TI-RTOS Half-Day Hands-on Training

May 31, 2013

Based on TI-RTOS 1.10.00.23

Agenda

- Prerequisites Check
- Introduction to TI-RTOS
- Build UART Example
- “Under the Hood”
- Break
- HTTP Lab
- Summary & Q/A

Prerequisites Check

It is assumed you have the following

Software

1. CCS 5.4.0 installed
2. TI-RTOS 1.10.00.23 installed
3. Files for HTTP lab (available at http://processors.wiki.ti.com/index.php/TI-RTOS_HTTP_Example).

If you are missing any of these, please use the provided USB thumb drives to install the missing software.

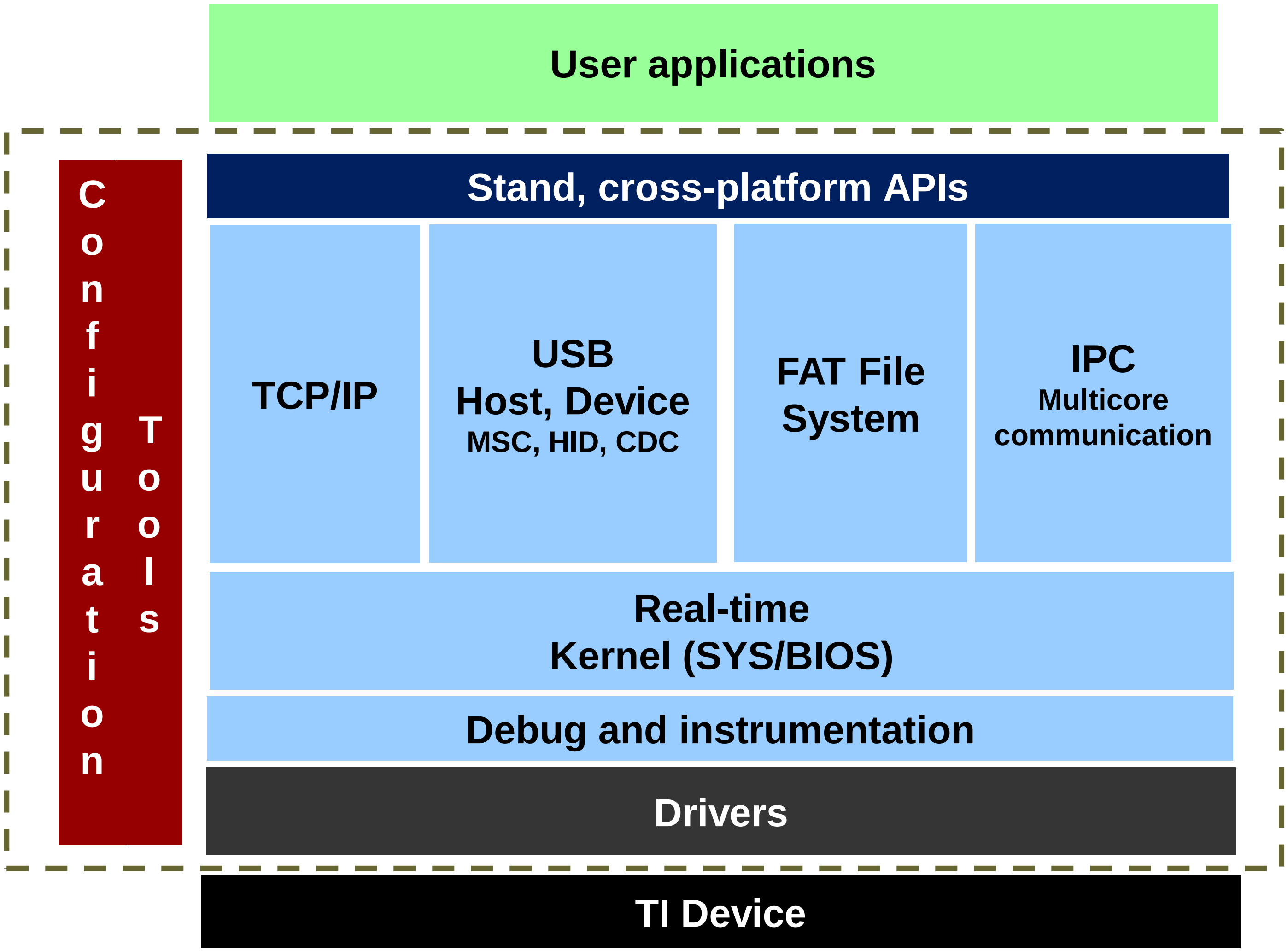
Hardware

1. Target Board: Concerto (TMDXDOCK28M36 or TMDXDOCKH52C1) board and power.
2. Mini-B USB cable for emulation and UART connection to the target
3. Ethernet cable
   - Standard cable if using local network that has a DHCP server present
   - Cross-over cable if connecting directly to your PC.

What is TI-RTOS?

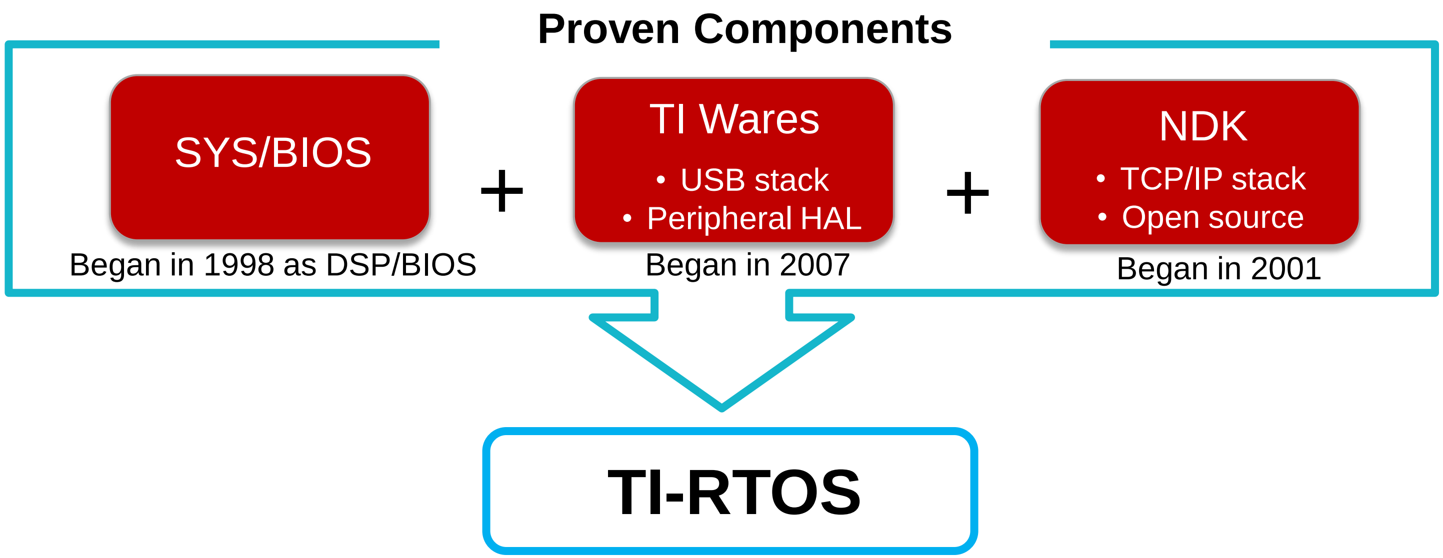
- A real-time operating system that scales from a minimal 3 Kb kernel to a fully network-enabled environment
- Compatible with CodeComposer Studio (CCS) IDE v5.x, but shipped independently
  - Hosts: Windows XP (SP2 or SP3), Windows Vista, Windows 7, Linux [RH4], Ubuntu [10.04]
  - Single install of all components
  - Docs: Getting Started Guide, User Guide
  - Projects and examples are fully integrated to work with CCS
- Completely free
  - No development or per-project license fees
  - No run-time royalties
  - All source code provided
- Developed and supported by TI

What is TI-RTOS?



TI-RTOS Uses Robust, Proven Components

- Reliability is critical in embedded applications
- TI-RTOS is based on mature software components



Why is TI-RTOS Important?

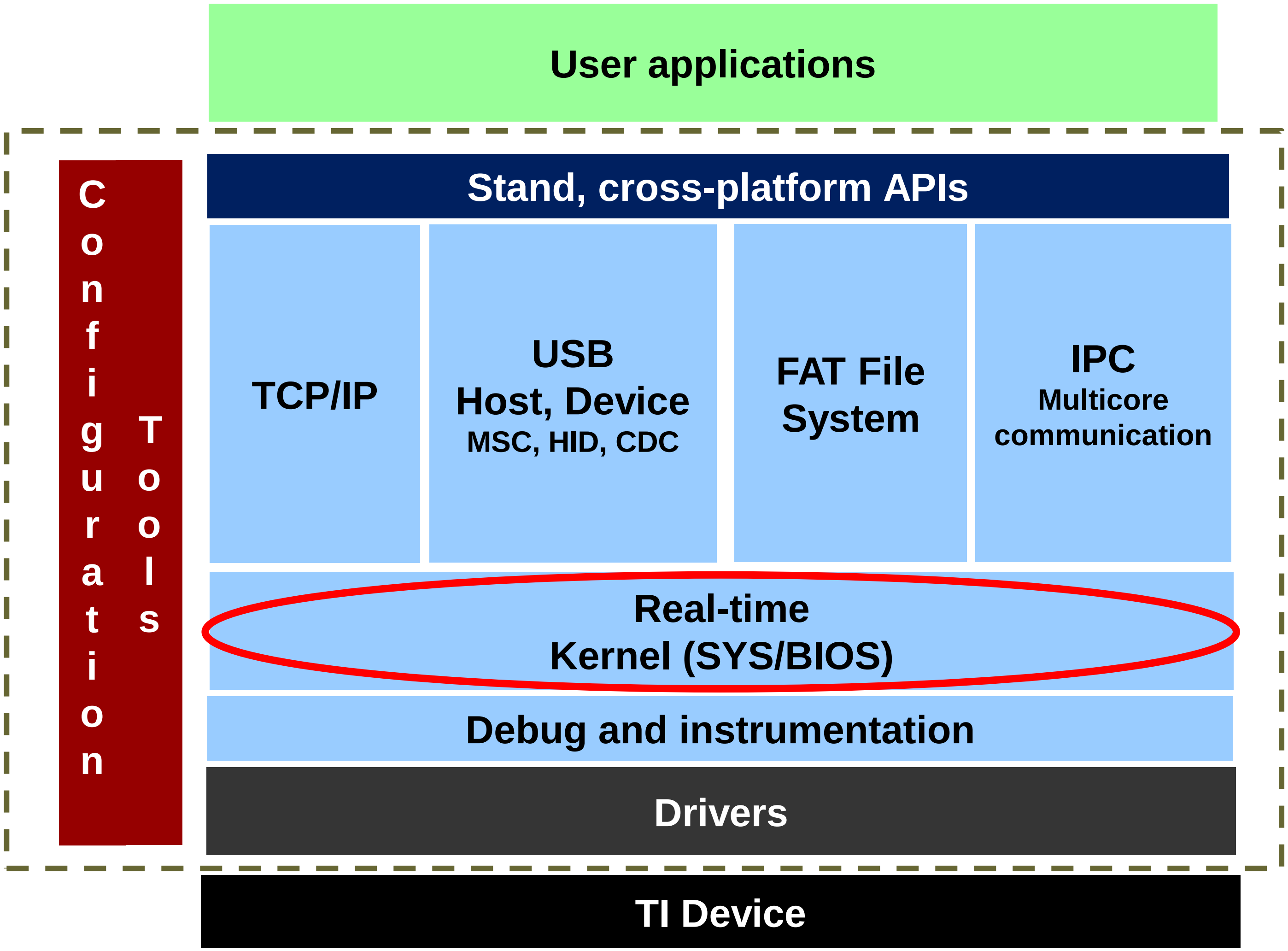
- Today's MCU applications are becoming increasingly complex – TCP/IP, USB, wireless connectivity, touch-screen GUIs, …
- Developers want to focus on their specific application:
  - TI-RTOS includes commonly required features such as TCP/IP and USB off-the-shelf
- For more specialized software (such as cryptographic libraries), developers also require access to a rich software ecosystem
  - TI-RTOS provides a standard software platform that third-party developers can easily target without NRE
- TI MCU customers also want to be able to select the most appropriate TI MCU and quickly port existing applications easily
  - TI-RTOS provides a standard software platform across TI's different MCU ISAs, making applications highly portable

TI-RTOS Release History and Roadmap

Version	Date	Features
1.00.00	July 16, 2012	TMDXDOCKH52C1 (for F28M35x), EKS-LM4F232
1.00.01	Sep 28, 2012	Adds LM4F120H5QR
1.01.00	Jan 28, 2013	Adds TMDXDOCK28M36 (for F28M36x)
1.10.00	**May 2013**	**SPI Driver, CC3000 support**
1.20.00	Q3 2013	MSP430, IAR Support
1.30.00	Q4 2013	Additional devices/drivers based on BU and customer demand

- Download TI-RTOS releases from: http://www.ti.com/tool/ti-rtos

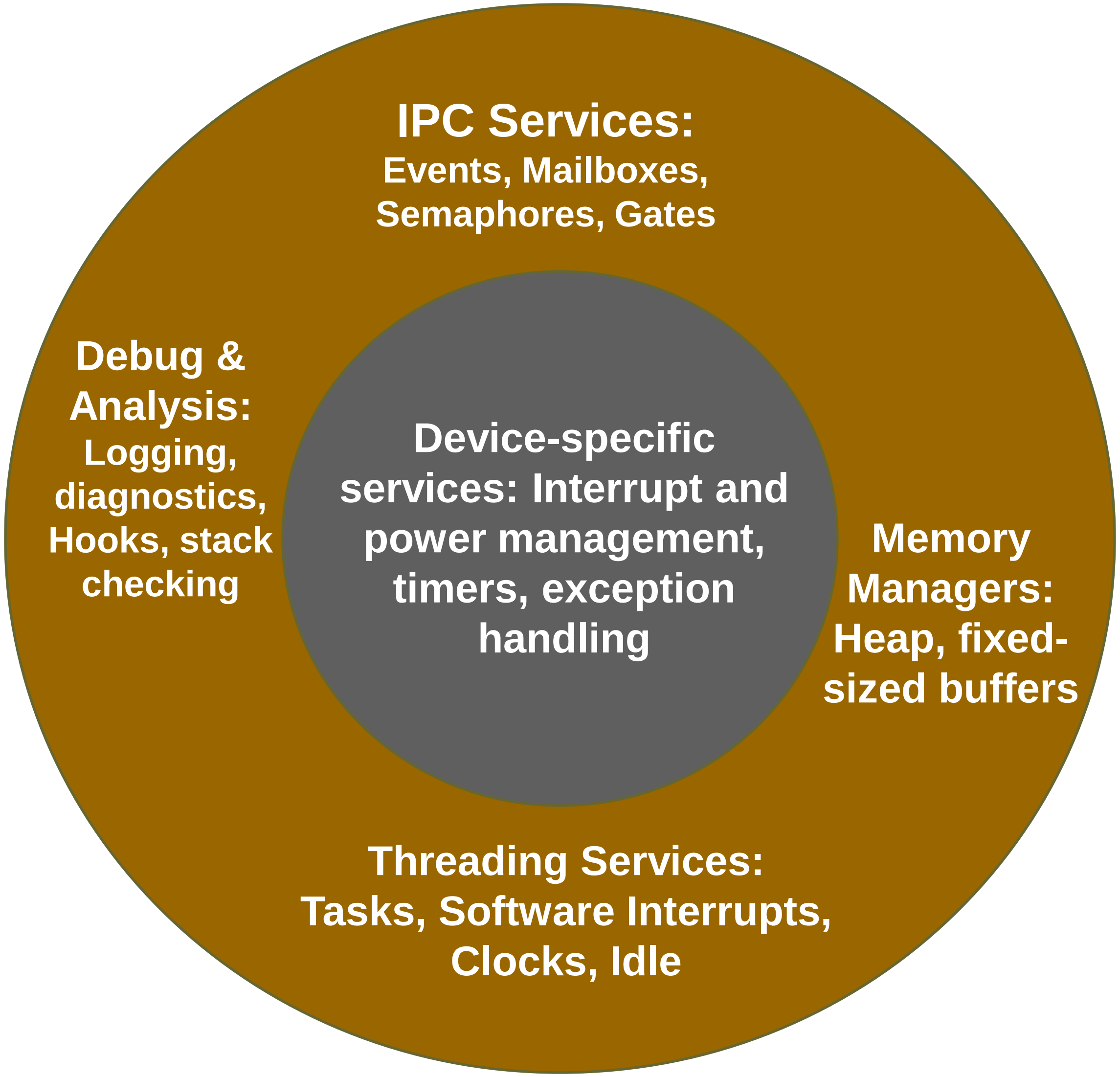
TI-RTOS: Real-Time Kernel



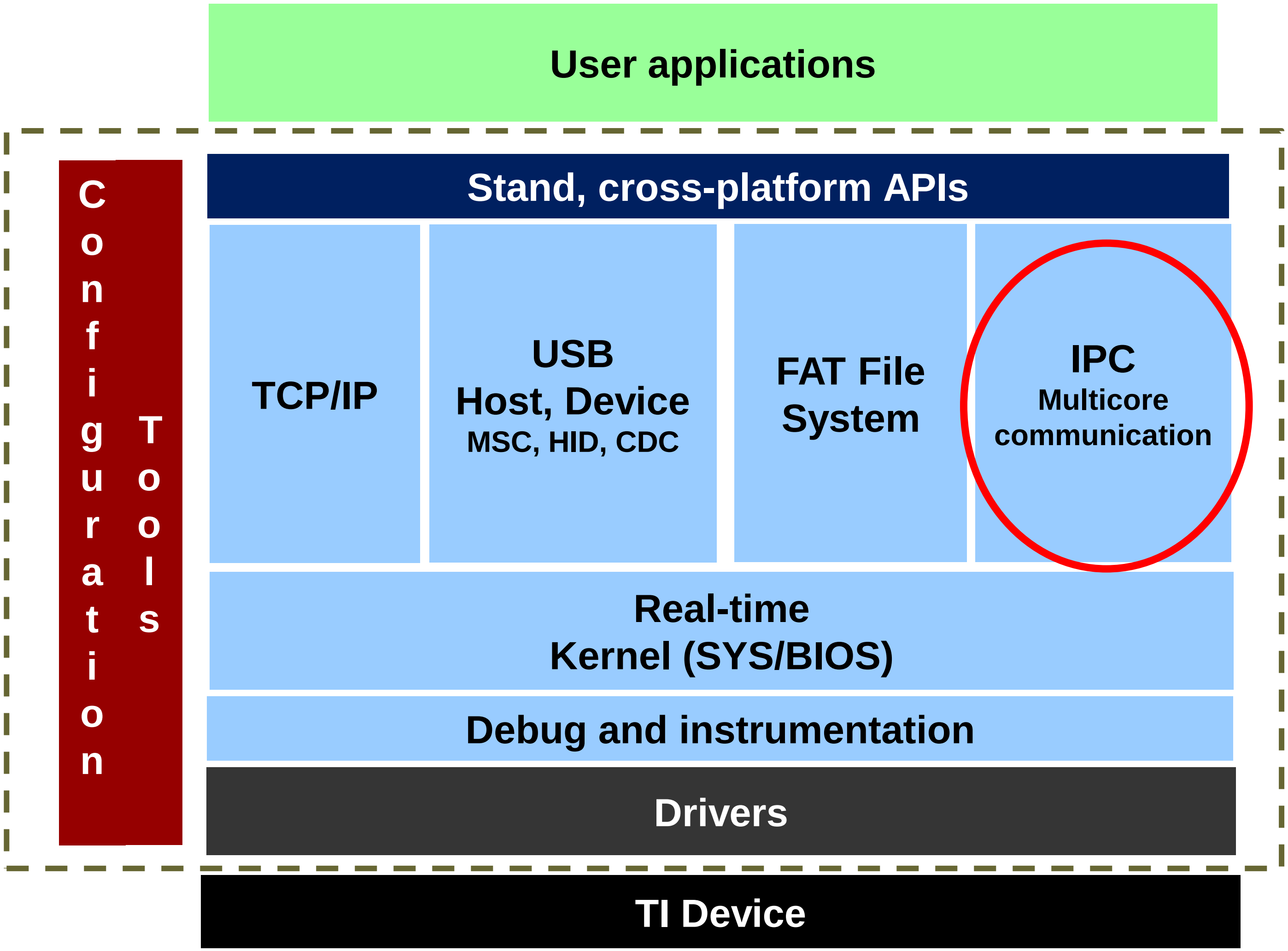
SYS/BIOS Real-Time Kernel

- Designed for real-time applications
  - Scheduler is deterministic so kernel system calls complete operation in a predictable time
  - Interrupt latency is low
  - "Zero-latency Interrupts" enable kernel to be used in hard real-time applications
- Low footprint to meet MCU memory constraints
  - Kernel is highly configurable so unneeded functions are excluded
  - Static configuration enables very low footprints by eliminating need for heaps or create/delete calls if desired
- Kernel system calls accessible as C function calls
- Interrupt handlers are easy to write in C
  - Kernel's interrupt dispatcher handles low-level specifics

