**Date Submitted: 11/14/18 9:02 PM**

**Task 01:**

Youtube Link: <https://www.youtube.com/watch?v=upqy2P44ELY>

The purpose of Task 01 is to run the provided program to ensure to have a better understanding of the CC1350 Microcontroller. Task 01 also teaches how to debug the program as well.

Provided Code:

/\* 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"

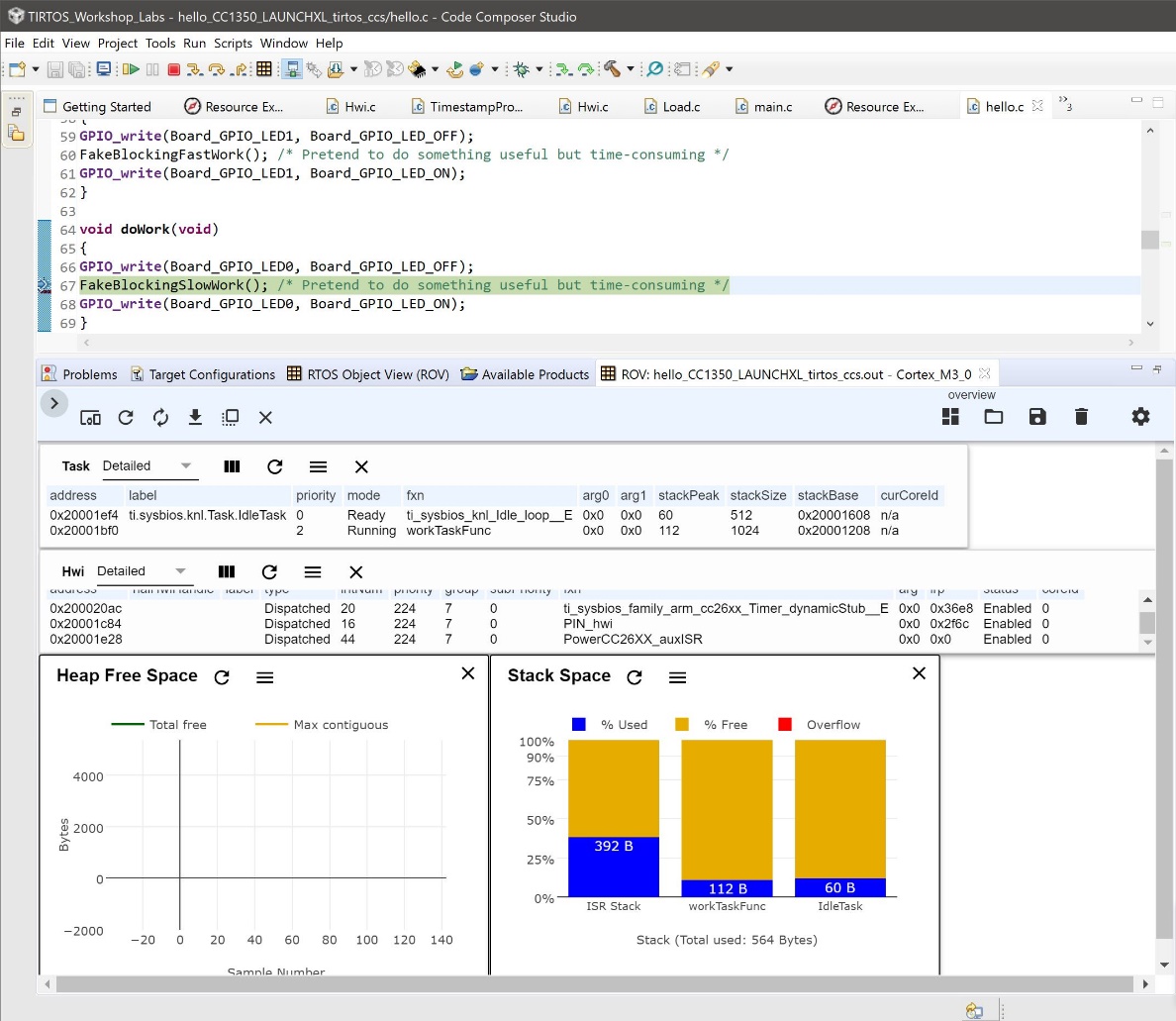
" bx lr\n");

