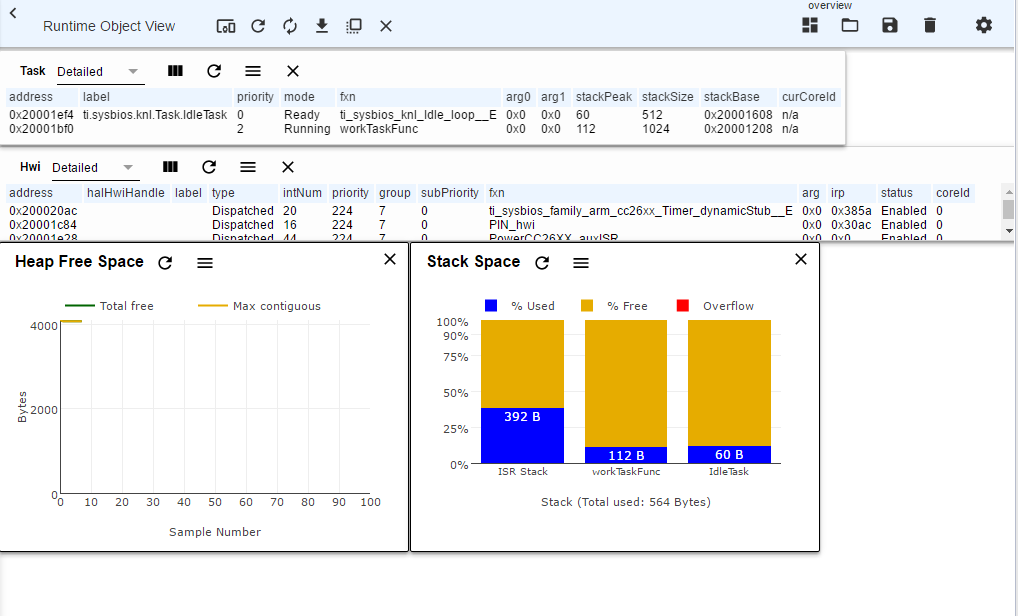
" bne.n myDelay\n"

" bx lr\n");

**-----------------------------------------------------------------------------------**

# Task 02: Debugging Tools

In Task 2 we are introduced to our debugging tools. First, we are shown the Runtime Object View which is a snapshot of the whole RTOS. This information is what results after running the program to the breakpoint a few times.



To access the Execution graph, we edit our configuration file, either graphically or through text. This is done through text by commenting out the “false” flag and un-commenting the “true” flag in the below code:

/\*

\* Enable logs in the BIOS library.

\*

\* Pick one:

\* - true (default)

\* Enables logs for debugging purposes.

\* - false

\* Disables logging for reduced code footprint and improved runtime

\* performance.

\*

\* When using BIOS in ROM:

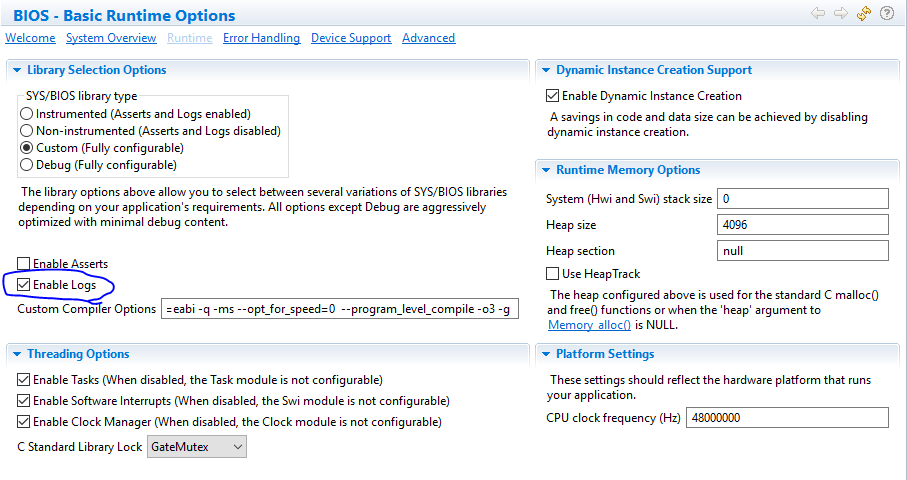
\* This option must be set to false.