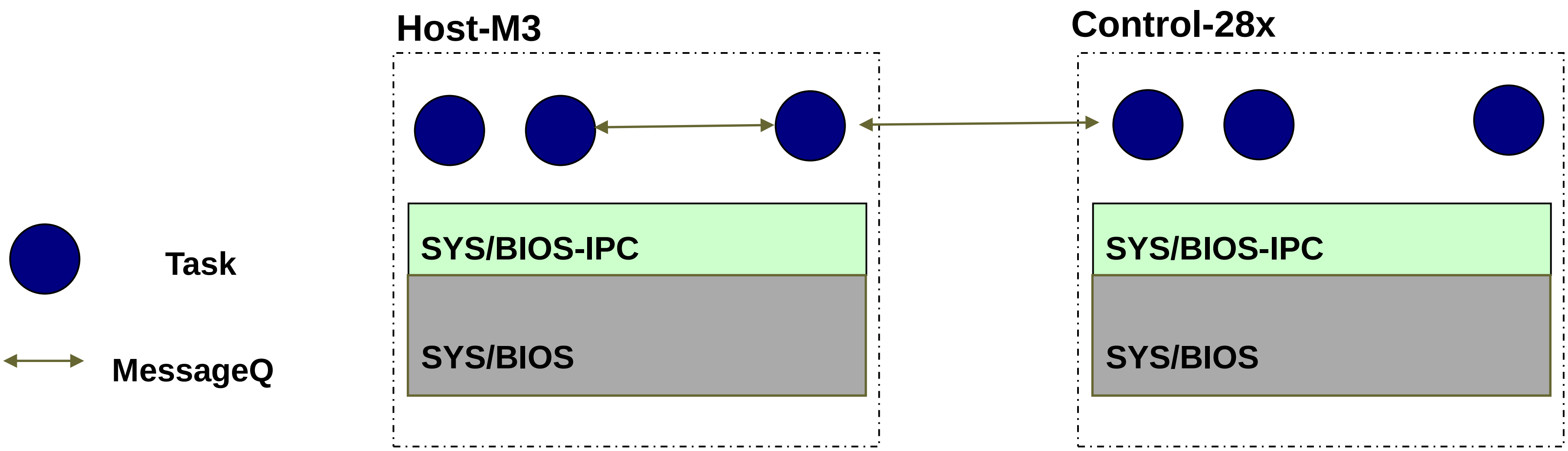
Kernel Services



TI-RTOS: Concerto M3-C28x Communication

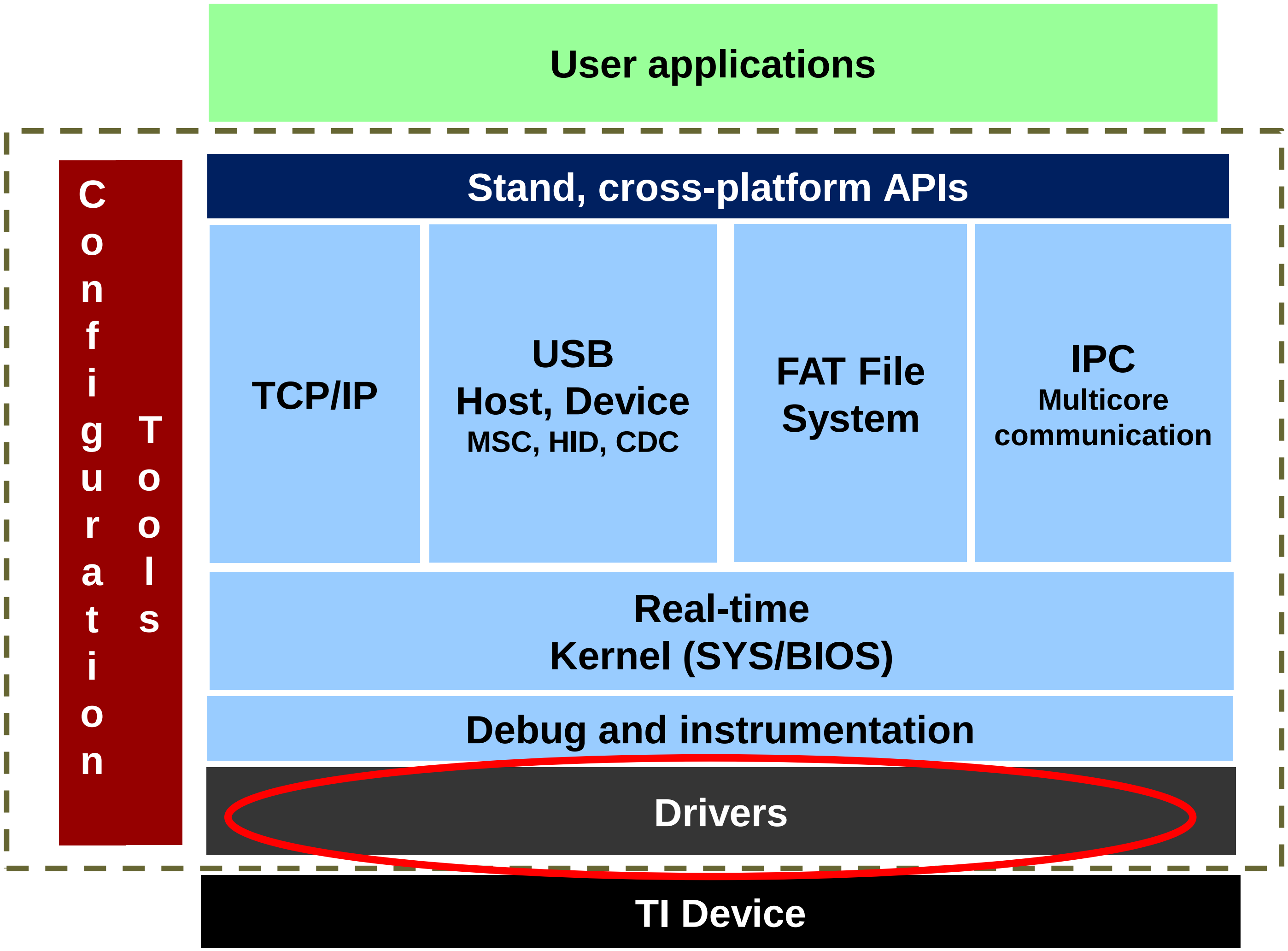


Multi-Core Support



- Simplified development as same real-time kernel runs on both cores
- Inter-Processor Communications (IPC) provides communication APIs between M3 and C28x cores
  - MessageQ for sending variable sized messages
  - Notify for fast single 32-bit word interface between cores
- Time-correlated log data aids debug of multi-core issues
- SPI MessageQ driver enables discrete MCU to MCU communication

TI-RTOS: Device Drivers



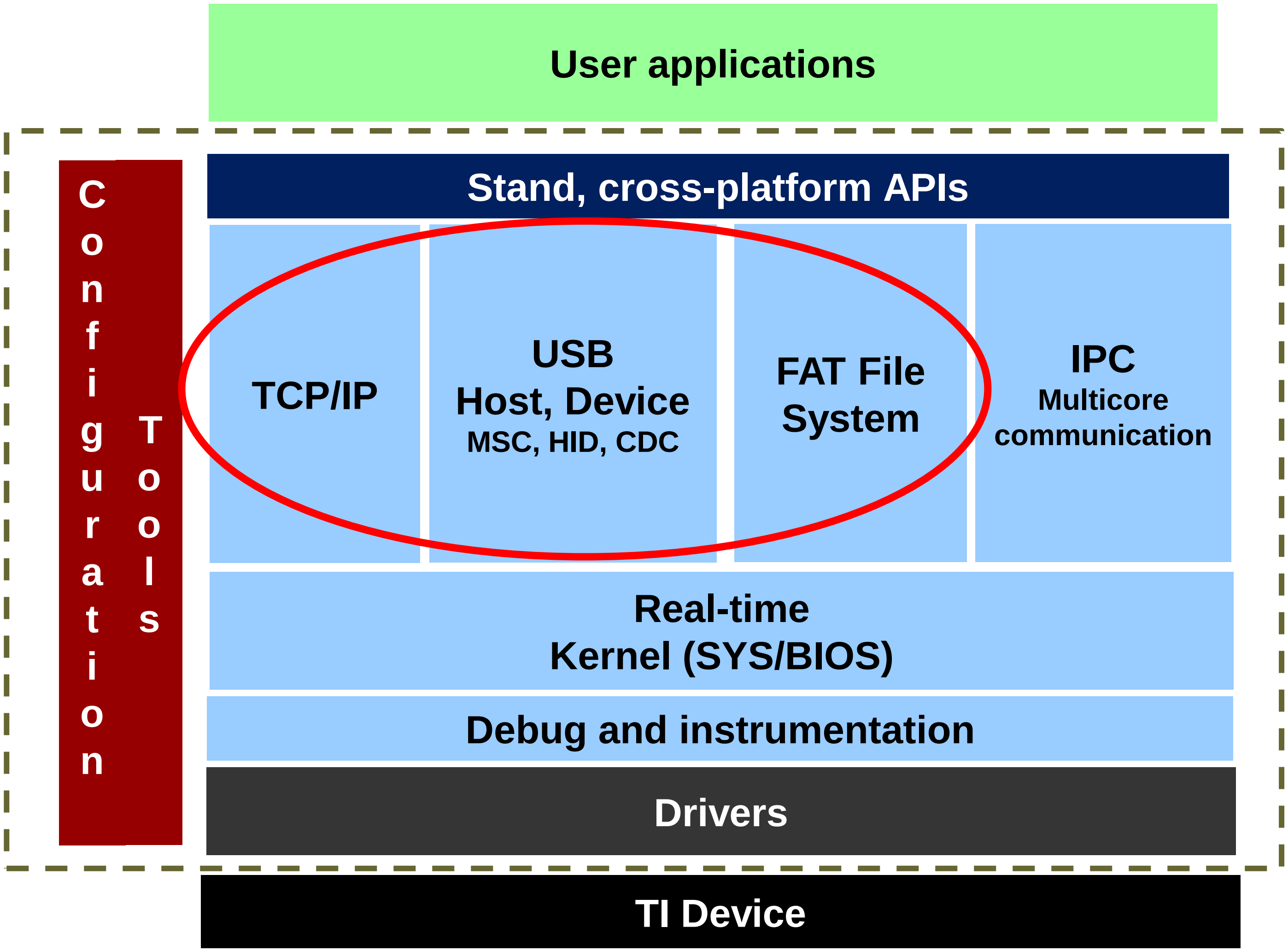
Device Drivers

Driver	Description
Ethernet	Used by TCP/IP stack
WiFi	Used by socket interface
SD Card	Used by FAT File System (uses SPI interface)
USB Host MSC to USB Flash Card	Used by combination of FAT file system, USB Host stack, and MSC class driver
USB HID	Working example code that will likely be modified by user
USB CDC	Working example code that will likely be modified by user
UART	Intended for direct use by application
SPI	Intended for direct use by application
Watchdog	Intended for direct use by application
I^2C	Intended for direct use by application
GPIO	Intended for direct use by application

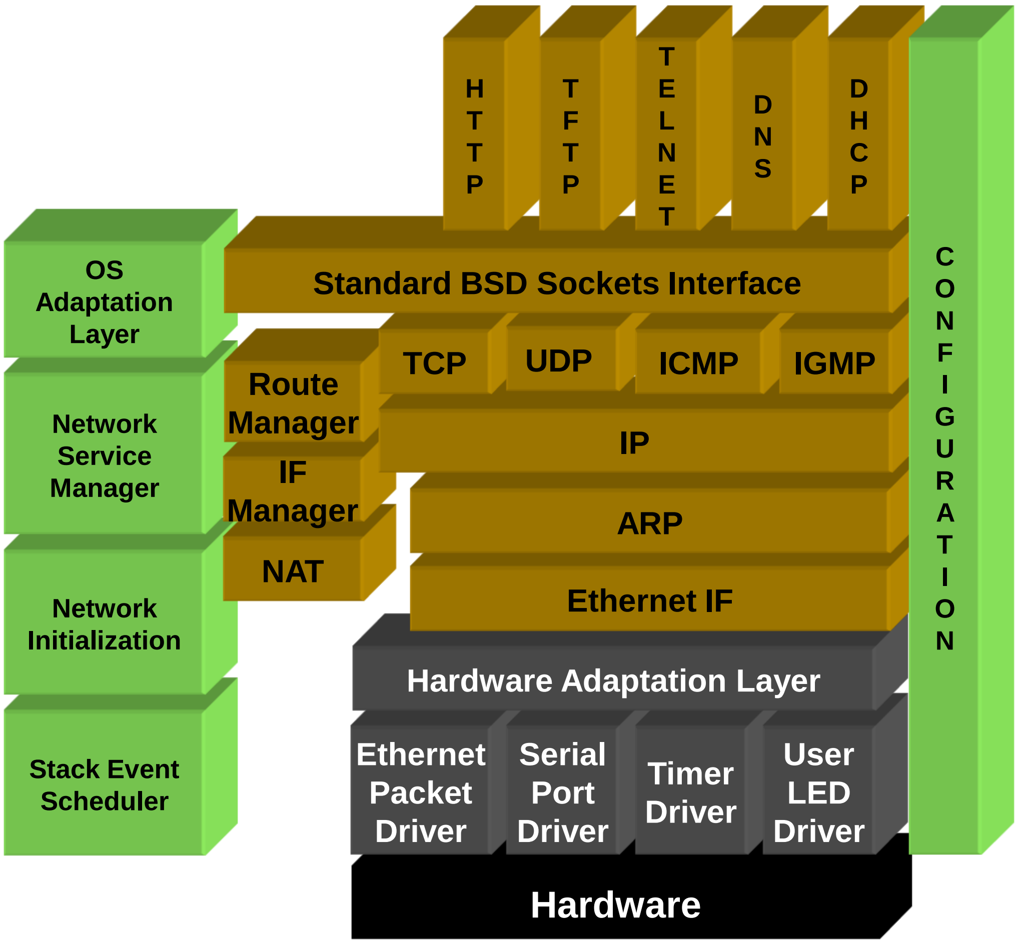
TI-RTOS Device Drivers

- UART, I2C, GPIO, SPI and Watchdog all have their own API set
  - APIs are consistent across device families making code portable
- Full C source code provided
- Drivers use TivaWare/MWare code
  - TI-RTOS Drivers designed to use *Ware code in a thread-safe way
  - TI-RTOS drivers plug into SYS/BIOS interrupt handler
  - TI-RTOS applications will use *Ware peripheral APIs directly for cases where the peripheral is not supported by TI-RTOS
    - Use of *Ware with TI-RTOS is covered on an external wiki site
- Drivers are written to work with tasks
- Driver configuration is very basic:
  - Turn instrumentation on/off
  - Currently there is no GRACE-like capability to configure bit-field values

TI-RTOS: Middleware



TCP/IP Stack



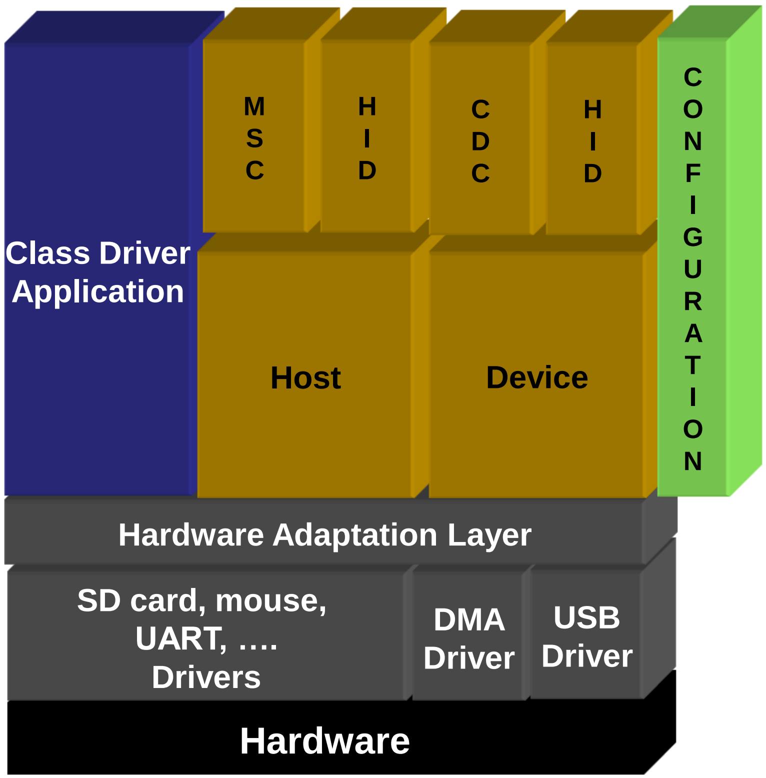
TCP/IP Key Features

- ✓ Supports both IPv4 and IPv6
- ✓ DHCP Client and Server
- ✓ HTTP Server
- ✓ Standard BSD Sockets interface
- ✓ Zero-copy sockets interface available
- ✓ Highly configurable to meet footprint constraints
- ✓ Example on how to create daemon task required by TCP/IP stacks

Wireless LAN Support

- TI-RTOS provides a WiFi driver
  - 1.10 version is based on CC3000
  - It currently only allows a single Task to use the networking stack. A future release of TI-RTOS will allow multiple threads to access the stack.
- Socket based interface
- Does not currently work with NDK networking applications
  - Future release will include HHTP over wireless
- Does not currently allow both NDK and WiFi in the same application.

