Debugging Applications that use TI-RTOS Technical Lab

Todd Mullanix

TI-RTOS Apps Manager

Lab Introduction

Goal

Use the techniques learned in the *Debugging Applications that use TI-RTOS Technical Overview* to solve common programming errors in an application.

Reminder: we learned about

- **Stack overflows**
- **Exceptions**
- **Memory Mismanagement**

Setup

The lab is based on the **MSP-EXP432P401R Launchpad.**

You need the following software which is included on the thumb drives (or zip file). Please copy to pieces that you need into c:\FAESummit.

- **CCS 6.1.2** (make sure you installed support for MSP432)
- **TI-RTOS for MSP43x v2.16.00.08** (**http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/tirtos/index.html**)
- **Exported CCS Console Projects**

Application

The application is more of a starter project. It has a simple console via the UART and has support for the two buttons.

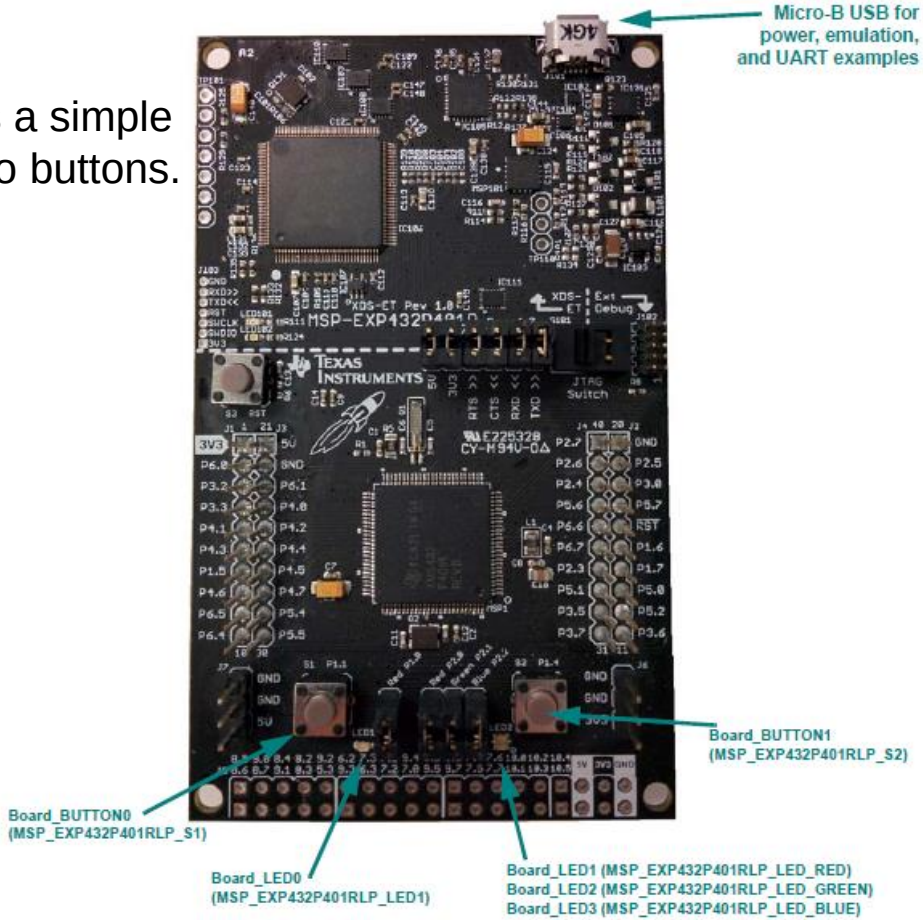
Here's the **console** options

```
Valid Commands
--------------
h: help
c: clear the screen
m: Display memory stats
p: print HeapTrack stats
r: restart console
>
```

Button0: increments a counter

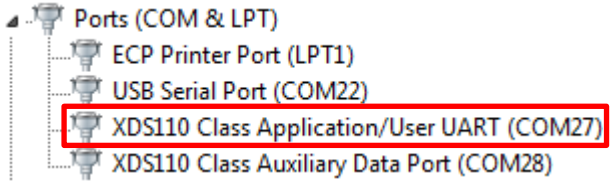
Button1: prints out how many time Button0 was pushed.

Note: System_printf and printf output is routed to the UART.

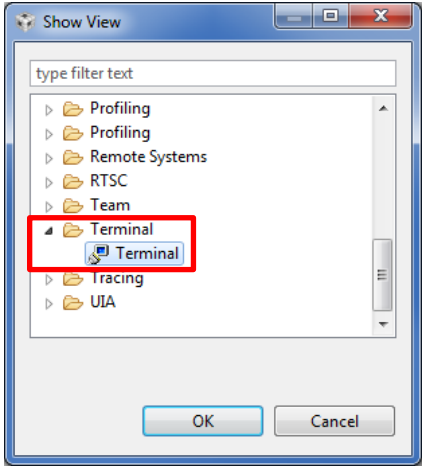


Lab 1: Terminal Session

1a. Open Window's "Device Manager" and find which port is the UART.