\*/

BIOS.logsEnabled = **true**;

//BIOS.logsEnabled = false;

This can also be done graphically, as seen below.



For our CC1350 device, we will also need to use a kernel placed in flash rather than the ROM in order for logging to be enabled. To do this, we commend out the below code in our configuration file:

/\* ================ ROM configuration ================ \*/

/\*

\* To use BIOS in flash, comment out the code block below.

\*

var ROM = xdc.useModule('ti.sysbios.rom.ROM');

if (Program.cpu.deviceName.match(/CC2640R2F/)) {

ROM.romName = ROM.CC2640R2F;

}

else if (Program.cpu.deviceName.match(/CC26.2/)) {

ROM.romName = ROM.CC26X2;

}

else if (Program.cpu.deviceName.match(/CC13.2/)) {

ROM.romName = ROM.CC13X2;

}

else if (Program.cpu.deviceName.match(/CC26/)) {

ROM.romName = ROM.CC2650;

}

else if (Program.cpu.deviceName.match(/CC13/)) {

ROM.romName = ROM.CC1350;

}

\*/

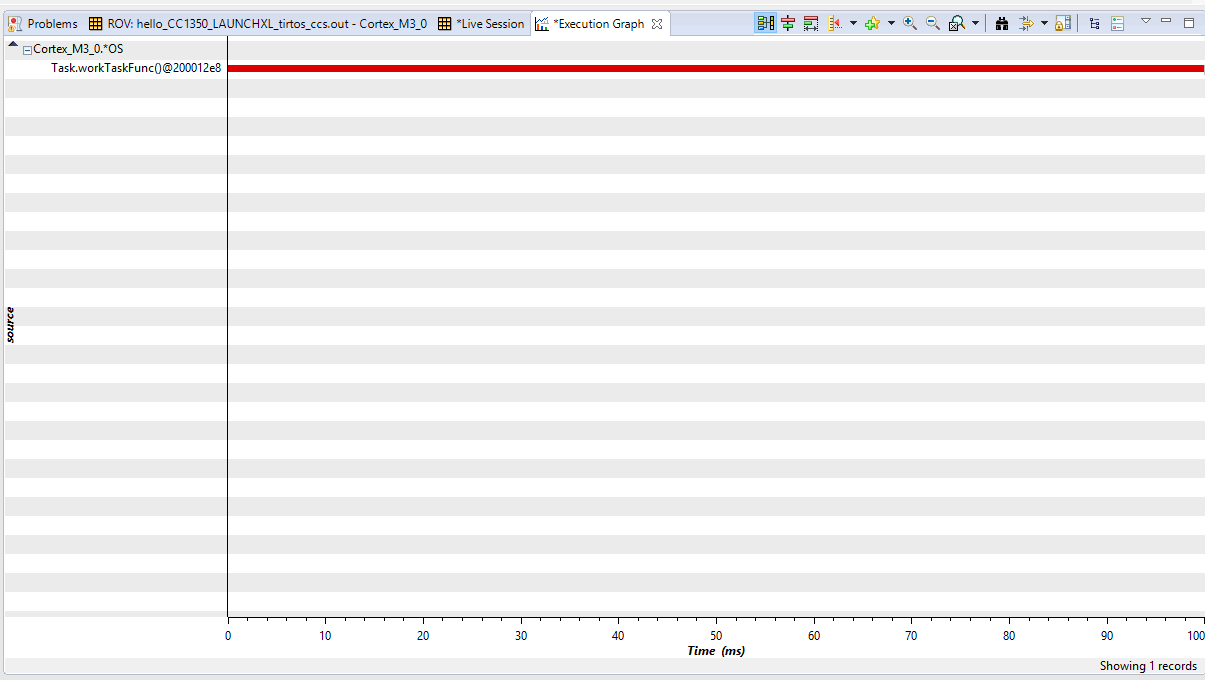
We also add the following code to the end of our configuration file to maintain buffers where the log files will reside.