USB Stack



USB Key Features

- ✓ Uses TivaWare USBLIB unmodified
- ✓ MSC Host Class Driver
- ✓ HID Host & Device Class Drivers
- ✓ CDC Device Class Driver
- ✓ Examples for each class driver
- ✓ Example of using MSC Host Driver under FAT file system

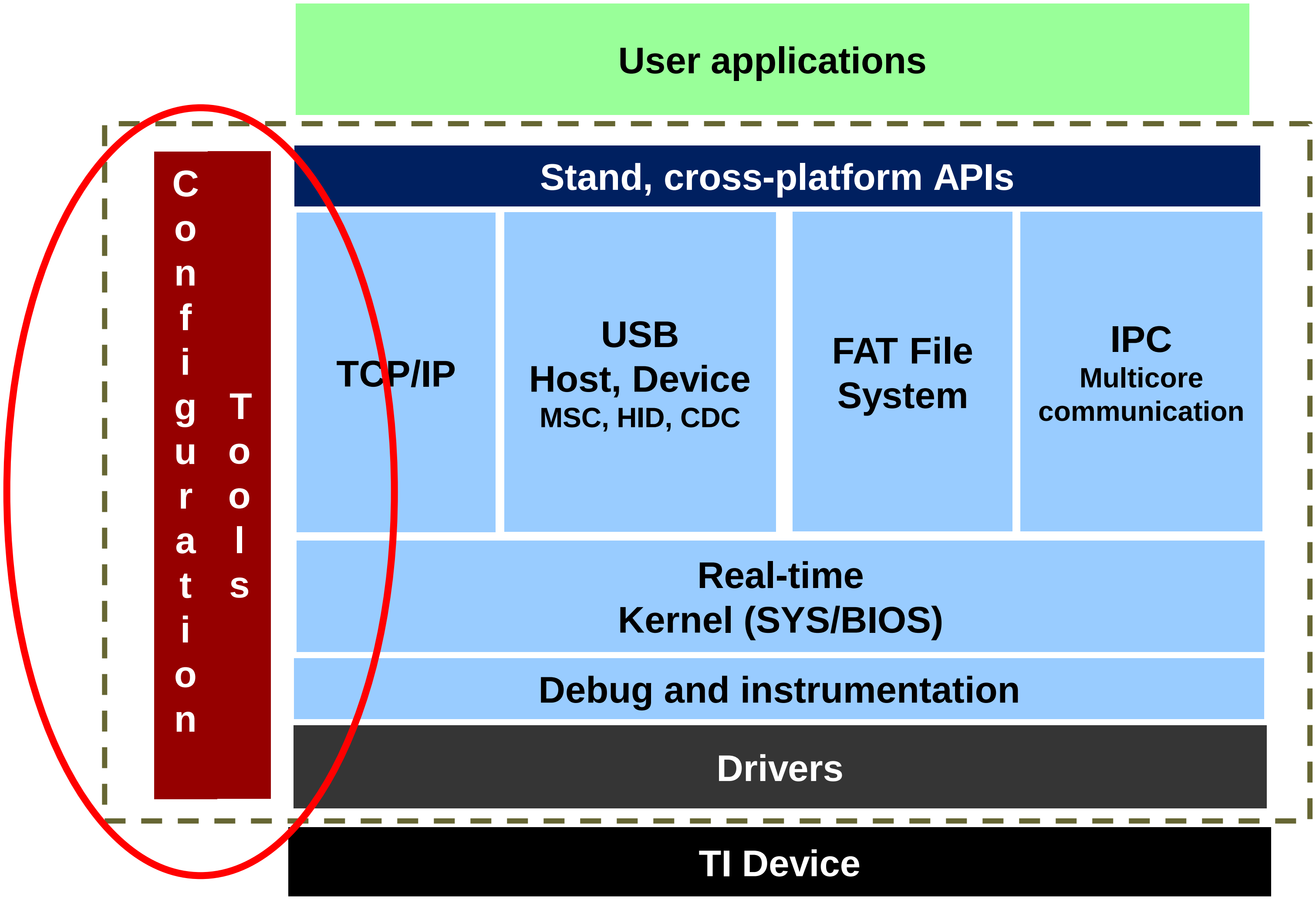
File System

- We use open source software called FatFs
  - The FAT file system is shipped within SYS/BIOS
- Key features:
  - Separate buffer for FAT structure and each file, suitable for fast multiple file access.
  - Supports multiple drives and partitions.
  - Supports FAT12, FAT16 and FAT32
    - Supports 8.3 format file name.
  - Long file names (VFAT) are not supported in our default build
    - Customer can rebuild sources to add
    - TI does NOT indemnify against VFAT patents
- Drivers options:
  - SD Card (via SPI driver)
  - USB flash drive (via USB MSC host)

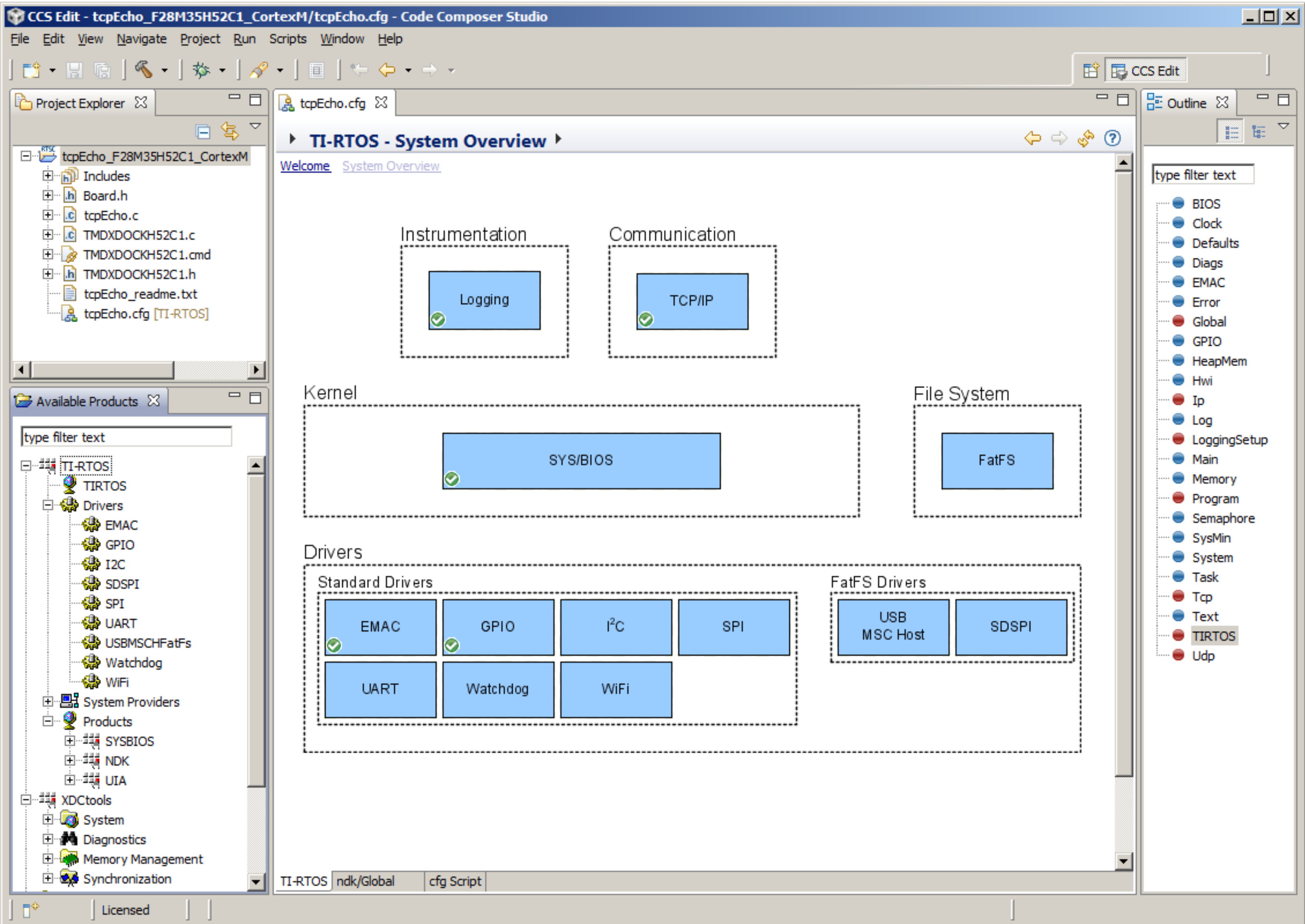
C RTS Support

- The C Run-time System consists of standard run-time libraries that are used commonly by C programs:
  - malloc (), printf (), div (), fmod (), strlen (), fopen () …
- TI-RTOS uses TI compiler C RTS library
  - This RTS has pluggable locks to support re-entrancy
    - SYS/BIOS plugs locks with appropriate SYS/BIOS mechanisms to make functions thread-safe
  - SYS/BIOS replaces malloc () with its own version
  - C RTS file-based calls are plugged to interface with FAT file system

TI-RTOS: Configuration & Debug Tools

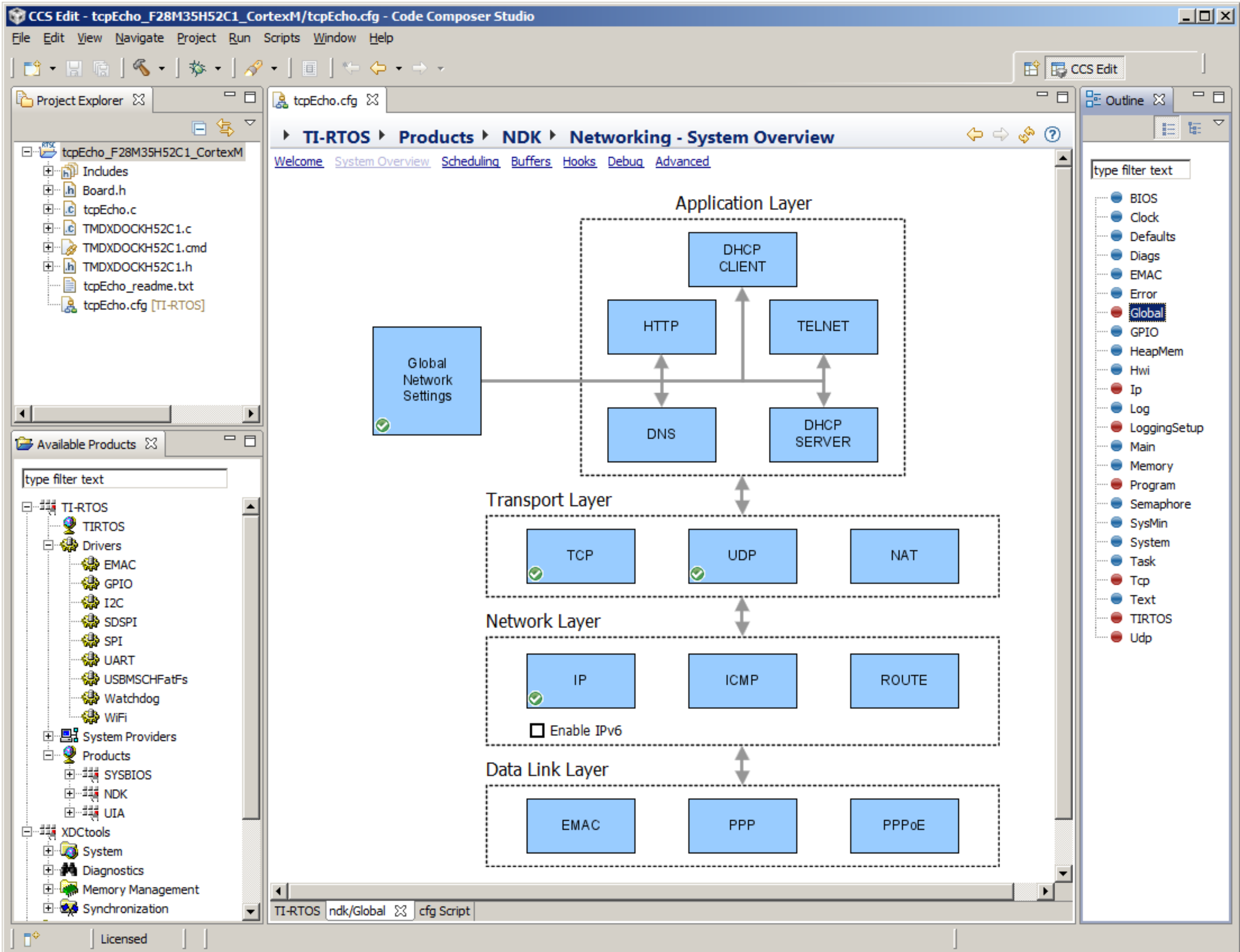


Configuration Tools: System Overview

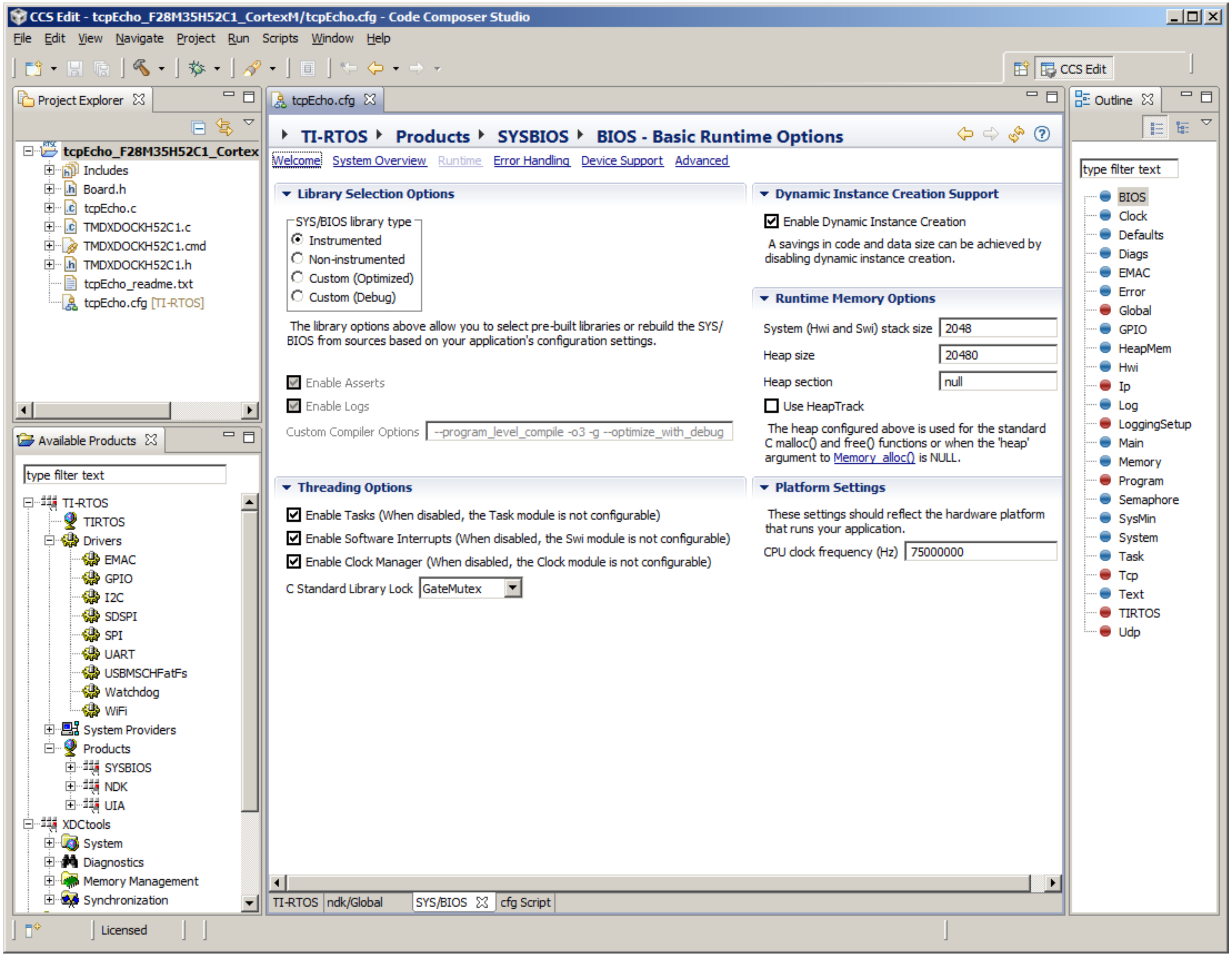


Texas Instruments

Configuration Tools: TCP/IP Overview



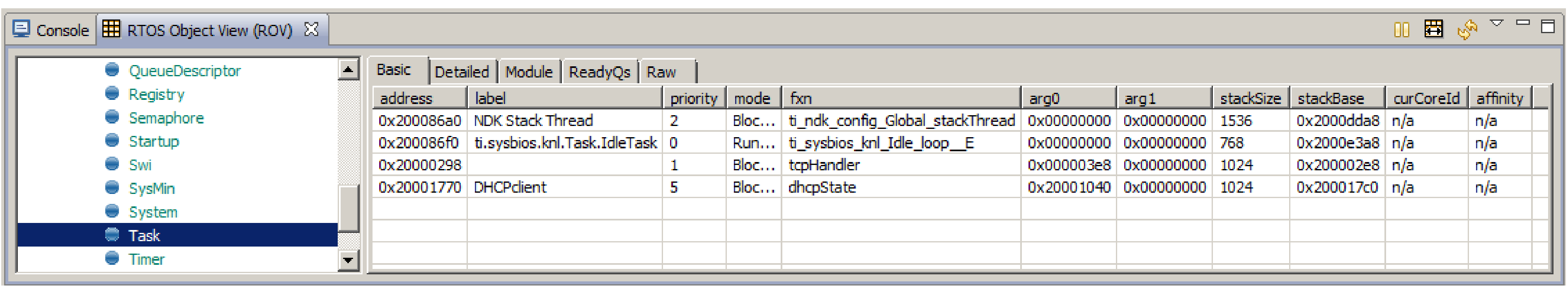
SYS/BIOS GUI Configuration Editor



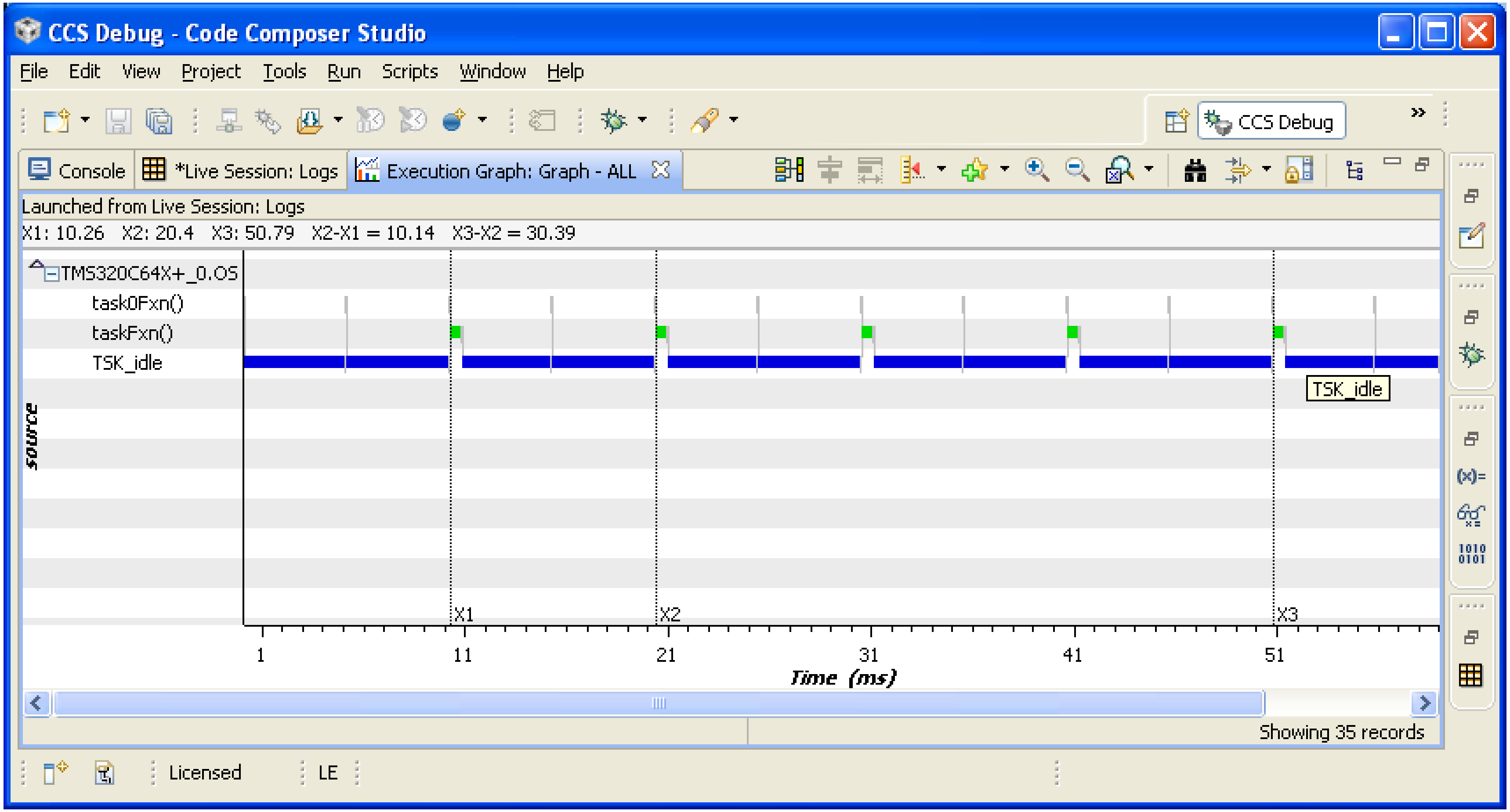
Texas Instruments

RTOS Object Viewer (ROV)

- ROV provides different views of modules. For example the Task module has the following views
  - Basic: simpler view of state for each object (instance)
  - Detailed: advanced view of state for each object
  - Module: module-wide (global) state
  - ReadyQs: List of tasks that are ready to run and the task that is running
  - Raw: all the state in an unfiltered view (much like a C struct view)
- Runs in the same Debug perspective as C source debugger



System Analyzer and Instrumentation



- Displays for thread execution timeline and CPU load

Summary

- TI-RTOS reduces our customer's time-to-market by provide commonly required software modules off-the-shelf
- TI-RTOS is integrated with Code Composer Studio
- TI-RTOS is augmented by graphical configuration and system-level debug tools
- TI-RTOS no-cost licensing removes commercial barriers to deployment

UART Lab Exercise

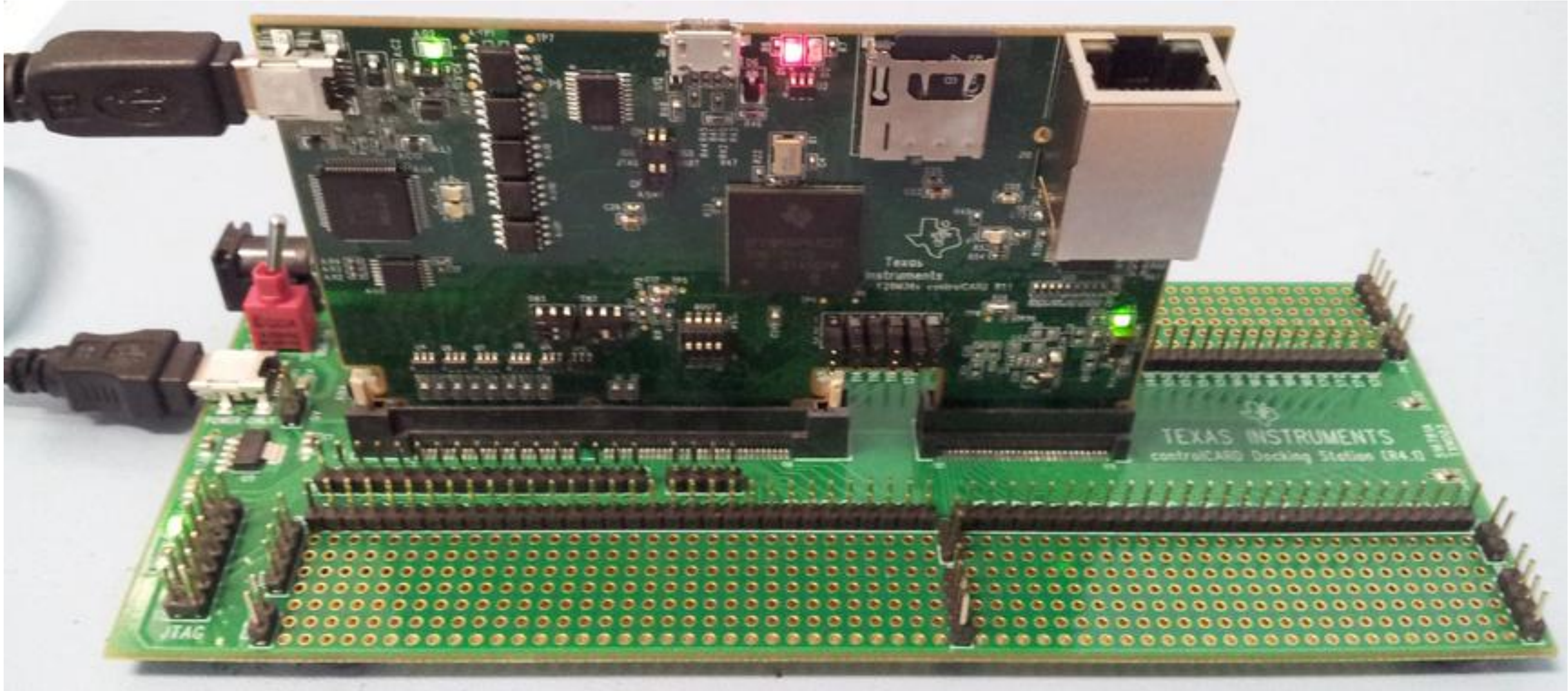
UART Lab Requirements

Software

- CCS v5.4.0 or greater
- TI-RTOS 1.10.00.23
- Terminal Plug-in in CCS (or PuTTY Terminal Emulator, HyperTerminal or TeraTerm).