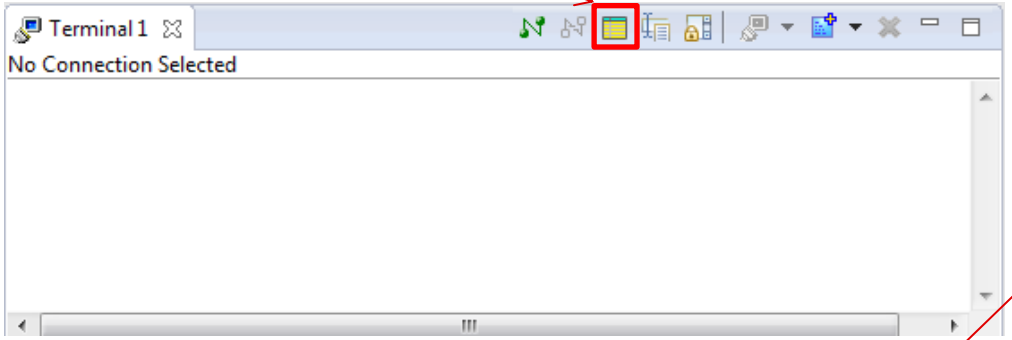


1b. Open your favorite terminal window. "Terminal" is in CCS. Open "View->Other…" and find "Terminal".



Lab 1: Terminal Session

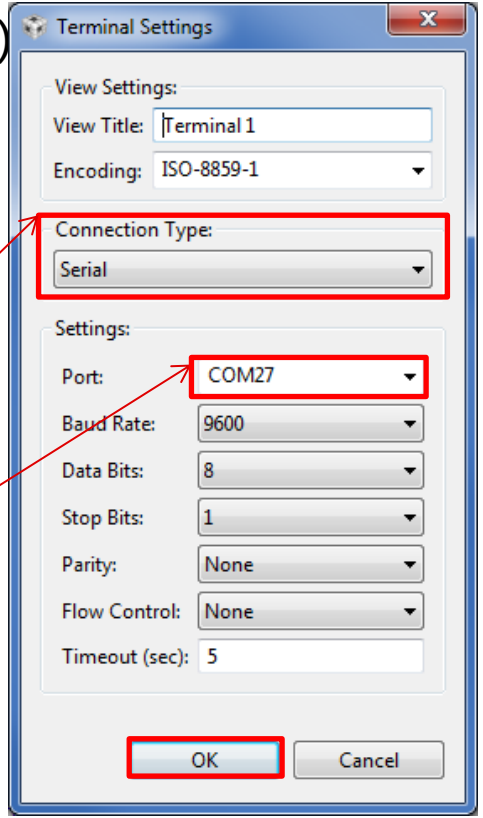
1c. Select “Settings” in the Terminal Session (if using Terminal)



1d. Configure the “Connection Type” to be “Serial”

and select the port specified in the Device Manager (you may have to type in the string).

Note: We’ve seen issues with the MSP432 UART with Window drivers.
If the port cannot be opened, close CCS and unplug the Launchpad. Plug the Launchpad back in and start CCS. Reload the application and start terminal again.