**var LoggingSetup = xdc.useModule('ti.uia.sysbios.LoggingSetup');**

**LoggingSetup.sysbiosLoggerSize = 1024;**

**LoggingSetup.loadLogging = false;**

We then rerun the program without any breakpoints for approximately 15 seconds and then open our Execution Analysis tool.



The graph indicates that our work function has been executing 100% of the time, with no idle task running.

**-----------------------------------------------------------------------------------**

# Task 03: Sleeping Well

For Task 3 we investigate why our idle task is not shown running and find that our device is never actually going idle. This is due to our time-burning delay function not truly being idle, it is simply being busy doing “nothing”. In the next task we will implement a new idling function, to prepare we add the additional include to our program:

**#include** <ti/sysbios/knl/Clock.h>

In our configuration file we also find Clock.tickPeriod = 10, indicating a tick period of 10µs. We will use this in the next task to create a proper idle function.

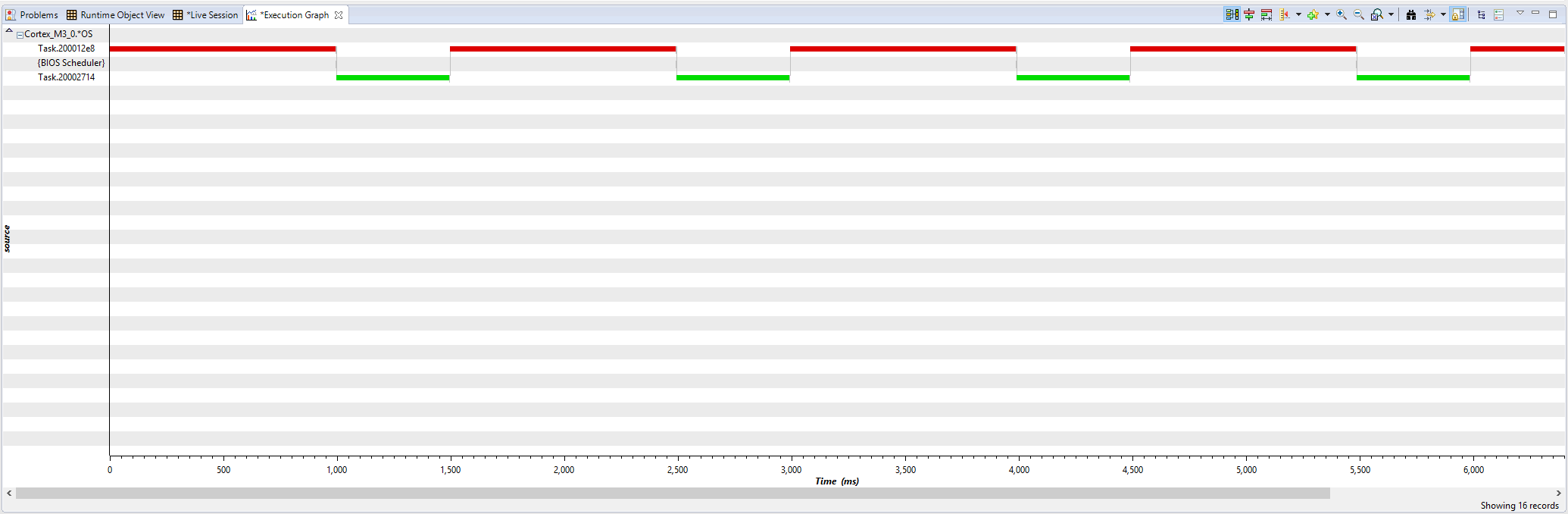
**-----------------------------------------------------------------------------------**

# Task 03.1

For Task 3.1 we replace the myDelay function with Task\_sleep to sleep our device for 500ms. As there are 1000µs/1ms, our new function will be:

Task\_sleep(500 \* ( 1000 / 10 )); //Sleep for 500ms, tick size 10µs.