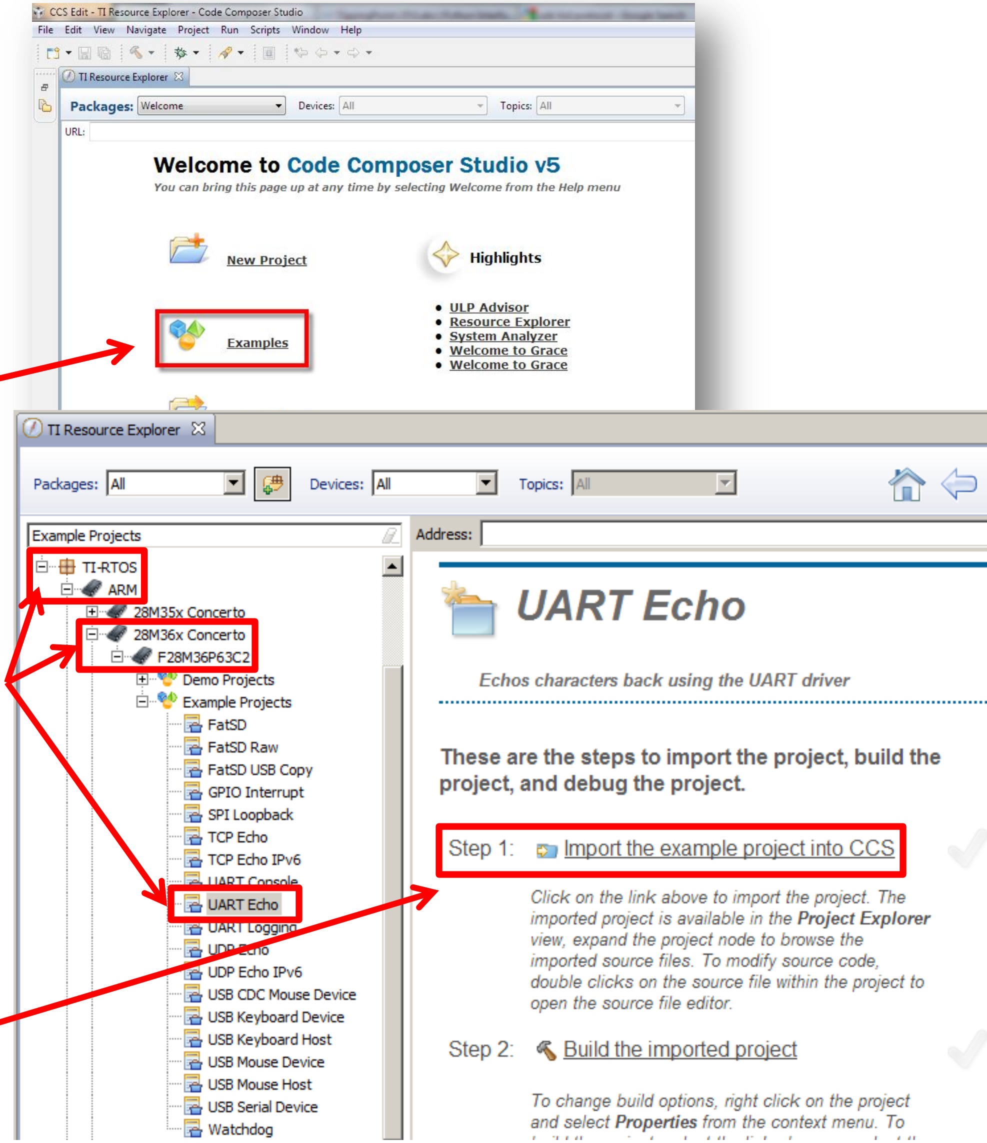
Hardware

- TMDXDOCK28M36 (shown) or TMDXDOCKH52C1 development board
- Mini-B USB for emulation and UART
- Power Supply or USB (Mini-B USB shown)

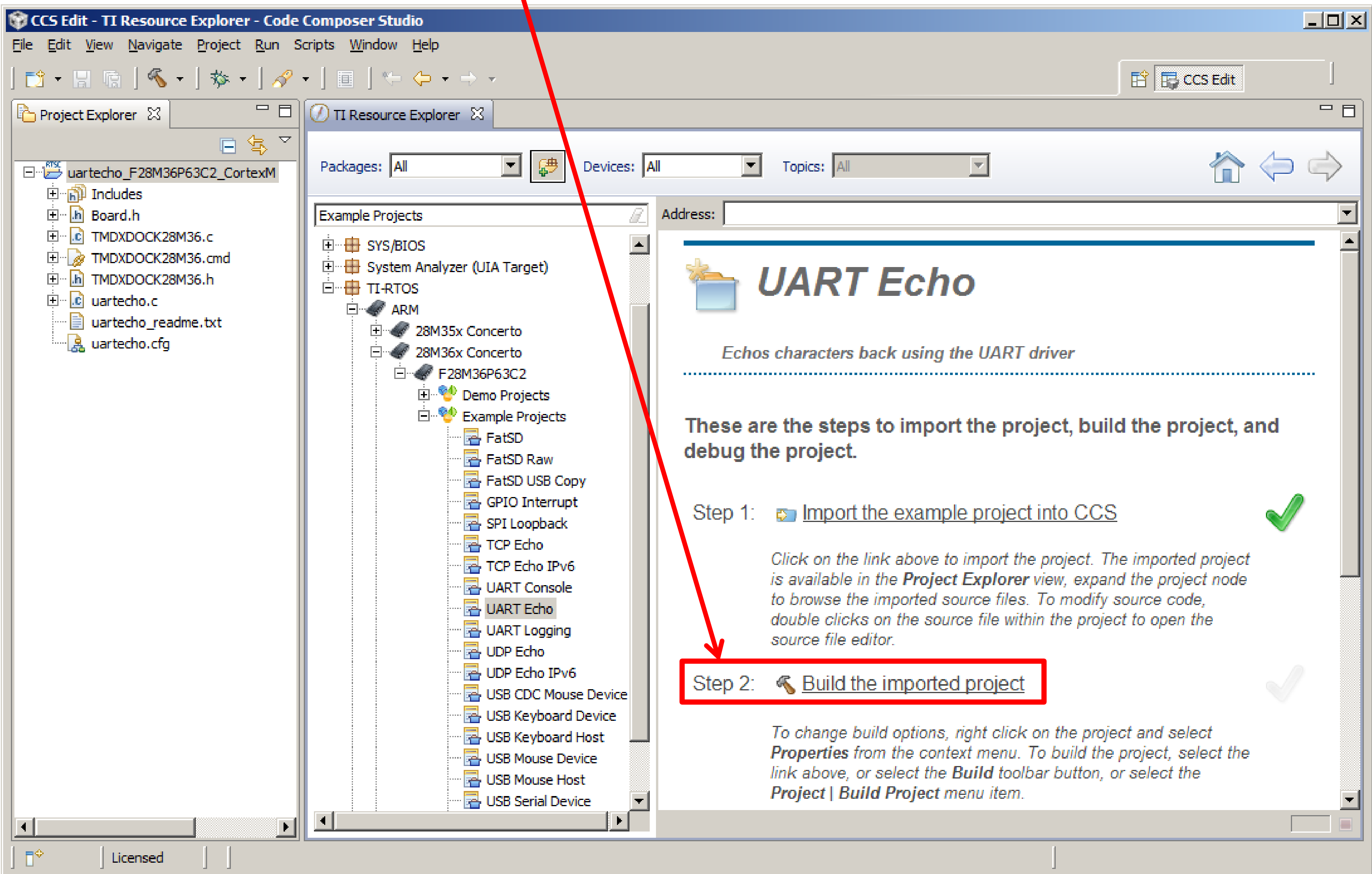


Import the example project

1. Open CCS
   – Create a new workspace e.g. “tirtos_lab”
2. Go to the Resource Explorer “Examples”
3. Import the example
   – Find the ARM’s “UART Echo” example under the TI-RTOS-> ARM->28M36x Concerto-> F28M36P63C2->Example folder.
   – Click on “Import the example into CCS”

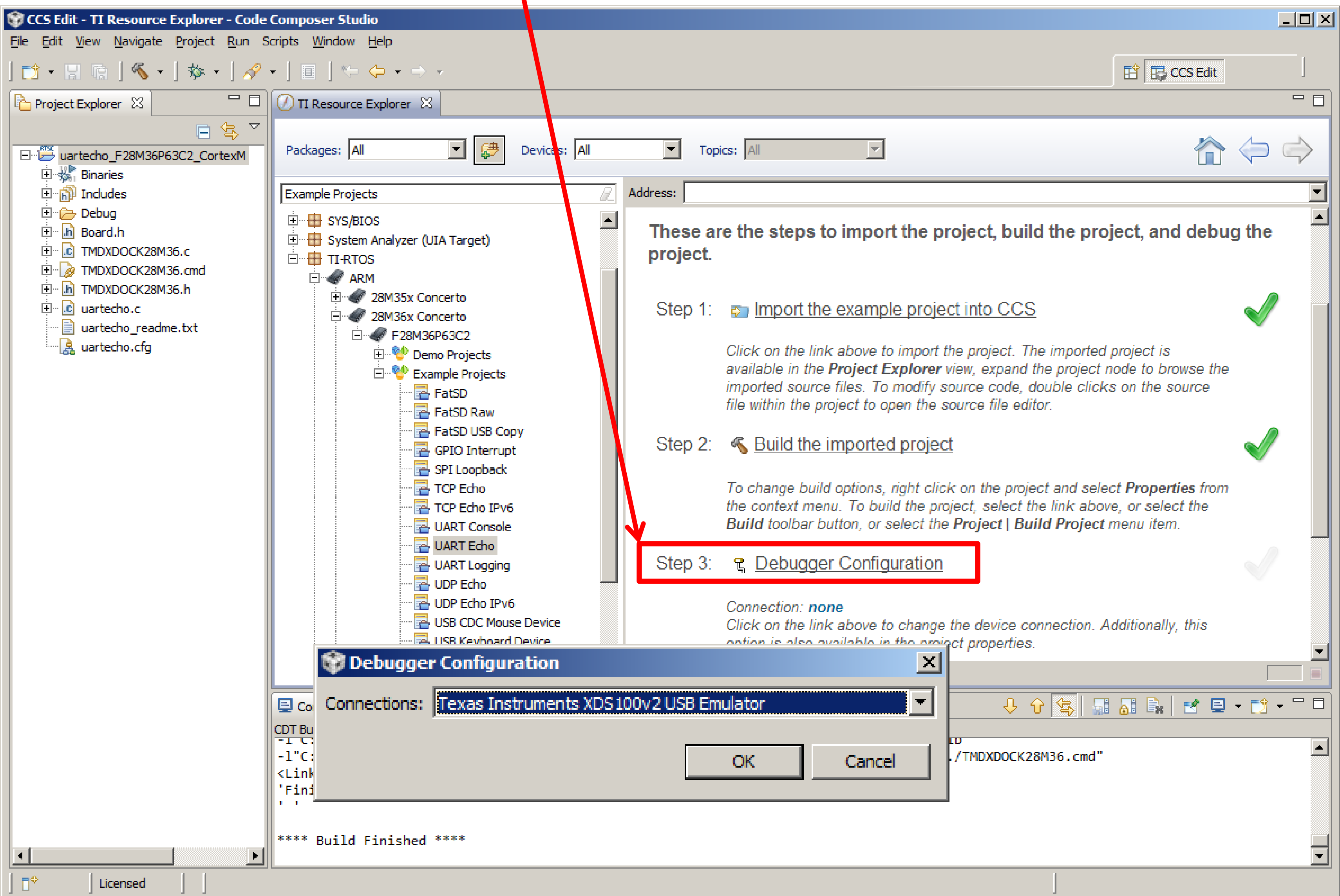


Build the example project



Texas Instruments

Setting Debugger Configuration for the example



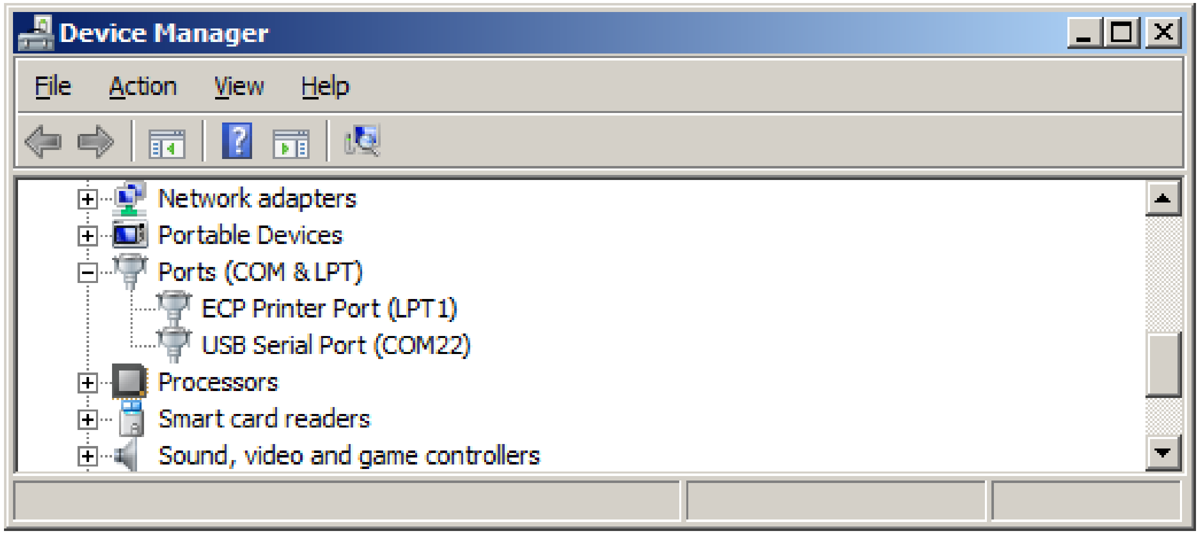
Texas Instruments

Terminal Session

CCS comes with a Terminal Plug-in.

First you need to figure out the correct port. Open the Windows “Device Manager” in Control Panel and determine the port that is being use. Note the emulation and UART are going through the same port.

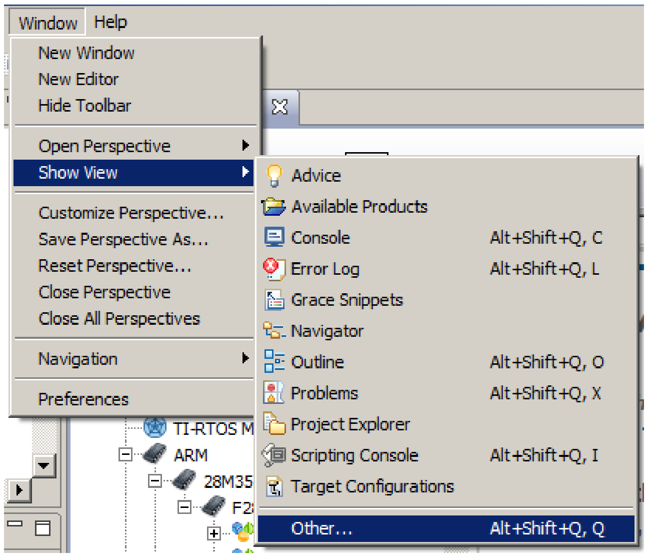
This step might not be required since the CCS Terminal Plug-in detects the COM port. However, you might have multiple COM ports active and it is good to check it.



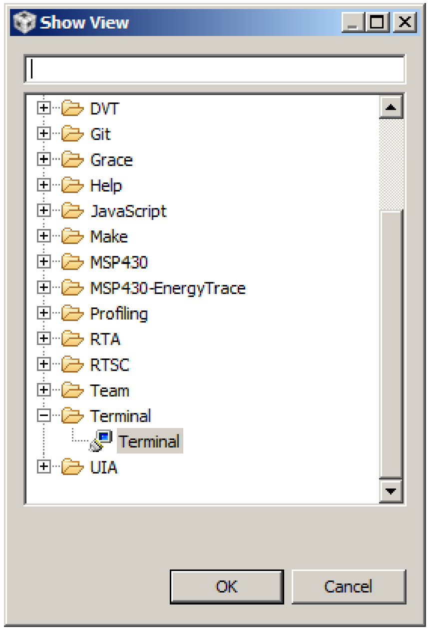
Texas Instruments

Terminal Session [cont.]