Lab Problem

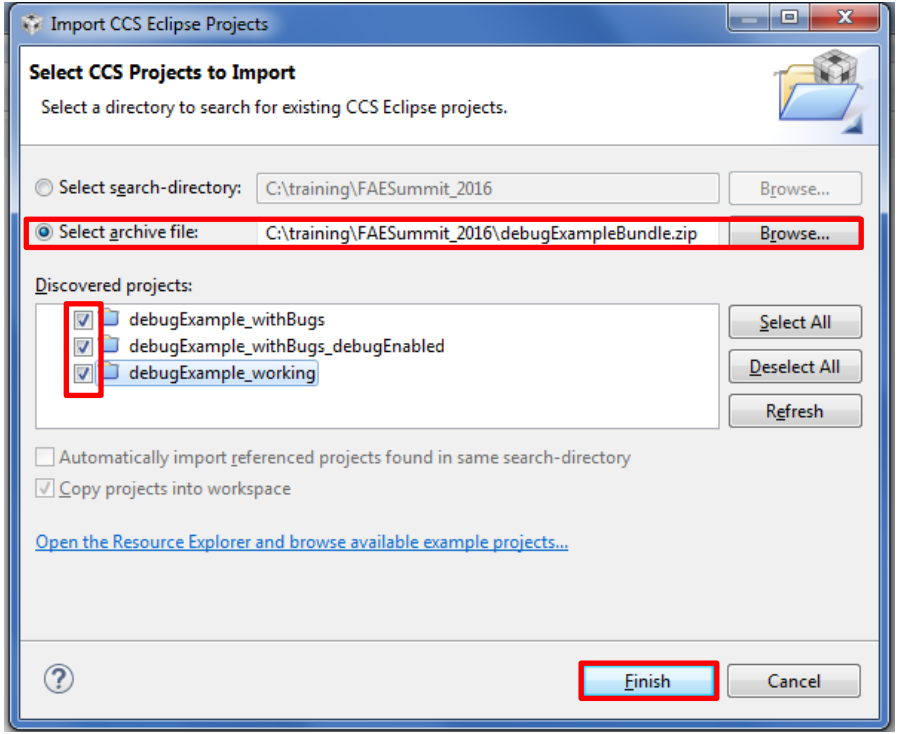
There are three problems with this application. **All the three subtle problems are in main.c.** Play around with the console commands and buttons on the board to see if you can find and fix them.

The following projects are supplied for users:

- **debugExample_withBugs**: Debugging is turned off. This project appears to work fine…?
- **debugExample_withBugs_debugEnabled**: Same as above, but with the debugging topics we talked about enabled (at the bottom of the .cfg file).
- **debugExample_withBugs_working**: Bugs fixed and debugging disabled.

Importing zipped projects into Desktop CCS

Project->Import CCS Projects…



Lab: What's wrong?

Import the *debugExample_withBugs* project (see the previous slide for help). Build and run the example on the MSP-EXP432P401R launchpad. See if you can determine any problems before halting and doing the following techniques we've learned.

1. **Check System and Task stack peaks in ROV or "Scan for Errors…":** A quick and easy way to see if there are any issues detected is select "BIOS->Scan for errors…" in ROV. Stack overflows will show up here as well as Hwi and Task.
2. **Turn on TI-RTOS "Minimal" or "Enhanced" Exception Handling**.
3. **Enable HeapTrack if you have a dynamic allocation.**

Big hint: the debugging options we learned are disabled in the **bottom** of the .cfg file, so it might be useful to enable them (or you can import the *debugExample_withBugs_debugEnabled* project as a short-cut to enabling them).

Can you find and fix the 3 bugs in main.c now? The next slides have the answers…

Answers: Bug #1…Stack too small

If we halt the target and "**Scan for Errors…**" in ROV->BIOS, we see a strange message about the task stack

Module | Scan for errors... | Raw

mod	tab	inst	field	message
ti.sysbios.knl.Task	Detailed	console	blockedOn	Invalid task internal state: pend element address (0x200009d4) is not within the task's stack

Let's enable stack initialization and checking in the .cfg file and then rebuild/reload/run.

```
Task.initStackFlag = true;
Task.checkStackFlag = true;
halHwi.initStackFlag = true;
halHwi.checkStackFlag = true;
```

Now when we run it and look, we see the Task stack overflow (also shows in Task->Detailed in ROV).

```
> Cti.sysbios.knl.Task: line 373: E_stackOverflow: Task 0x20000fc0 stack overflo
w.
xdc.runtime.Error.raise: terminating execution
```

Let's bump it up to 1024 and rebuild/reload/run.

```
#define TASKSTACKSIZE     1024
```

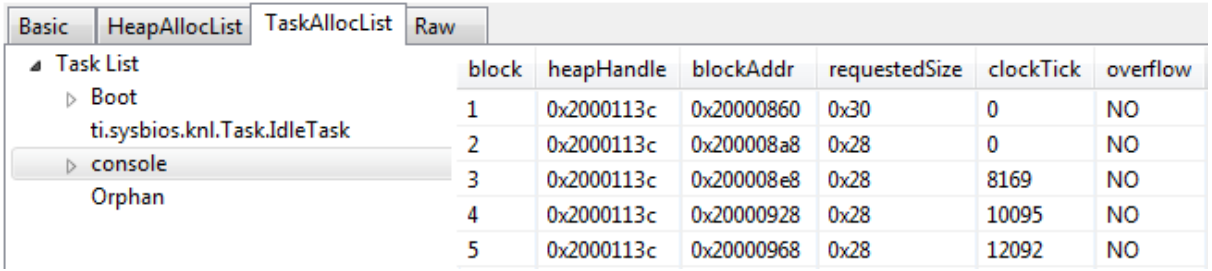
When we are done, we could shrink this down to better number.

