Serak Gebremedhin
Github root directory: https://github.com/Ber-geb/solid-octo-tribble.git

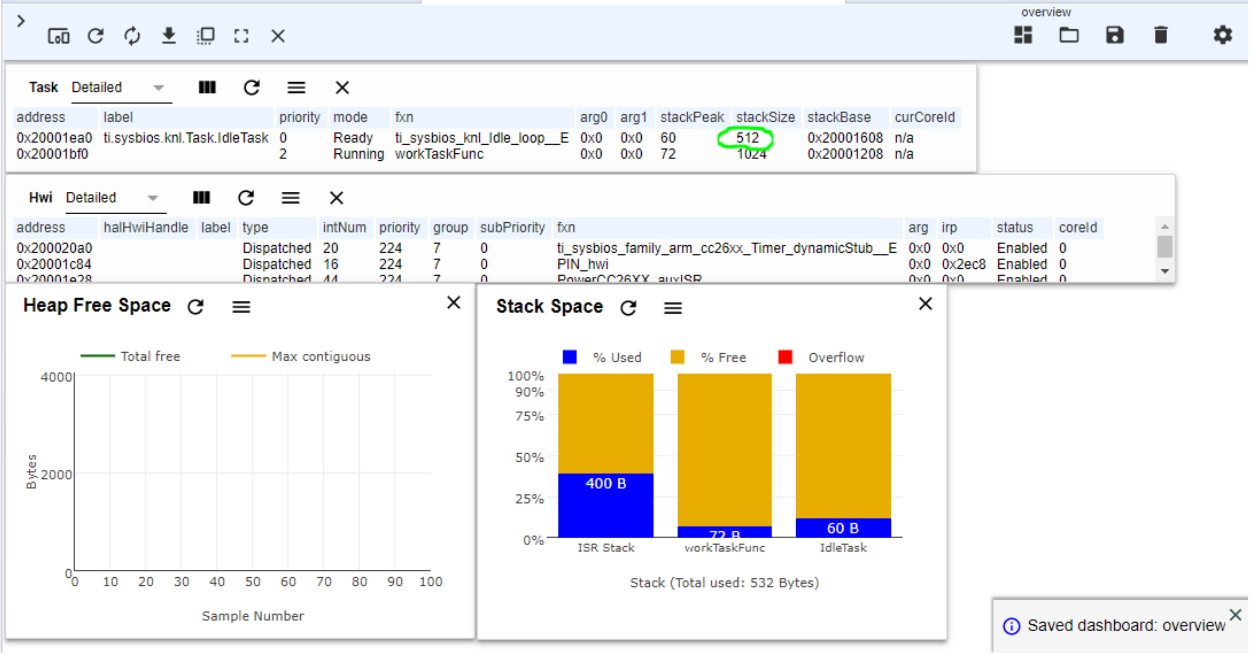
**Date Submitted:** **12/11/19**

---

## Task 01:

Youtube Link: https://youtu.be/Sodr39Yg8BQ

### Modified Schematic (if applicable):
### Runtime Object View (ROV)



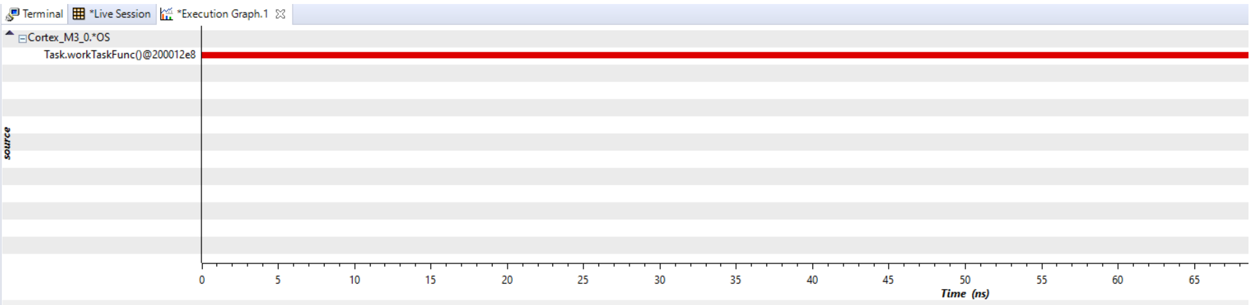
Task Detailed

| address | label | priority | mode | fxn | arg0 | arg1 | stackPeak | stackSize | stackBase | curCoreId |
|---|---|---|---|---|---|---|---|---|---|---|
| 0x20001ea0 | ti.sysbios.knl.Task.IdleTask | 0 | Ready | ti_sysbios_knl_Idle_loop__E | 0x0 | 0x0 | 60 | 512 | 0x20001608 | n/a |
| 0x20001bf0 | | 2 | Running | workTaskFunc | 0x0 | 0x0 | 72 | 1024 | 0x20001208 | n/a |