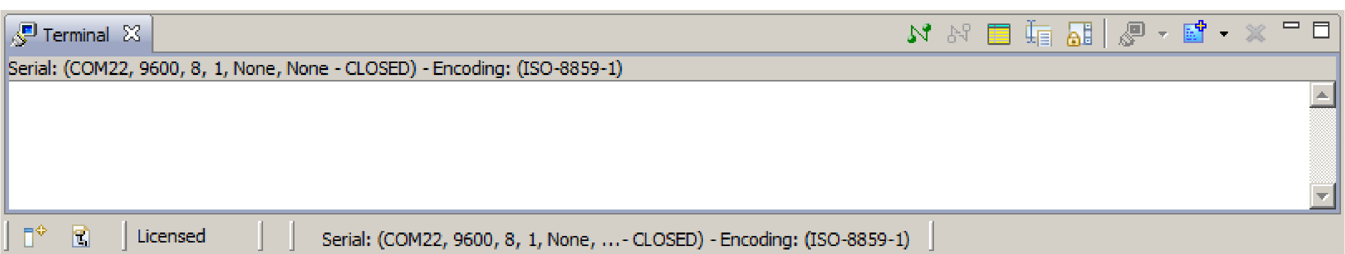
1. In CCS, select
   Window->Show View->Other…



2. Select "Terminal"

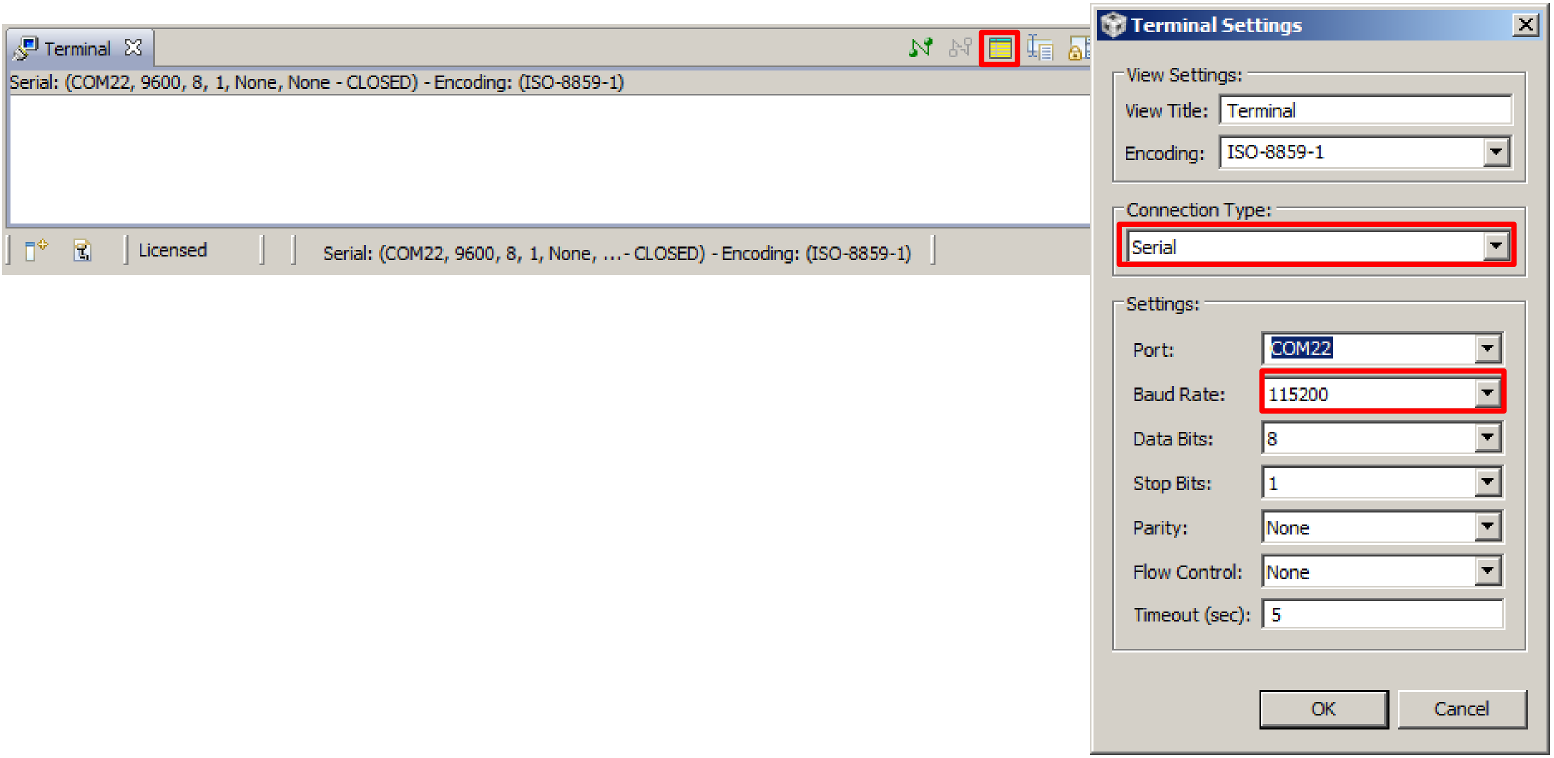


3. You'll get the following.



Terminal Session [cont.]

4. Select the “Settings” icon and set the “Connection Type” to Serial. Now set the baudrate to 115200 (this is what the application is using)

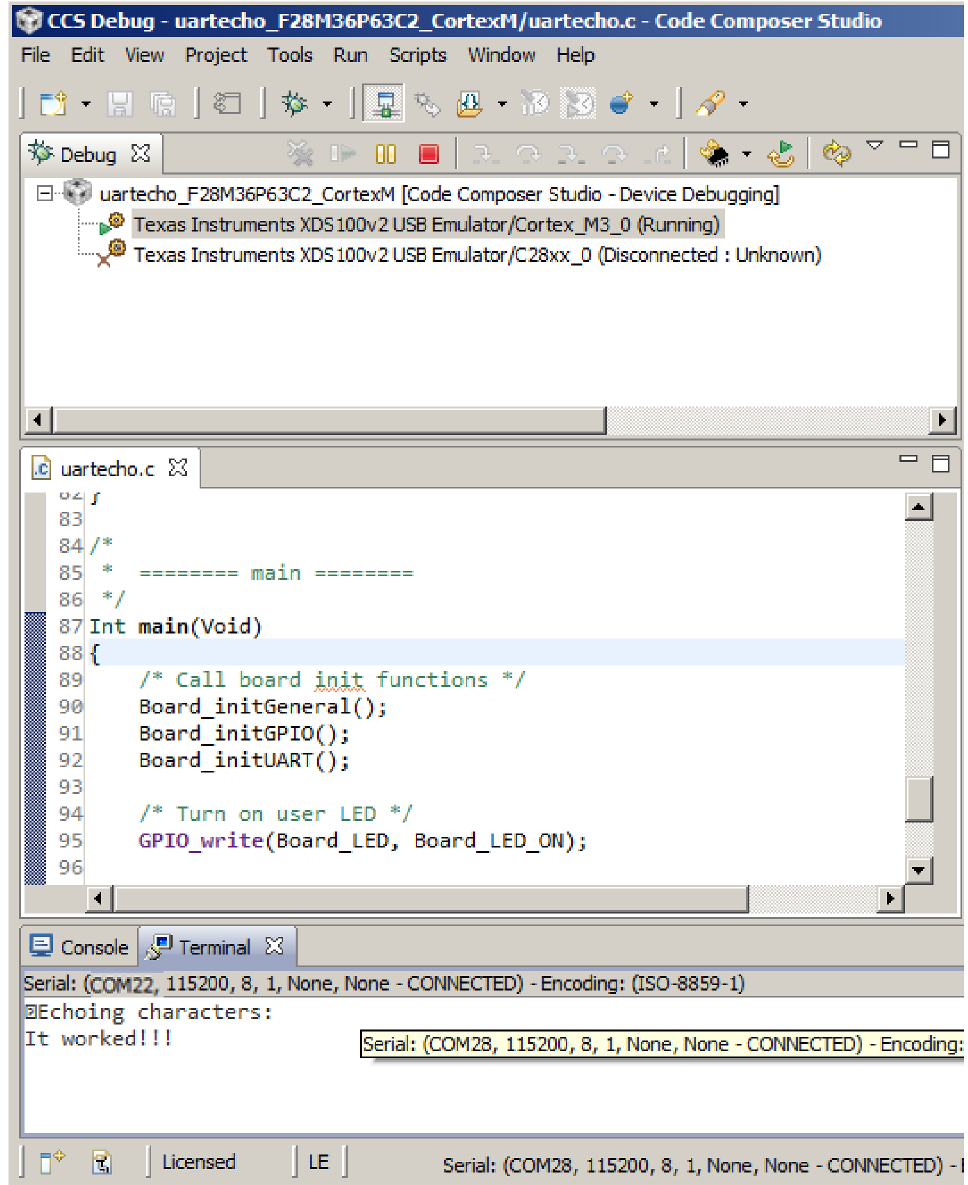


Texas Instruments

Run the example

- Click on “Debug the imported project” in Resource Explorer
- Run the example
- Verify it worked correctly by typing something in the terminal session. For example, “It worked!!!!”.

  Ok…this may or not work properly! There is a known issue in CCS with the terminal plug-in at high rates. Let’s change the baudrate to 9600 and get this to work…

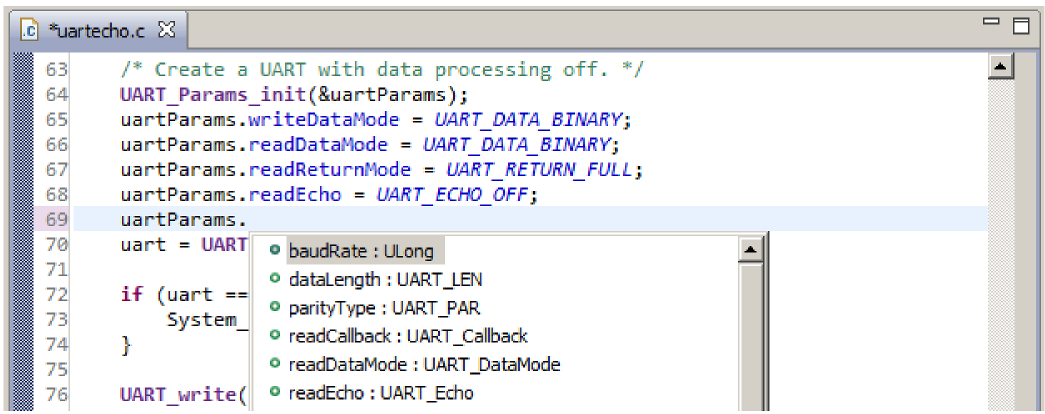


Run the example

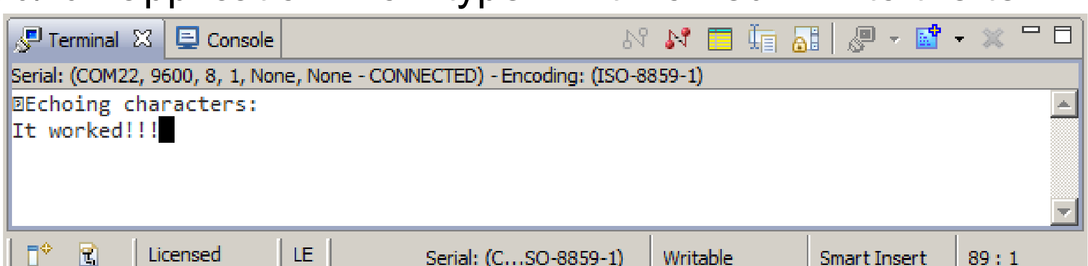
- First change disconnect your terminal session ( ) and set the baudrate to 9600bps.
- In uartecho.c, add the following line before the UART_open (uartParams.baudrate is set to the default 115200 in UART_Params_init()).

```
uartParams.baudRate = 9600;
```

Note as you type “uartParams.” CCS helps you type in the name of the field.



- Build/load/run application. Now type in “It worked!!!” into the terminal session.



Checkpoint

At this stage, you should have done the following:

- Board Setup
  - Powered up the development board
  - Connected the development board's debugger to the workstation
- Build the example
  - Imported the UART Echo example into CCS
  - Built the project with no build errors
- Run the example
  - The example echo'd back characters from CCS Terminal (or PuTTY)

Pop Quiz

Halt the processor

1. Set a break-point in the “echoFxn” while loop and run to it.
2. How much stack is the echo Task using?
3. What is the CPU Load for this app?
4. How many UARTs are in the system?
5. Which interrupts are being used in this application?

Pop Quiz Answers

Q: Set a break-point in the “echoFxn” while loop.

A: Insert a hardware breakpoint in the while loop.

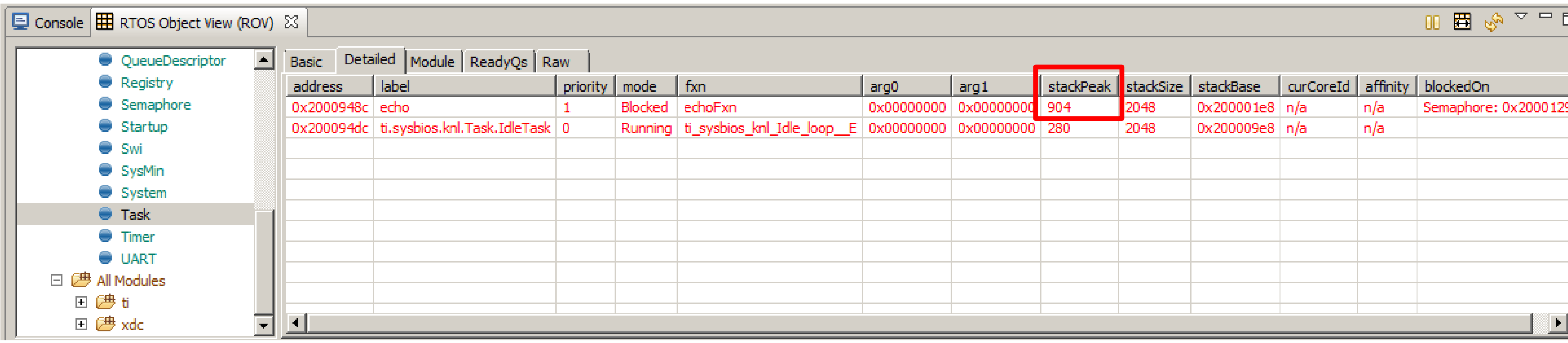
uartecho.c

```c
    UART_Params_init(&uartParams);
    uartParams.writeDataMode = UART_DATA_BINARY;
    uartParams.readDataMode = UART_DATA_BINARY;
    uartParams.readReturnMode = UART_RETURN_FULL;
    uartParams.readEcho = UART_ECHO_OFF;
    uart = UART_open(Board_UART, &uartParams);

    if (uart == NULL) {
        System_abort("Error opening the UART");
    }

    UART_write(uart, echoPrompt, sizeof(echoPrompt));

    /* Loop forever echoing */
    while (TRUE) {
        UART_read(uart, &input, 1);
        UART_write(uart, &input, 1);
    }
```

Pop Quiz Answers [cont.]

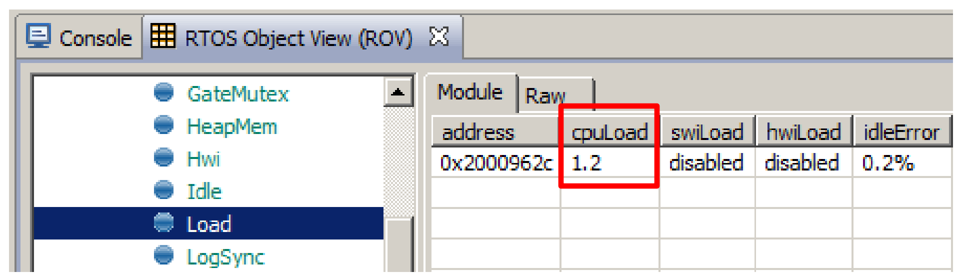
Q: How much stack is the echo Task using?

A: Halt the target and open Tools->RTOS Object Viewer (ROV). Look at the Detailed Tab on Task



Q: What is the CPU Load for this app?

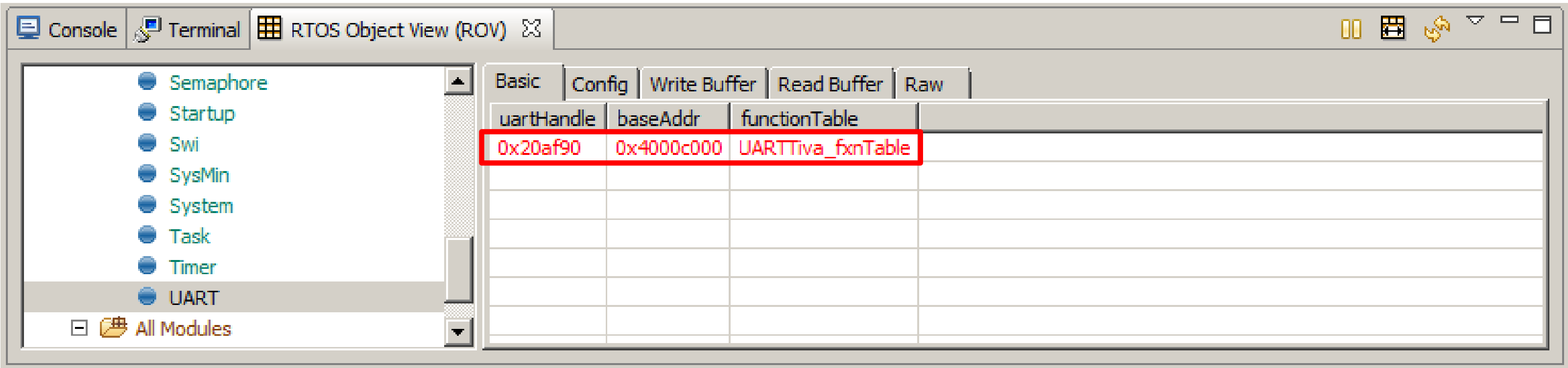
A: Look in the ROV's Load module



Pop Quiz Answers [cont.]

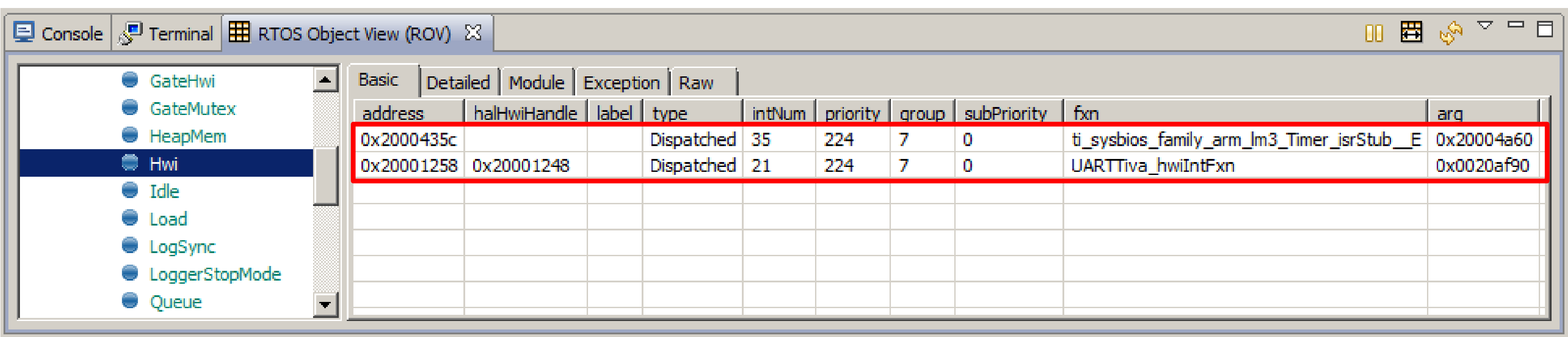
Q: How many UARTs are in the system?

A: Look in the ROV's UART module...there is one



Q: Which interrupts are being used in this application?

A: Look in the ROV's Hwi module…there are two. One for the UART and one for a timer that SYS/BIOS' Clock module uses.