Answers: Bug #2…Memory Leak

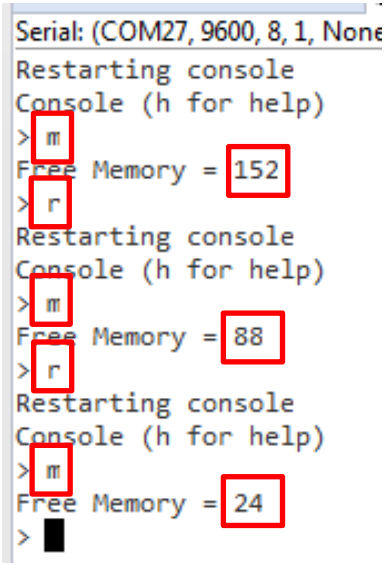
If you restarted (r) the console and happened to look at the memory usage… it is going down! Something is leaking! Let's turn on HeapTrack (rebuild/reload/run).

```
var HeapTrack = xdc.useModule('ti.sysbios.heaps.HeapTrack');
BIOS.heapTrackEnabled = true;
```

Before the first clock tick, we allocated a block of 0x28 (40D). After ~8.2, ~10.0, etc. seconds we have the same size again. We are only using malloc, so that has an 8 byte overhead. So we are looking for a size of 32 bytes in the console app.



block	heapHandle	blockAddr	requestedSize	clockTick	overflow
1	0x2000113c	0x20000860	0x30	0	NO
2	0x2000113c	0x200008a8	0x28	0	NO
3	0x2000113c	0x200008e8	0x28	8169	NO
4	0x2000113c	0x20000928	0x28	10095	NO
5	0x2000113c	0x20000968	0x28	12092	NO

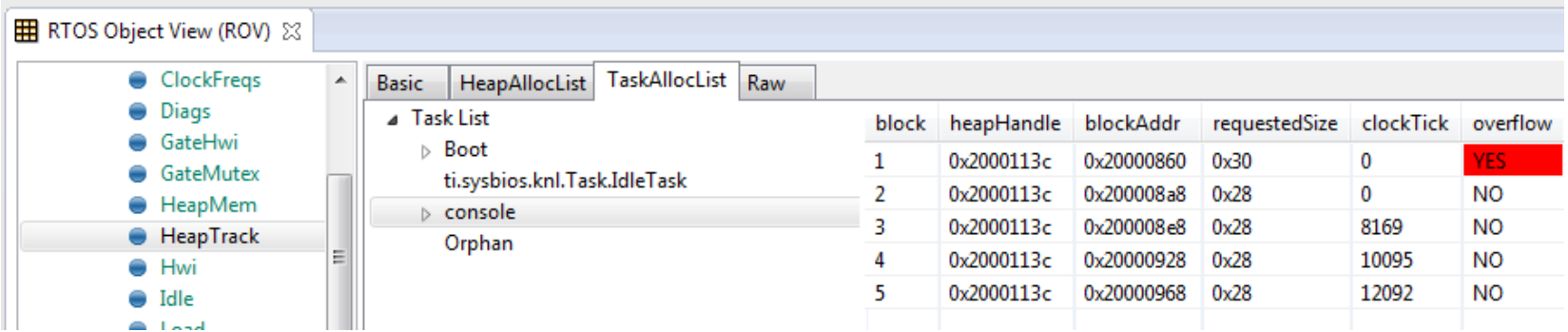


After looking at simpleConsole() function we see that the comment/check was wrong!

```
/* simpleConsole returns 0 if the buffer needs to be freed */
rc = simpleConsole(consoleBuffer, CONSOLEBUFFERSIZE);
if (rc == 0) {
   free(buffer);
}
```

Answers: Bug #3…Overflowing Buffer

Try pushing button0 10 times or more and then button1 to print the count. Look the console task's memory allocation in HeapTrack.



block	heapHandle	blockAddr	requestedSize	clockTick	overflow
1	0x2000113c	0x20000860	0x30	0	YES
2	0x2000113c	0x200008a8	0x28	0	NO
3	0x2000113c	0x200008e8	0x28	8169	NO
4	0x2000113c	0x20000928	0x28	10095	NO
5	0x2000113c	0x20000968	0x28	12092	NO

There is an overflow. With a little searching and maybe looking at the memory browser, we see that we did give enough space for string + two digits + two '\n' in counterStr in main(), but forgot the string terminate character ('\0'). Reminder: strlen does not include the '\0' character in the returned count.

```
/*
 * Allocate buffer for gpioButtonFxn1.
 * Get the size of the string + 2 (for two digits) + 2 (for '\n' chars).
 */
 countStr = malloc(strlen(PUSH_STR) + 4);
```