Hwi Detailed

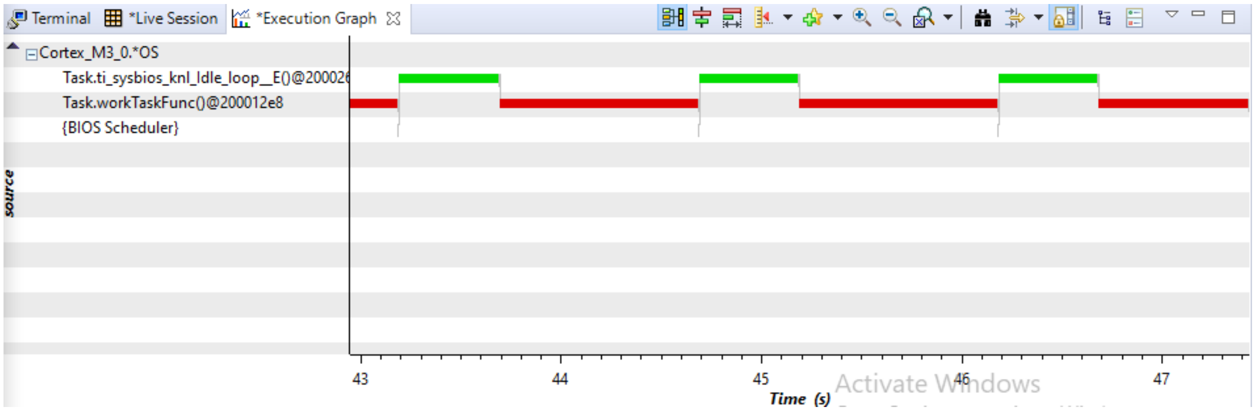
| address | halHwiHandle | label | type | intNum | priority | group | subPriority | fxn | arg | irp | status | coreId |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0x200020a0 | | | Dispatched | 20 | 224 | 7 | 0 | ti_sysbios_family_arm_cc26xx_Timer_dynamicStub__E | 0x0 | 0x0 | Enabled | 0 |
| 0x20001c84 | | | Dispatched | 16 | 224 | 7 | 0 | PIN_hwi | 0x0 | 0x2ec8 | Enabled | 0 |
| 0x20001e28 | | | Dispatched | 44 | 224 | 7 | 0 | PowerCC26XX_auxISR | 0x0 | 0x0 | Enabled | 0 |

### Execution graphs:
### Task1:

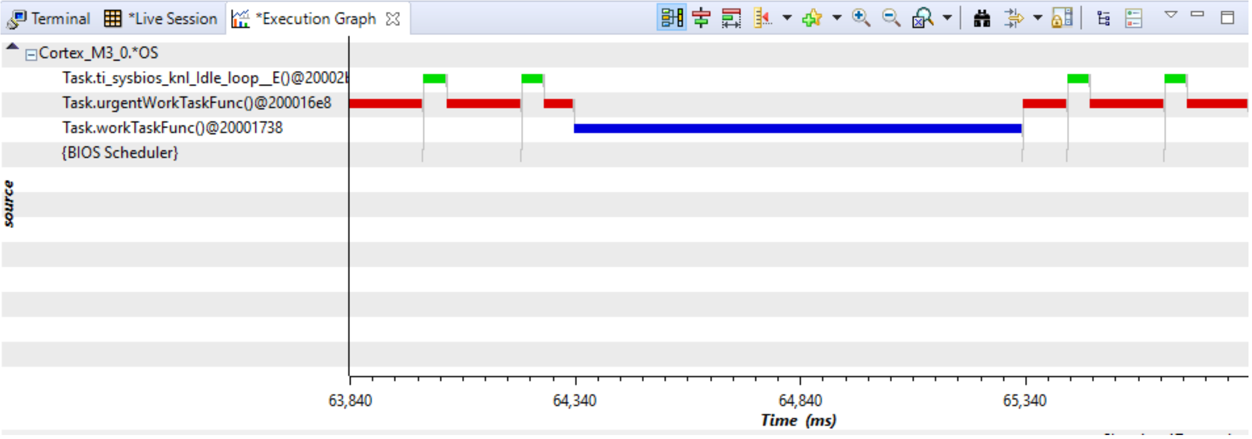


### Task function and idle task:

**Grading scheme:** 30% Coding, 30% Documentation, 40% Execution/Video.



workTaskFunc and urgendWorkTaskFunk:



Modified 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>
#include <ti/sysbios/knl/Clock.h> //use of 'Clock_tickPeriod'
/* 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)


//information regarding each task - how the kernel will act for each task
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];


//doUrgentWork will utilize LED 1 (not LED 0)
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 afrg1)
{
    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)); //sleep for 500ms, does not waste
CPU cycles
    }
}

void urgentWorkTaskFunc(UArg arg0, UArg afrg1)
{
    while (1) {
        /* Do 'urgent' work */
        doUrgentWork();
        /* Wait a while, because doWork should be a periodic thing, not continuous.*/
        //myDelay(24000000);
        Task_sleep(50 * (1000/Clock_tickPeriod)); //sleep for 50ms, does not waste
CPU cycles
    }
}

/*
 * ======== 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;

    //create the task..
    Task_construct(&workTask, workTaskFunc, &workTaskParams, NULL);

    //set up new task (urgent task)
    workTaskParams.priority = 3; //priority previously was 1
    workTaskParams.stack = &urgentWorkTaskStack;

    //create the new task..
    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");
```

**Grading scheme:** 30% Coding, 30% Documentation, 40% Execution/Video.

## Task 03:

Youtube Link:

**Modified Schematic (if applicable):**

**Modified Code:**

```
// Insert code here
------------------------------------------------------------------------------------
```

**Grading scheme:** 30% Coding, 30% Documentation, 40% Execution/Video.