Texas Instruments

Lab Summary

At this stage you should have been able to:

- Import and build the sample UART example project
- Set breakpoints
- Use ROV to view the state of the system

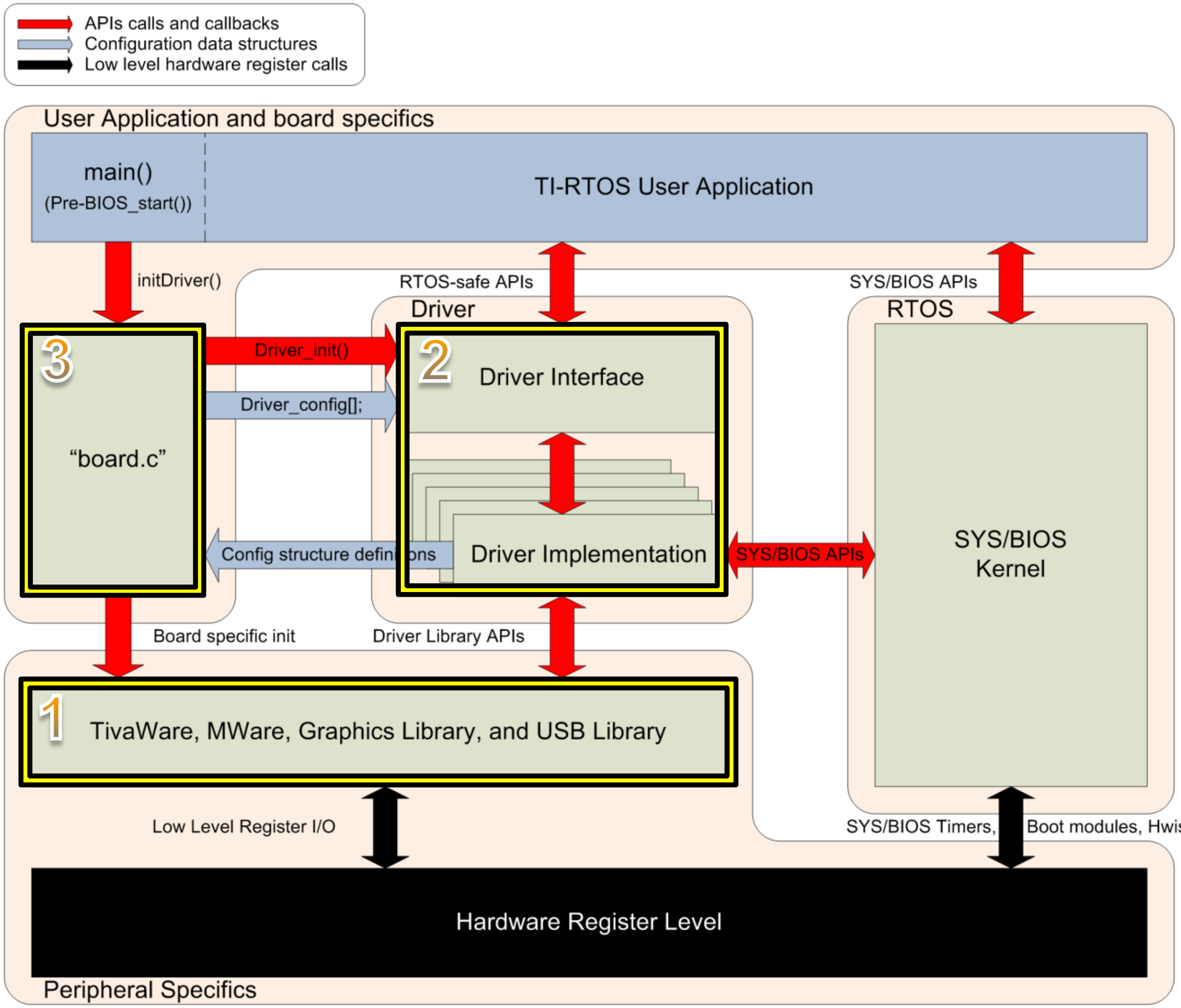
TI-RTOS “Under the Hood”

TI-RTOS Driver Details Topics

- **TI-RTOS Drivers**
  - TI-RTOS Driver Components
  - TI-RTOS Driver Structure
  - Example: I^2C driver
- Questions & Answers

TI-RTOS Driver Components

1. *Ware (TivaWare/MWare/etc.)
   Device specific
2. Driver
   Peripheral specific
3. "board.c"
   Board and Application specific



TI-RTOS Driver Components: More Details

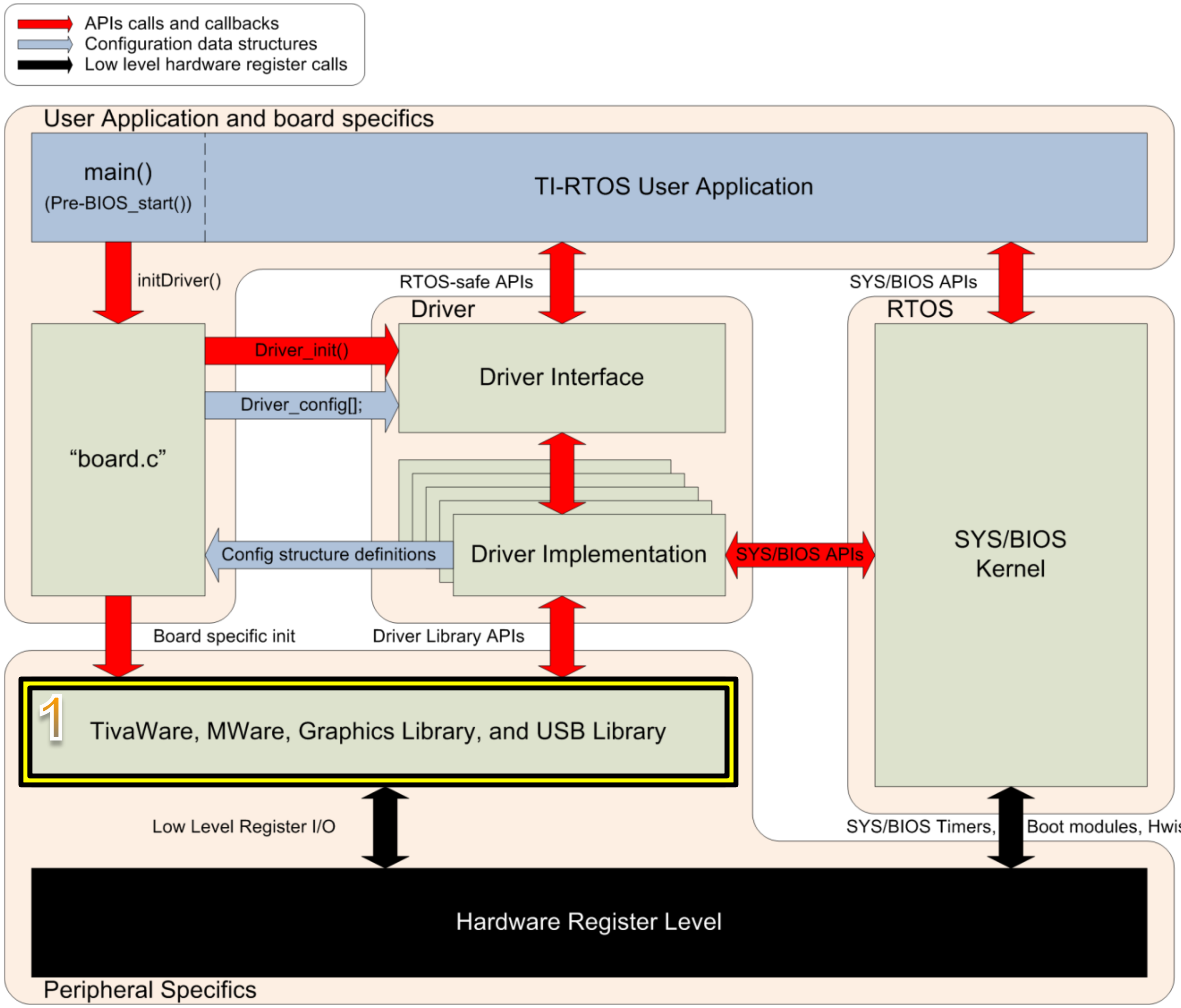
1. *Ware (TivaWare /MWare)
   - Used by the TI-RTOS drivers for direct peripheral register access
   - Shipped with TI-RTOS
2. TI-RTOS Drivers
   - Provide thread-safe APIs for SYS/BIOS tasks
   - Delivered as libraries and source
     - Instrumented: Provide Log prints
     - Non-instrumented: Compiler optimized
   - Contain support for ROV for debugging
   - Kept generic to operate with the peripheral’s IP, not device or board
3. Board/Device Specifics (Dubbed as “board.c”)
   - Customizable by the customer
   - Contain board specific settings
     - Enabling peripherals, configuring pins, pads and associated pin-muxing
     - Items unique to the board, such a driver configuration parameters
   - Supplied with the TI-RTOS examples
     - Concerto: TMDXDOCKH52C1.c
     - Stellaris LM4F232: EKS_LM4F232.c
     - etc.

TI-RTOS Driver Component: *Ware

The TI-RTOS drivers make calls to *Ware to control the peripheral's functionality.

*Ware handles register level specifics, allowing the TI-RTOS driver to be easily reused for other devices.

Please note, the *Wares (TivaWare, Mware, etc.) are products that hundreds of customers currently use today.



I^2C *Ware APIs

- These APIs are used by the I^2C driver to control the peripheral's functionality
- These handle the hardware abstraction into the peripheral's registers.

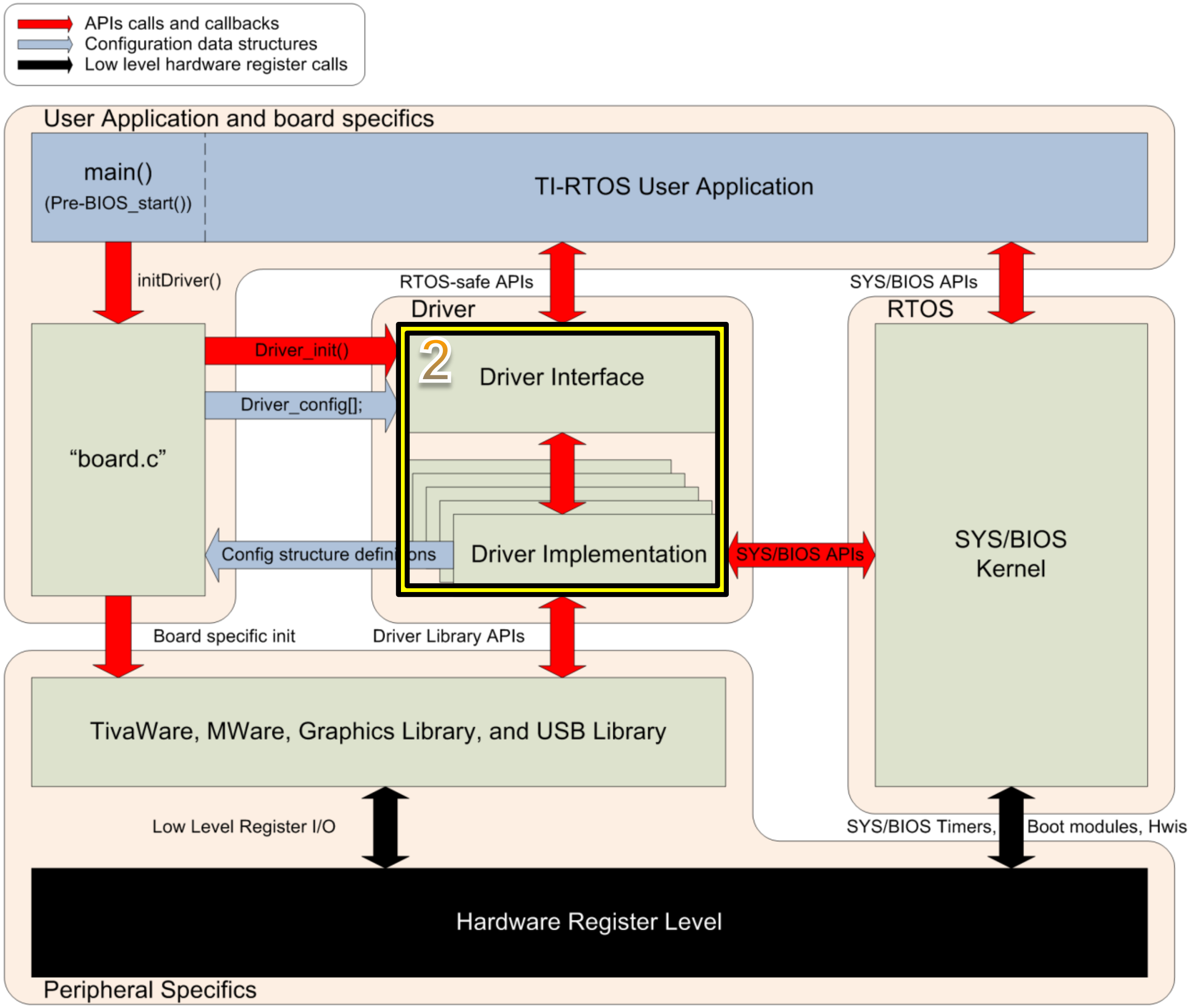
```
extern void I2CIntRegister(…)
extern void I2CIntUnregister(…);
extern tBoolean I2CMasterBusBusy(…);
extern tBoolean I2CMasterBusy(…);
extern void I2CMasterControl(…);
extern unsigned long I2CMasterDataGet(…);
extern void I2CMasterDataPut(…);
extern void I2CMasterDisable(…);
extern void I2CMasterEnable(…);
extern unsigned long I2CMasterErr(…);
extern void I2CMasterInitExpClk(…);
extern void I2CMasterIntClear(…);
extern void I2CMasterIntDisable(…);
extern void I2CMasterIntEnable(…);
extern tBoolean I2CMasterIntStatus(…);
extern unsigned long I2CMasterLineStateGet(…);
extern void I2CMasterSlaveAddrSet(…);
```

TI-RTOS Driver Component: Drivers

TI-RTOS Drivers

The TI-RTOS drivers are designed for/with:

- Peripheral control
- Easy to use and configure
- RTOS thread-safety
- Efficient RTOS scheduling
- Light on resources (e.g. RAM)
- ROV support
- Debug logging
- Support multiple instances and different types of peripherals (e.g. USCI and EUSCI UARTs)



TI-RTOS Driver Component: Driver API

Each driver* has a documented API set. These APIs are device and board independent. These APIs are available for all MCUs and EVMs to enable easy application portable.

For example, the following represents the complete API set for the I^{2}C Driver:

```
Void I2C_init(Void);     // initializes the driver’s data
                         // structures…not HW registers.
Void I2C_Params_init(I2C_Params *params);
I2C_Handle I2C_open(UInt index, I2C_Params *params);
Void I2C_close(I2C_Handle handle);
Bool I2C_transfer(I2C_Handle handle,
                  I2C_Transaction *transaction);
```

* EMAC, SDSPI and USBMSCHFatFs plug into NDK and FatFS, so there are minimal APIs.

TI-RTOS Driver Component: Driver API

Let's look at the I2C_transfer a little closer. Here is the pseudo-code for the function. Notice both SYS/BIOS and *Ware calls.

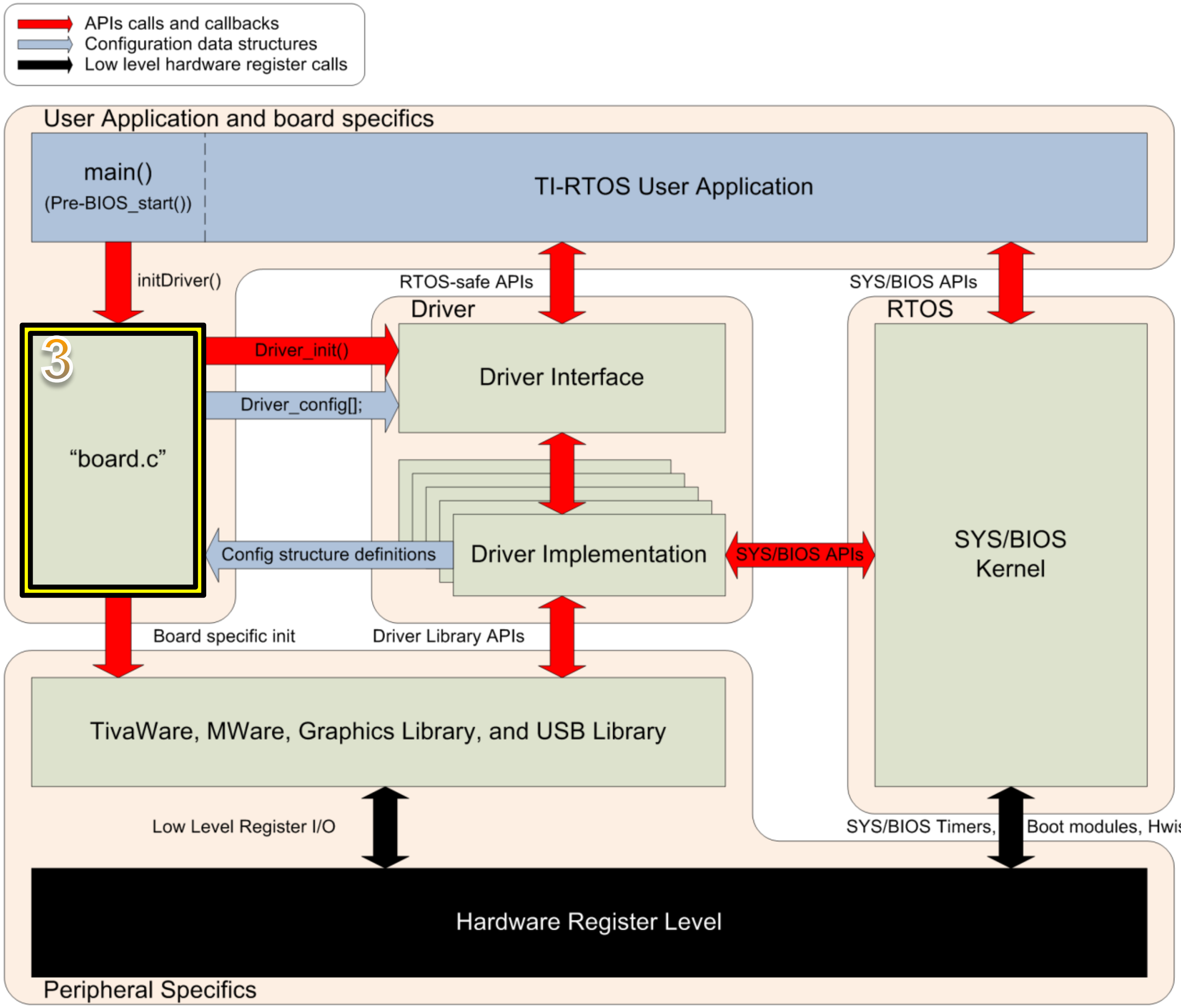
```
static Bool I2CTiva_transfer(I2C_Handle handle,I2C_Transaction *transaction)
{

    Semaphore_pend(mutex)

    I2CMasterSlaveAddrSet() // Set I2C slave address
    I2CMasterDataPut()      // Put data into the transmit data register
    I2CMasterControl()      // Start the I2C transaction

    Semaphore_pend(transferComplete) // blocked until ISR completes

    Semaphore_post(mutex)
}
```

TI-RTOS Driver Component: board.c

The “board.c” is where the application writer initializes the peripherals and then calls the TI-RTOS driver init APIs.

This file is provided with the TI-RTOS examples and can be easily modified by the customer to fit their own system needs.

When you move to your custom board, use an existing board.c as the template for starting point.



“board.c” File: Peripheral Initialization

Let’s look at the I^{2}C code in the TMDXDOCKH52C1 board.c…

Below is the I^{2}C peripheral hardware initialization for the TMDXDOCKH52C1 board…

TMDXDOCKH52C1.c

```
/*
 *  ======== TMDXDOCKH52C1_initI2C ========
 */
Void TMDXDOCKH52C1_initI2C(Void)
{
    /* I2C0 Init */
    /* Enable the peripheral */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

/* Configure the appropriate pins to be I2C instead of GPIO. */
    GPIOPinUnlock(GPIO_PORTB_BASE, GPIO_PIN_7);
    GPIOPinConfigure(GPIO_PB7_I2C0SCL); /* GPIO15 on Concerto base board */
    GPIOPinConfigure(GPIO_PB6_I2C0SDA); /* GPIO14 on Concerto base board */
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_6 | GPIO_PIN_7);

    I2C_init();
}
```

It is here where board specific initialization occurs. For example

- Peripheral clock rates
- Peripheral power control
- Pin-muxing
- etc.

“board.c” File: Driver Configuration

The “board.c” also supplies the driver configuration. Each driver requires a Driver_config structure to be specified. This structure includes object memory (the drivers do not allocate memory), hardware attributes and a function table. The function table allows for multiple types of peripherals (e.g. USCI UART and eUSCI UART) in the same system. Let’s look at the I2C configuration for TMDXDOCKH52C1…

TMDXDOCKH52C1.h

```
typedef enum TMDXDOCKH52C1_I2CName {
   TMDXDOCKH52C1_I2C0 = 0,
   TMDXDOCKH52C1_I2C1,

   TMDXDOCKH52C1_I2CCOUNT
} TMDXDOCKH52C1_I2CName;
```

TMDXDOCKH52C1.c

```
#include <ti/drivers/I2C.h>
#include <ti/drivers/i2c/I2CTiva.h>

I2CTiva_Object i2cTivaObjects[TMDXDOCKH52C1_I2CCOUNT];

const I2CTiva_HWAttrs i2cTivaHWAttrs[TMDXDOCKH52C1_I2CCOUNT] = {
    {I2C0_MASTER_BASE, INT_I2C0},
    {I2C1_MASTER_BASE, INT_I2C1}
};

const I2C_Config I2C_config[] = {
    {&I2CTiva_fxnTable, &i2cTivaObjects[0], &i2cTivaHWAttrs[0]},
    {&I2CTiva_fxnTable, &i2cTivaObjects[1], &i2cTivaHWAttrs[1]},
    {NULL, NULL, NULL} // must be null terminated
};

Void TMDXDOCKH52C1_initI2C(Void)
{
    … // board specific setup (e.g. pin-mux, etc.)
    I2C_init();
```

“board.h” Files

For each board, there is a header file that defines board specific items. For instance from the previous slide:

TMDXDOCKH52C1.h

```
typedef enum TMDXDOCKH52C1_I2CName {
   TMDXDOCKH52C1_I2C0 = 0,
   TMDXDOCKH52C1_I2C1,

   TMDXDOCKH52C1_I2CCOUNT
} TMDXDOCKH52C1_I2CName;
```

For all the examples, there is a file called board.h that allows the examples to generically call Board_xxx APIs and use Board_yyy defines. Here is the board.h file the

Board.h

```
#include "TMDXDOCKH52C1.h"

#define Board_initGeneral            EKS_LM4F232_initGeneral
#define Board_initGPIO               EKS_LM4F232_initGPIO
…
#define Board_I2C_EEPROM             TMDXDOCKH52C1_I2C0
#define Board_I2C_TMP                TMDXDOCKH52C1_I2C1
…
```

Exercise: Add another I^2C peripheral

Exercise: For this exercise we are going to add a third I^2C to a EKS_LM4F232 example and communicate to it.

The “board.c” files for the different boards come with the peripheral already configured, but not all of them. This is because some might be mutually exclusive. The EKS_LM4F232.c board file has 2 I^2C configured.