**Task 02:**

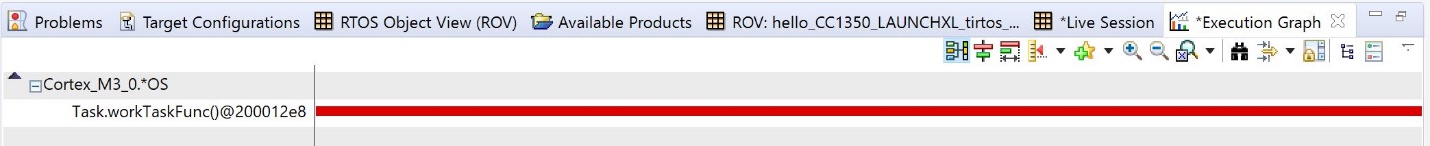
Youtube Link: <https://www.youtube.com/watch?v=6JZRIkYg8IE>

The purpose of Task 02 is to become familiar with the tools needed to perform troubleshooting and debugging on the CC1350. Without these tools, debugging can become incredibly difficult. The program is still the same.

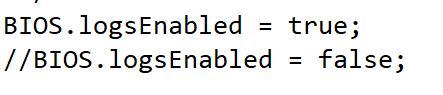
Here is the output when putting a breakpoint at line 67 function: FakeBlockSlowWork() using the ROV option:

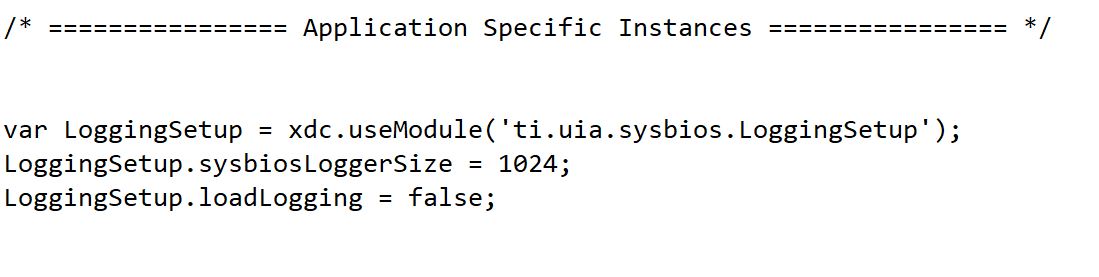


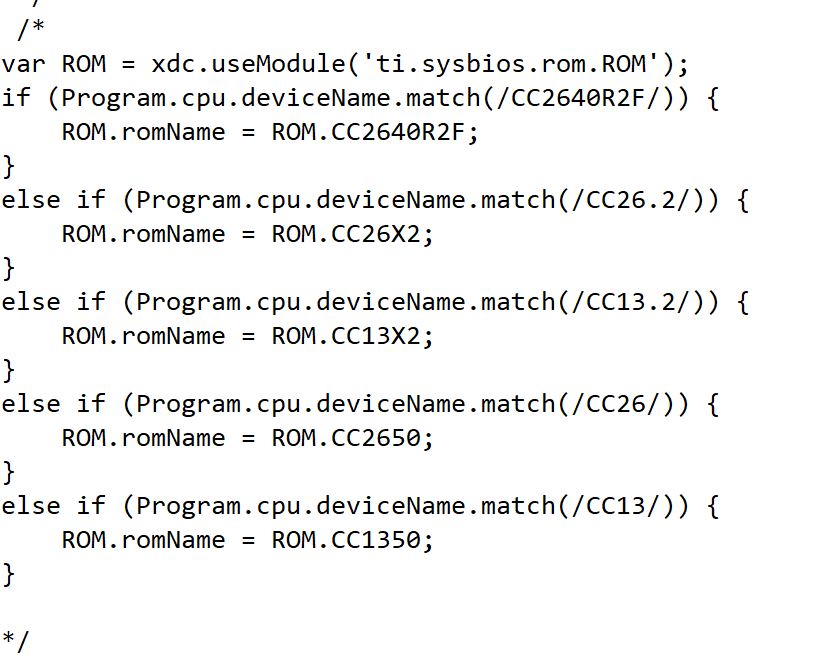
Here is the output when going to RTOS Analyzer -> Execution Analyzer and displaying the Execution Graph. It shows that workTaskFunc is doing all the work.