Using a variable to store the tick size will be helpful for more complicated implementations. As we run the Execution Analysis tool again, we can see that the main process runs for 1000ms, then the idle process runs for 500ms as intended.



**-----------------------------------------------------------------------------------**

# Task 04: Executing urgent work

**Youtube Link:** **https://youtu.be/kHtu2tIY52Q**

For Task 4 we are going to modify our code to introduce a new task with a higher priority than our current task. We do so in the steps detailed below:

1. Copy the workTaskFunc function and rename it urgentWorkTaskFunc.
2. Have our new function call doUrgentWork.
3. Create a new task in the same fashion, named doUrgentWork.
4. Reduce the Task\_sleep() time to 50ms in urgentWorkTaskFunc.

Our resulting new code contains the following two portions. Note the doUrgentWork task already existed in the provided code:

**static** uint8\_t urgentTaskStack[STACKSIZE];

**void** **doUrgentWork**(**void**)

{

**GPIO\_write**(Board\_GPIO\_LED1, Board\_GPIO\_LED\_OFF);

FakeBlockingFastWork(); /\* Pretend to do something useful but time-consuming \*/

**GPIO\_write**(Board\_GPIO\_LED1, Board\_GPIO\_LED\_ON);

}

Void **urgentWorkTaskFunc**(UArg arg0, UArg arg1) //Our new, more-urgent task

{

**while** (1) {

/\* Do Urgent work \*/

doUrgentWork();

/\* Wait a while, because doUrgentWork should be a periodic thing, not continuous.\*/

//myDelay(24000000);

Task\_sleep(50 \* ( 1000 / 10 )); //Sleep for 50ms, tick size 10µs.

}

Task\_Params UrgentTaskParams;

Task\_Params\_init(&UrgentTaskParams);

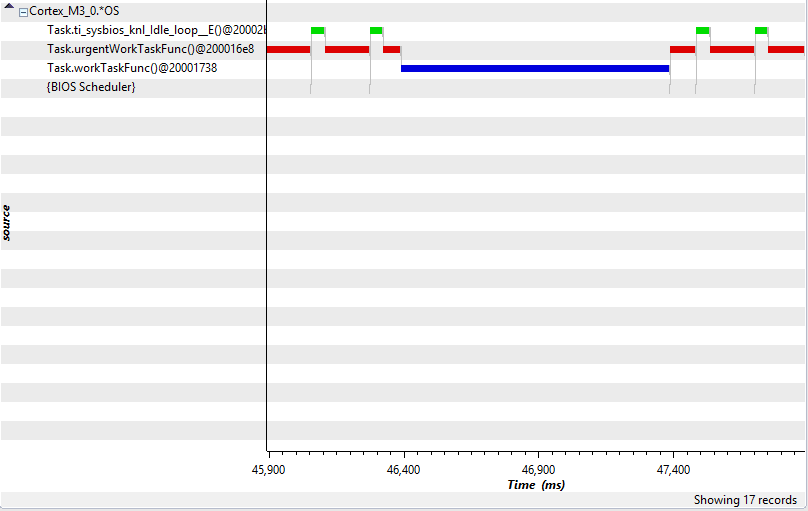
UrgentTaskParams.stackSize = STACKSIZE;

UrgentTaskParams.priority = 1;

UrgentTaskParams.stack = &urgentTaskStack;

Task\_construct(&urgentTask, urgentWorkTaskFunc, &UrgentTaskParams, NULL);

Executing our task, we see the following execution graph:



As we can see, LED 0 (red), which is our normal work function, is flashing at the desired rate while our urgent LED 1 (green) is not. This is due to priority 1 being lower than the priority 2 our standard function has. We modify the code shown below and see the resulting correct execution.

**Youtube Link:** **https://youtu.be/RNmES\_RAYr0**

UrgentTaskParams.priority = 3;