Goal: We are not going to build or run the example. This is to show what the steps would be though!

Adding a Driver

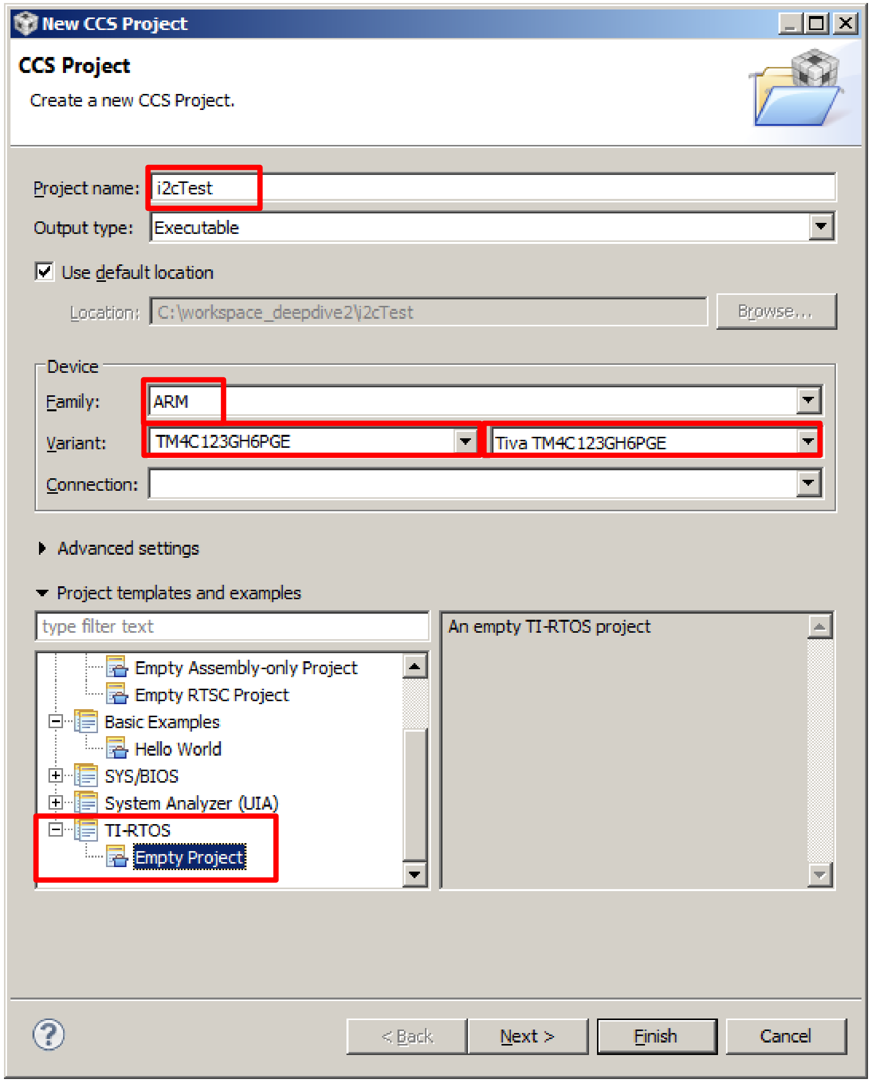
Goal: Make an application that talks to the third I2C peripheral…

First let's create an empty project via the CCS Project Wizard.

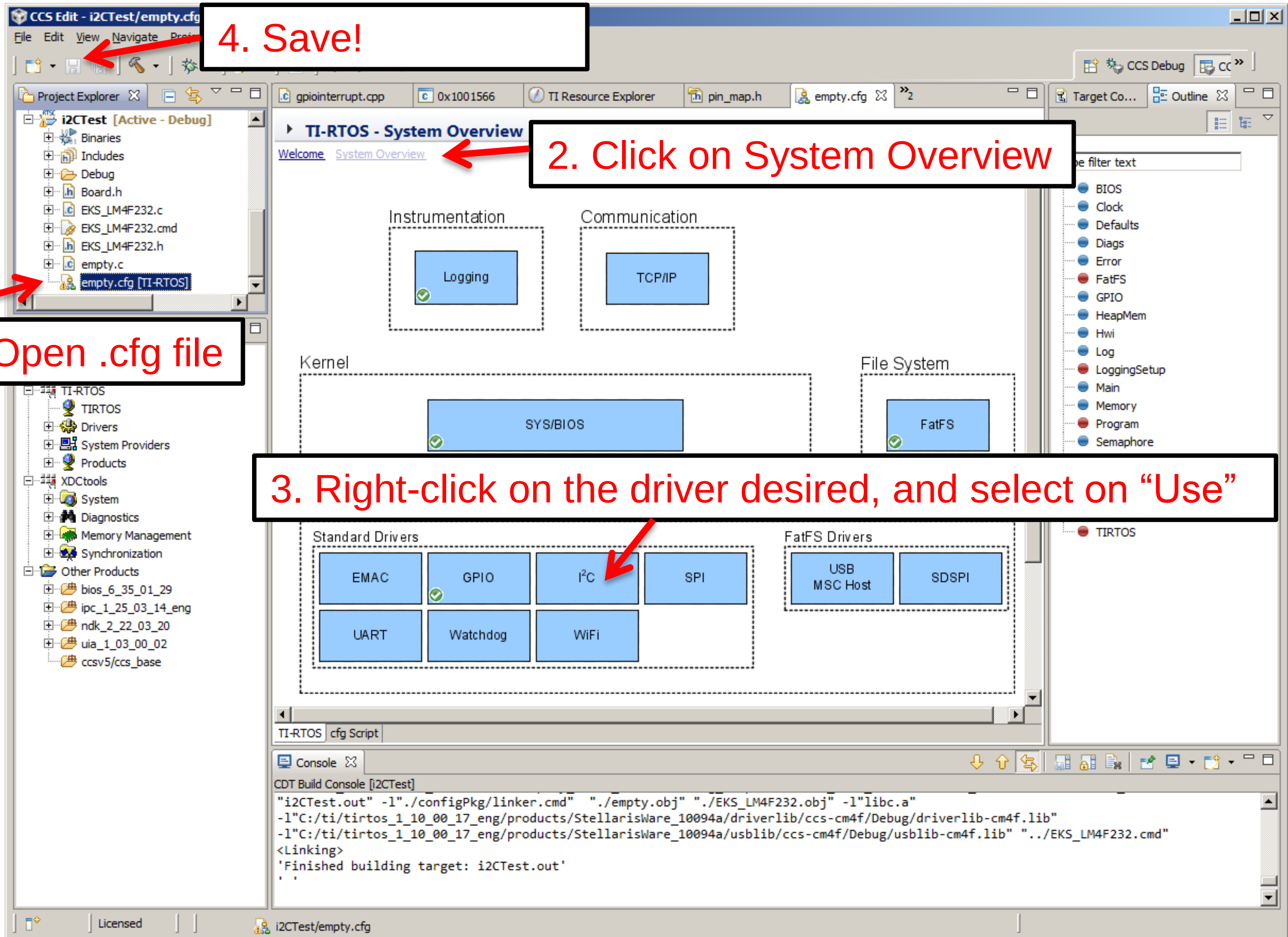
Doing "File->New->CCS Project" will start the wizard on the right.

You need to

1. Give the project a unique name (e.g. i2cTest)
2. Select the Family and Variant (type TM4C123GH6PGE in the filter help find the device).
3. Select the TI-RTOS Empty project
4. Hit "Finish"



Adding Drivers into Configuration



Texas Instruments

Adding an I^2C Peripheral

- In empty.c, remove comments from the following **bolded** lines

empty.c

```
#include <ti/drivers/I2C.h>

Int main(Void)
{
    /* Call board init functions */
    Board_initGeneral();
    // Board_initEMAC();
    Board_initGPIO();
    Board_initI2C();
```

Let's look at the board.c and board.h on next slide…

Adding an I²C Peripheral Configuration

EKS_LM4F232.h: let's add in a third I2C peripheral (in bold)

```
/*!
 *  @def    EKS_LM4F232_I2CName
 *  @brief  Enum of I2C names on the EKS_LM4F232 dev board
 */
typedef enum EKS_LM4F232_I2CName {
    EKS_LM4F232_I2C0 = 0,
    EKS_LM4F232_I2C2,
    EKS_LM4F232_I2C3,
    EKS_LM4F232_I2CCOUNT
} EKS_LM4F232_I2CName;
```

EKS_LM4F232.c: Update the I2C_config and I2CTiva_HWAttrs (in **bold**)

```
/* I2C objects */
I2CTiva_Object i2cTivaObjects[EKS_LM4F232_I2CCOUNT];

/* I2C configuration structure, describing which pins are to be used */
const I2CTiva_HWAttrs i2cTivaHWAttrs[EKS_LM4F232_I2CCOUNT] = {
    {I2C0_MASTER_BASE, INT_I2C0},{I2C1_MASTER_BASE, INT_I2C2}, {I2C3_MASTER_BASE, INT_I2C3}};

const I2C_Config I2C_config[] = {
    {&I2CTiva_fxnTable, &i2cTivaObjects[0], &i2cTivaHWAttrs[0]},
    {&I2CTiva_fxnTable, &i2cTivaObjects[1], &i2cTivaHWAttrs[1]},
    {&I2CTiva_fxnTable, &i2cTivaObjects[2], &i2cTivaHWAttrs[2]},
    {NULL, NULL, NULL}
};
```

Adding an I²C Peripheral Initialization

- Add in the initialization code (in **bold**)

EKS_LM4F232.c

```
Void EKS_LM4F232_initI2C(Void)
{
    /* I2C0 Init */
    …

    /* I2C1 Init */
    …

    /* I2C3 Init */
    /* Enable the peripheral */
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C3);
    /* Configure the appropriate pins to be I2C instead of GPIO. */
    GPIOPinConfigure(GPIO_PG0_I2C3SCL);
    GPIOPinConfigure(GPIO_PD1_I2C3SDA);
    GPIOPinTypeI2CSCL(GPIO_PORTG_BASE, GPIO_PIN_0);
    GPIOPinTypeI2C(GPIO_PORTD_BASE, GPIO_PIN_1);

    I2C_init();
}
```

Adding an I²C Application

- Finally add the application code (pseudo-code) in your application file

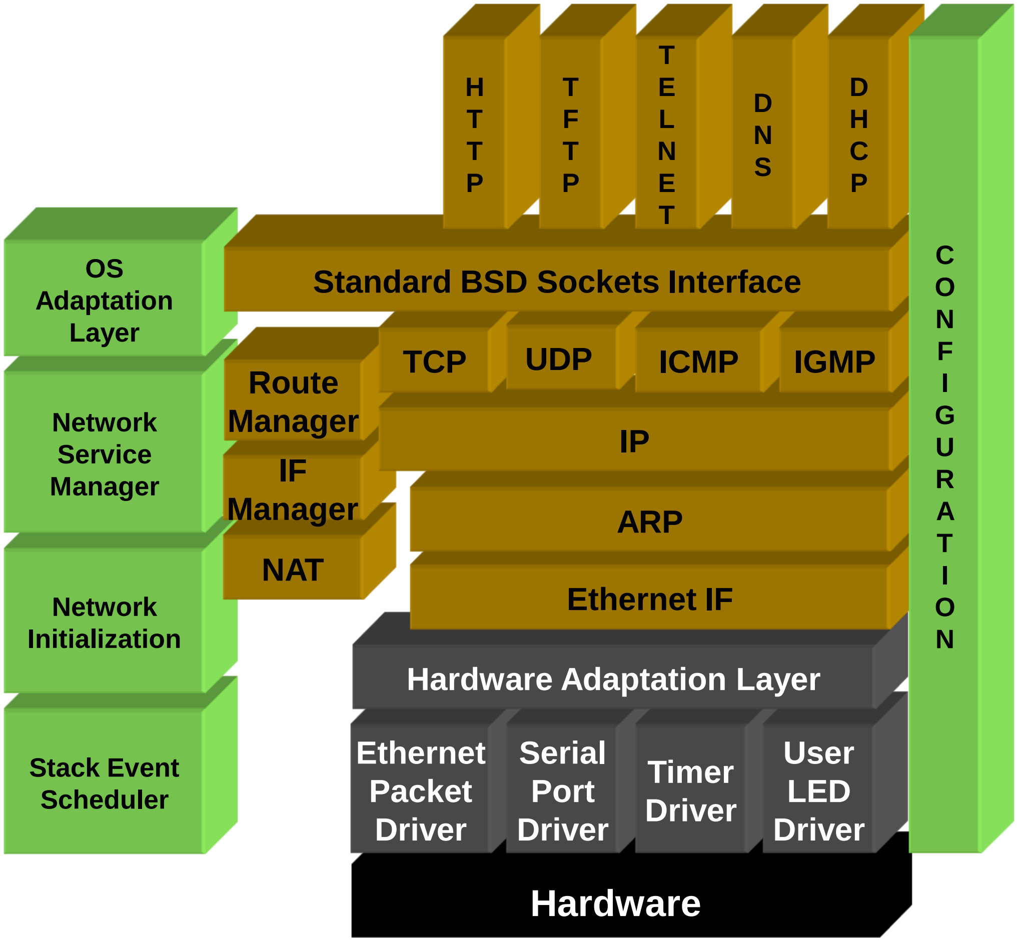
empty.c

```
#include <ti/drivers/I2C.h>
Void taskFxn(UArg arg0, UArg arg1)
{
    I2C_Transaction i2cTransaction;
    I2C_Handle      i2c;
    I2C_Params      i2cParams;

    /* Open the driver */
    I2C_Params_init(&i2cParams);
    i2cParams->transferMode = I2C_MODE_BLOCKING;
    i2cHandle = I2C_open(EKS_LM4F232_I2C2, &i2cParams);

    /* Perform I2C transfer */
    i2cTransaction.slaveAddress = slaveAddress;
    i2cTransaction.writeBuf = transmitBuffer;
    i2cTransaction.writeCount = txAddressLength + txDataLength;
    i2cTransaction.readBuf = NULL;
    i2cTransaction.readCount = 0;
    transferOK = I2C_transfer(i2cHandle, &i2cTransaction);
    …
}
```

Questions & Answers & Break!

HTTP Server Lab Exercise

TCP/IP Stack



TCP/IP Key Features

- ✓ Supports both IPv4 and IPv6
- ✓ DHCP Client and Server
- ✓ HTTP Server
- ✓ Standard BSD Sockets interface
- ✓ Zero-copy sockets interface available
- ✓ Highly configurable to meet footprint constraints
- ✓ Example on how to create daemon task required by TCP/IP stacks

Let’s Make a WebServer!

Steps

- **Create and build new “empty” project**
- Load and debug the empty project
- Add in Networking stack
- Add HTTP pages

Creating an Empty Project

First let's create an empty project via the CCS Project Wizard.

Doing "File->New->CCS Project" will start the wizard on the right.

You need to

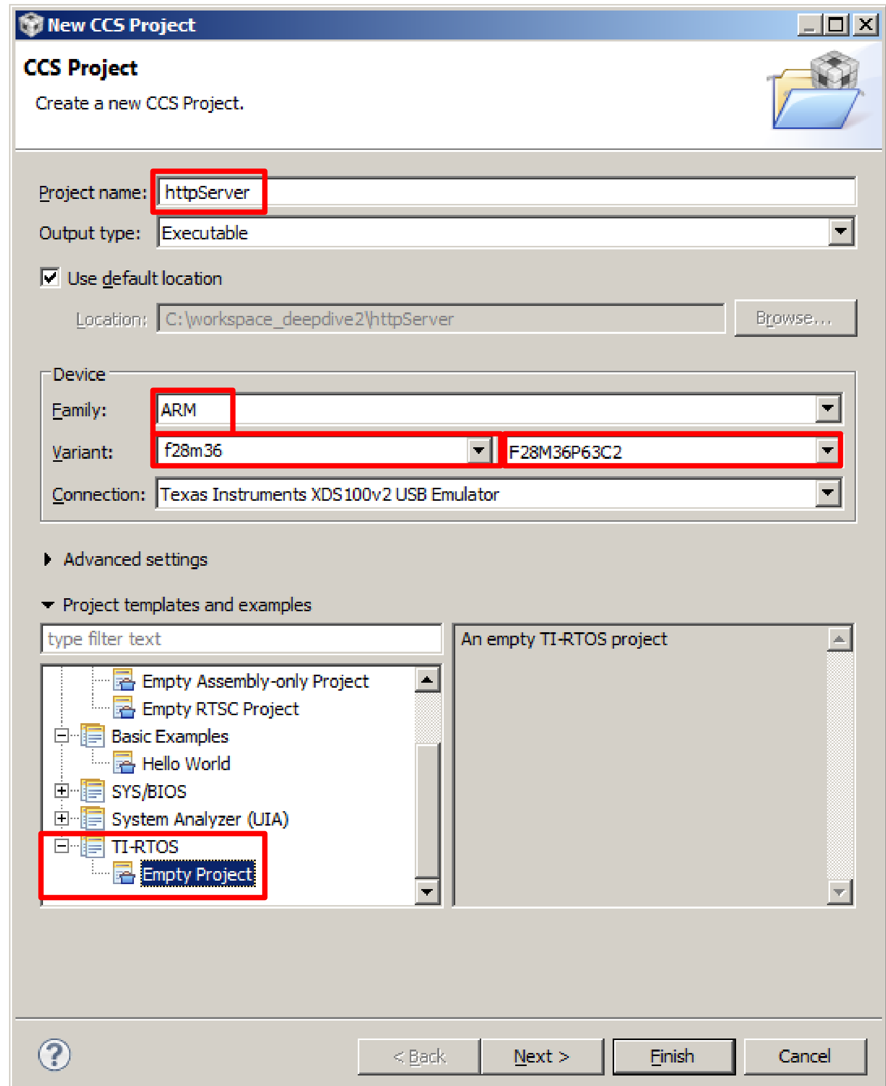
1. Give the project a unique name (e.g. "httpServer").

2. Select the Family and Variant (type f28m36 in the filter to help find the F28M36P63C2 device)

3. Select the Connection (optional, but we'll use this later)

4. Select the TI-RTOS Empty project

Hit "Next"

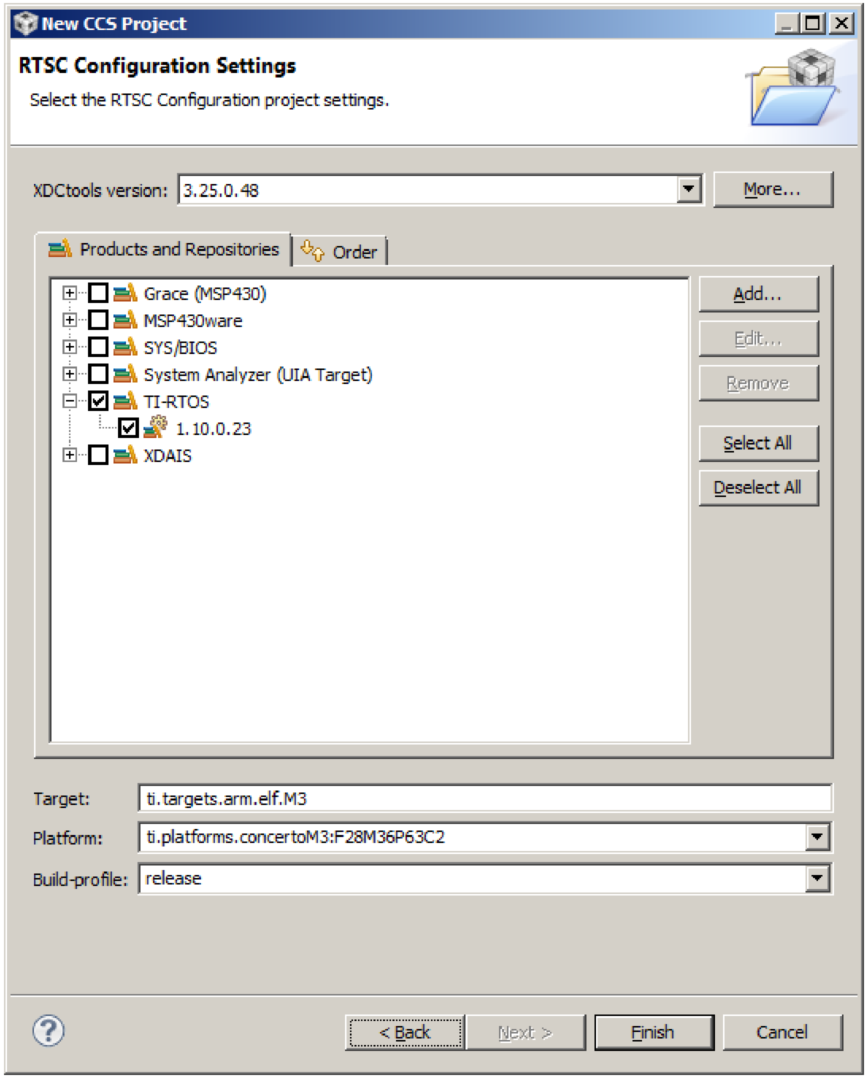


Creating an Empty Project

This page shows which software is being used.