Below are the modifications made on the hello.cfg file:



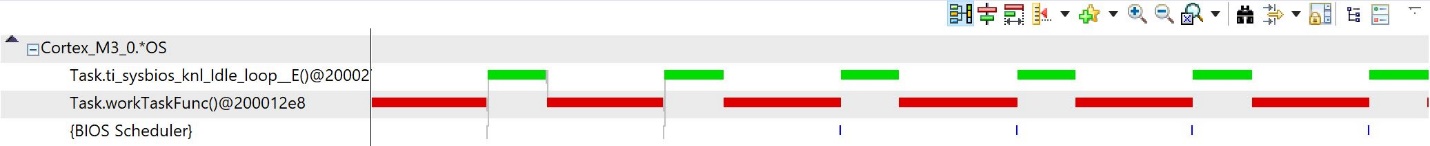


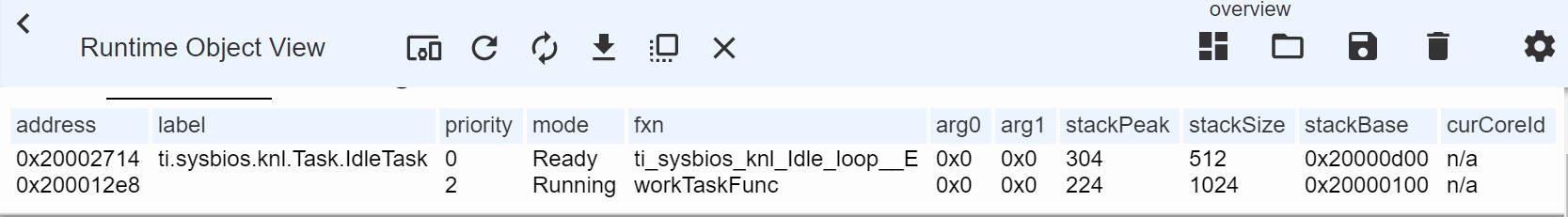


**Task 03:**

Youtube Link: <https://www.youtube.com/watch?v=A_hlm-SkPII>

This program displays the issue when using the delay function. When using the delay function, it is wasting CPU cycles. By adding Task\_sleep() it will actually put the Task to sleep thus causing less waste in resources.





Modified Code:

/\* TI-RTOS Header files \*/

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

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

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

**#include** <ti/sysbios/knl/Clock.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];

**static** uint8\_t urgentWorkTaskStack[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);

Task\_sleep(500 \* (1000 / Clock\_tickPeriod));

}

}

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

{

**while** (1) {

/\* Do work \*/

doUrgentWork();

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

//myDelay(24000000);

Task\_sleep(50 \* (1000 / Clock\_tickPeriod));

}

}

/\*

\* ======== 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 04:**

Youtube Link: <https://www.youtube.com/watch?v=2Dbtvy39zqI>

Youtube Link: <https://www.youtube.com/watch?v=Gply59Z_iT8>

Task 04 consisted of implementation of adding another LED peripheral. The peripheral adds a green LED. In Task04 we can also set the priority of the LEDs. In the first video linked above displays the green LED with priority 1 and the second video displays the priority at 3.

Modified Code:

/\*

\* 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/sysbios/knl/Clock.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;

Task\_Struct urgentWorkTask;

/\* 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];

**static** uint8\_t urgentWorkTaskStack[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);

Task\_sleep(500 \* (1000 / Clock\_tickPeriod));

}

}

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

{

**while** (1) {

/\* Do work \*/

doUrgentWork();

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

//myDelay(24000000);

Task\_sleep(50 \* (1000 / Clock\_tickPeriod));

}

}

/\*

\* ======== 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);

workTaskParams.priority = 3;

workTaskParams.stack = &urgentWorkTaskStack;

Task\_construct(&urgentWorkTask, urgentWorkTaskFunc, &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");