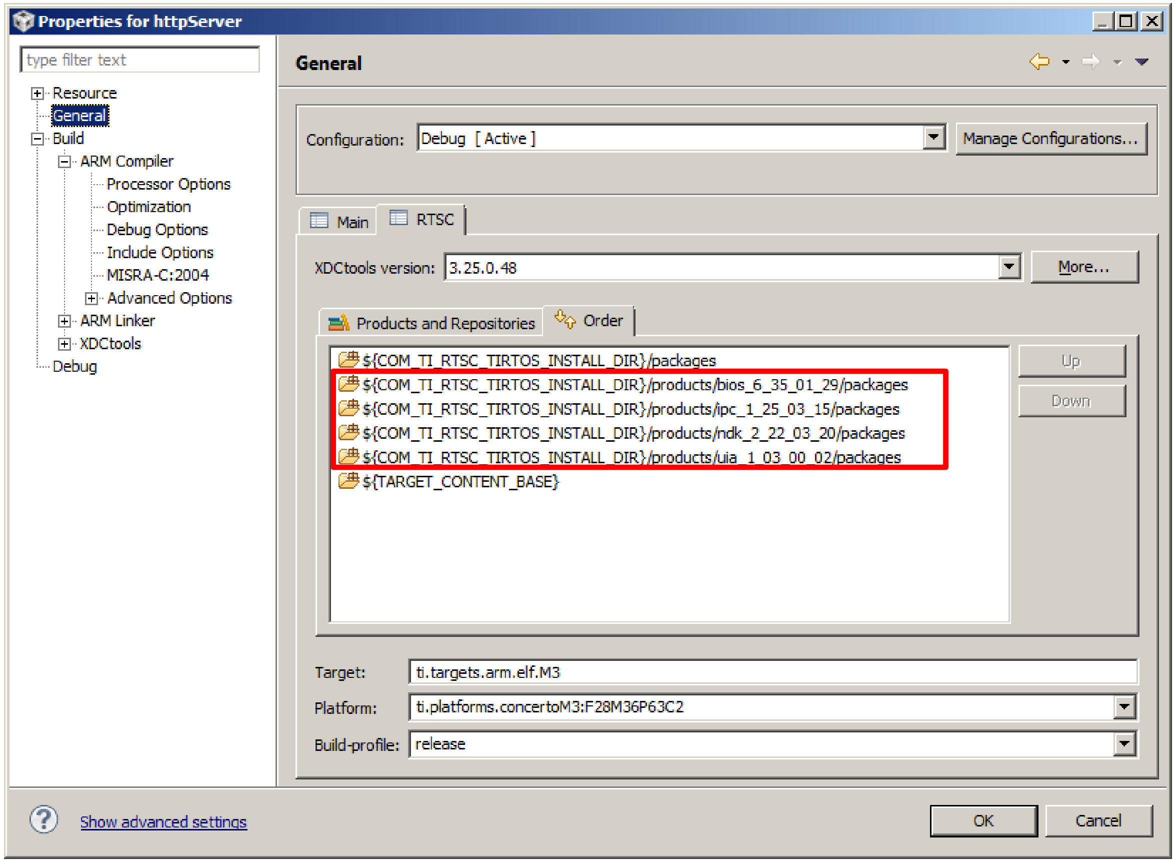
Hit “Finish”

Note: Only TI-RTOS is specified…why not SYS/BIOS, etc.?



TI-RTOS Components

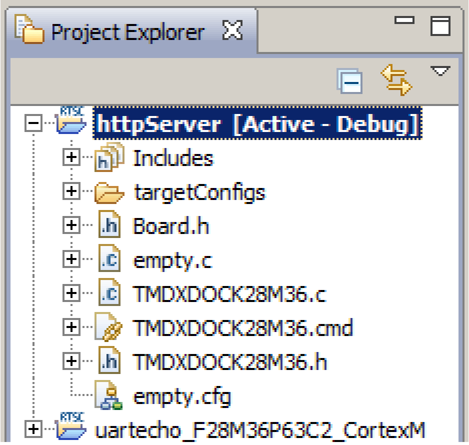
The other products are included in the path by TI-RTOS.



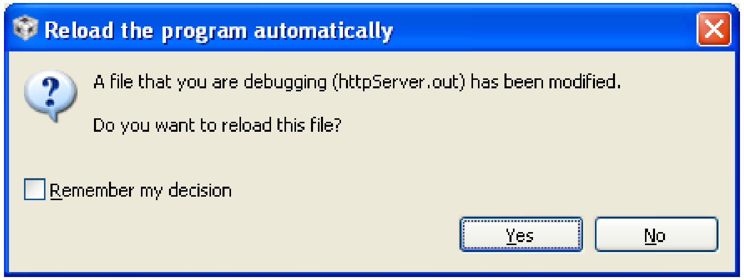
(Snapshot of "Project Properties" of httpServer project)

Building the httpServer Project

- Let's look at the files in the new project
  - empty.c: main source file
  - empty.cfg: main configuration file
  - TMDXDOCK28M36.c: board specific peripheral runtime configuration
  - TMDXDOCK28M36.cmd: Linker command file
  - TMDXDOCK28M36.h: header file for board specific peripheral runtime configuration APIs
  - Board.h: Small "shim" header file to make board specific APIs generic.



- Build project (right click, Project menu or build icon )
  - Build output goes to the console window
  - After the build is successful, if asked, do not load the application yet.



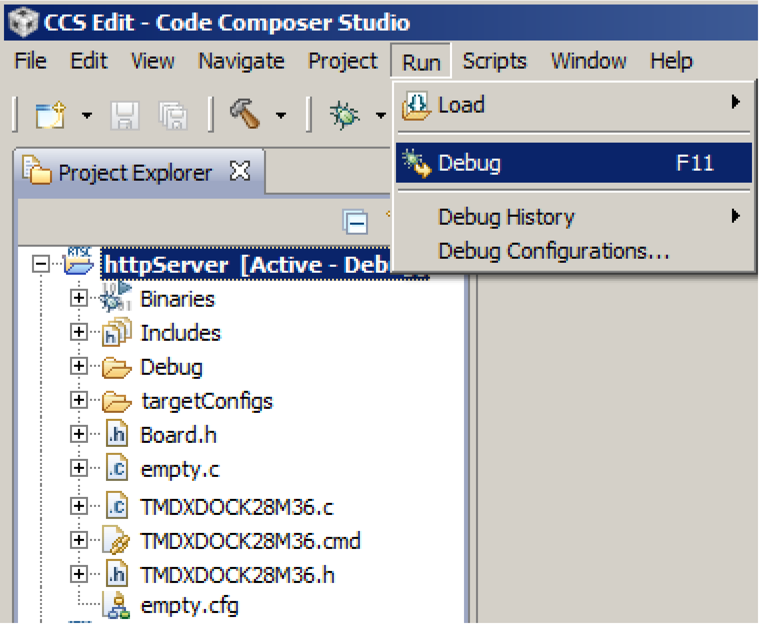
Let’s Make a WebServer!

Steps

- Create and build new “empty” project
- **Loading and debugging the empty project**
- Add in Networking stack
- Add HTTP pages

Loading the Project

- Debug the application.
  - Launch the Debug session (F11, Run->Debug, or Debug icon)



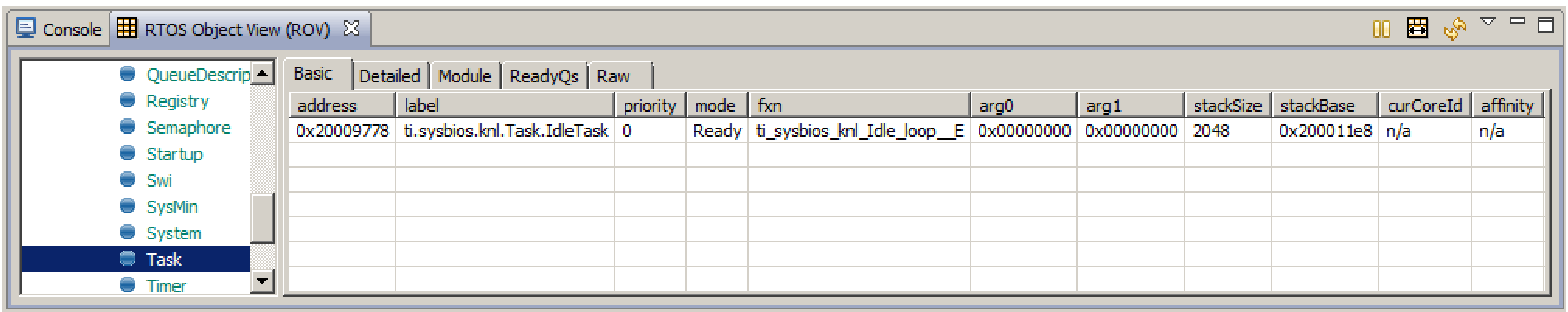
Run the project

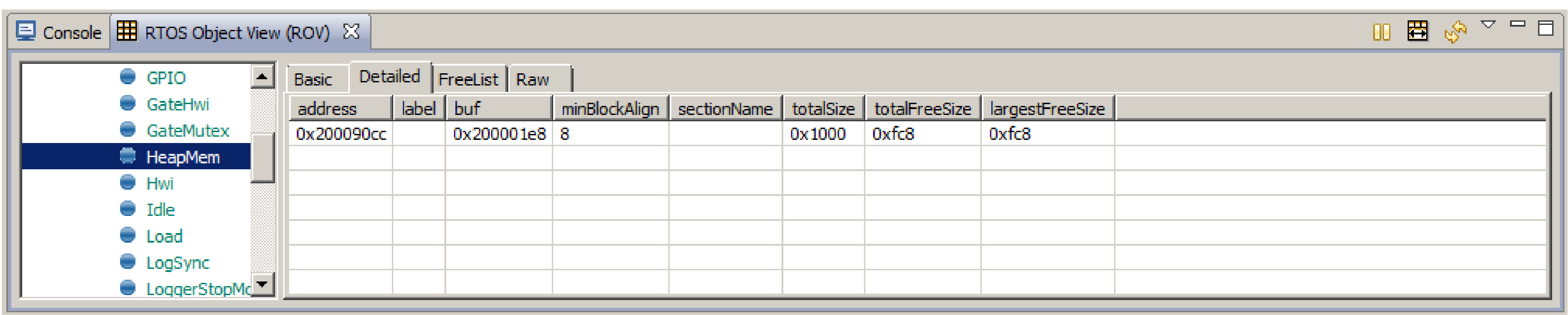
- Run.

You'll probably halt in main(). This is configurable via the Tools->Debugger Options->Auto Run and Launch Options

- Open RTOS Object View (Tools->ROV)

Look at Task and HeapMem->Detailed (shown below) (note: you have to halt the CPU for ROV to update).





Let’s Make a WebServer!

Steps

- Create and build new “empty” project
- Loading and debugging the empty project
- **Add in Networking stack**
- Add HTTP pages

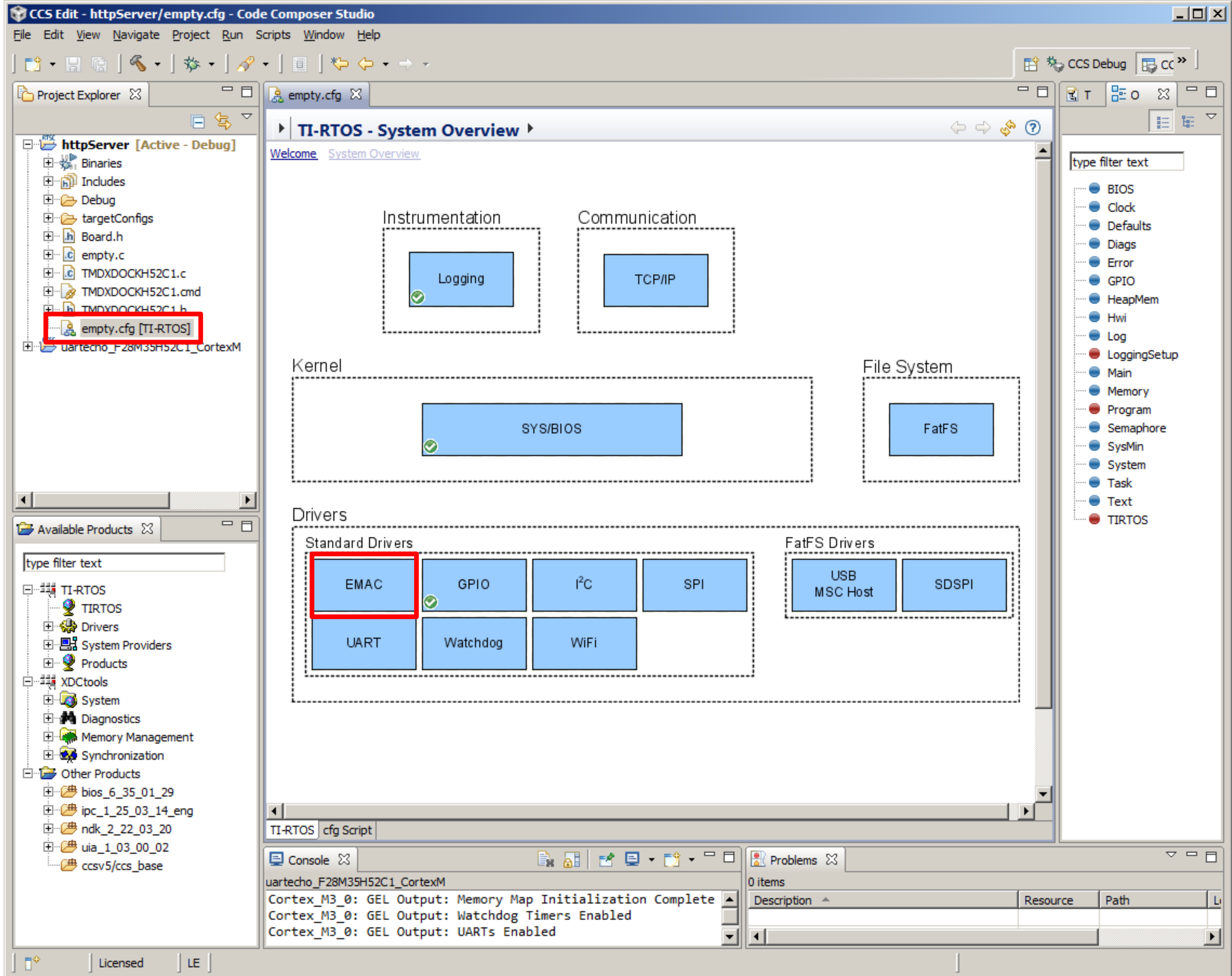
Adding Networking Stack

- Open up empty.cfg and view System Overview

- Add EMAC by right clicking and selecting “Use EMAC”

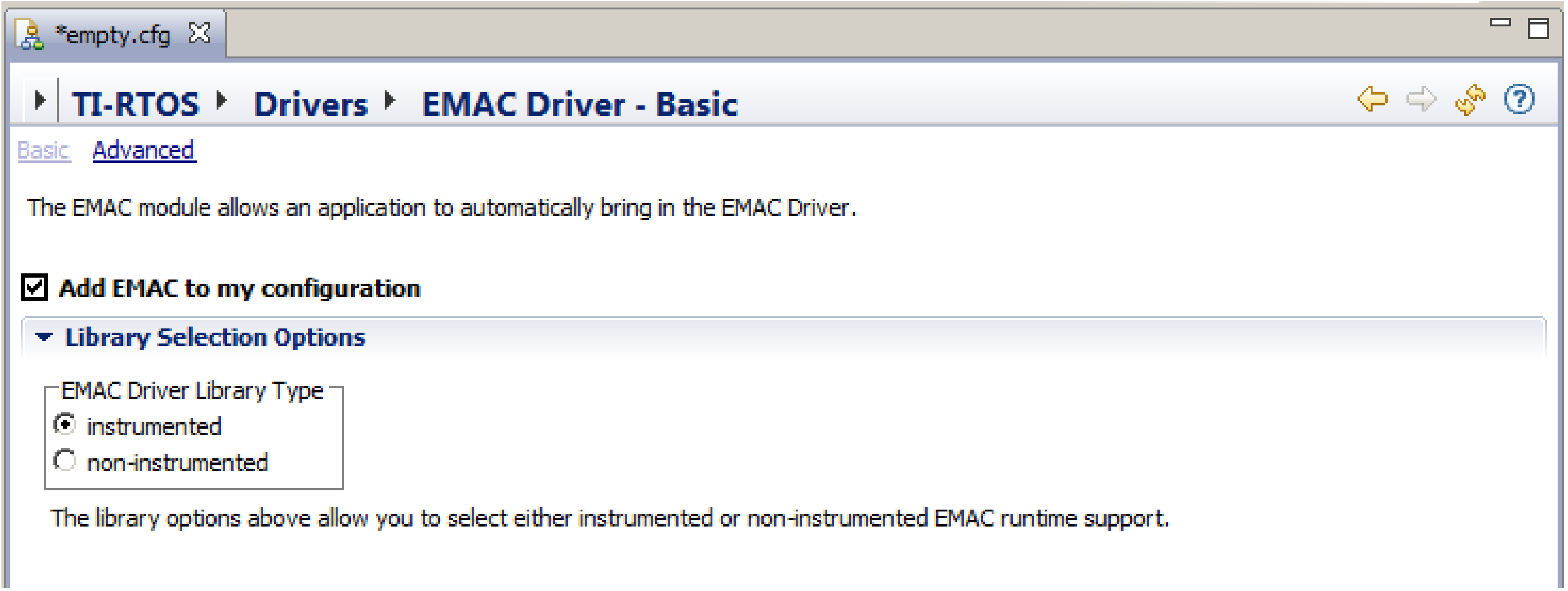
This will add the EMAC driver and allows you to configure it.

Note: TI-RTOS and GPIO are already used. This is denoted by the green check.



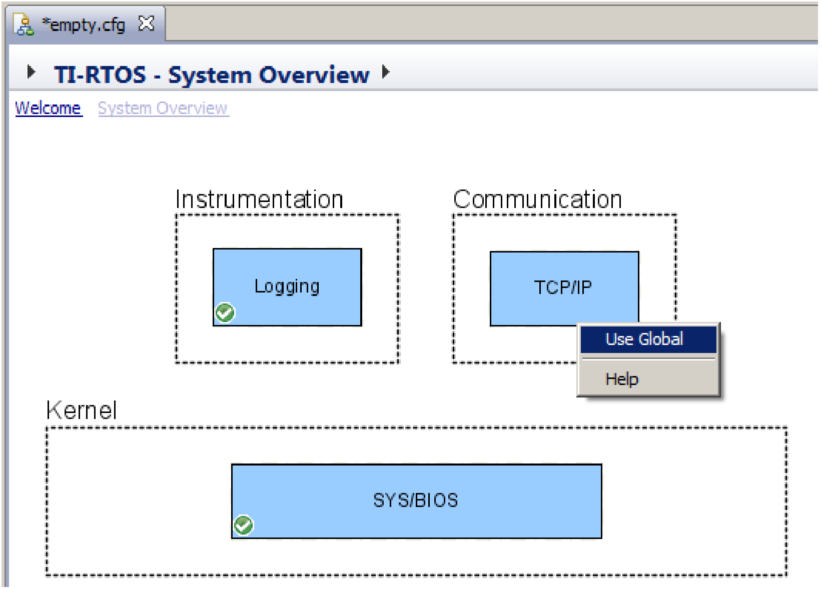
Adding Networking Stack

The instrumented library includes assert checking and Log events to help in debugging.

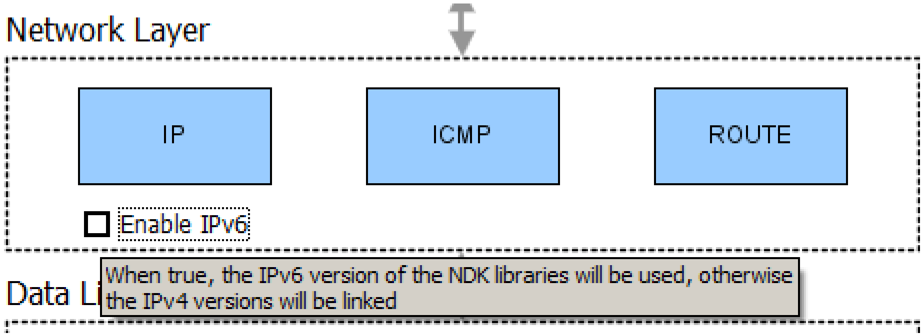


Adding Networking Stack

- Go back to the TI-RTOS System Overview and do a “Use” on the “TCP/IP” box.

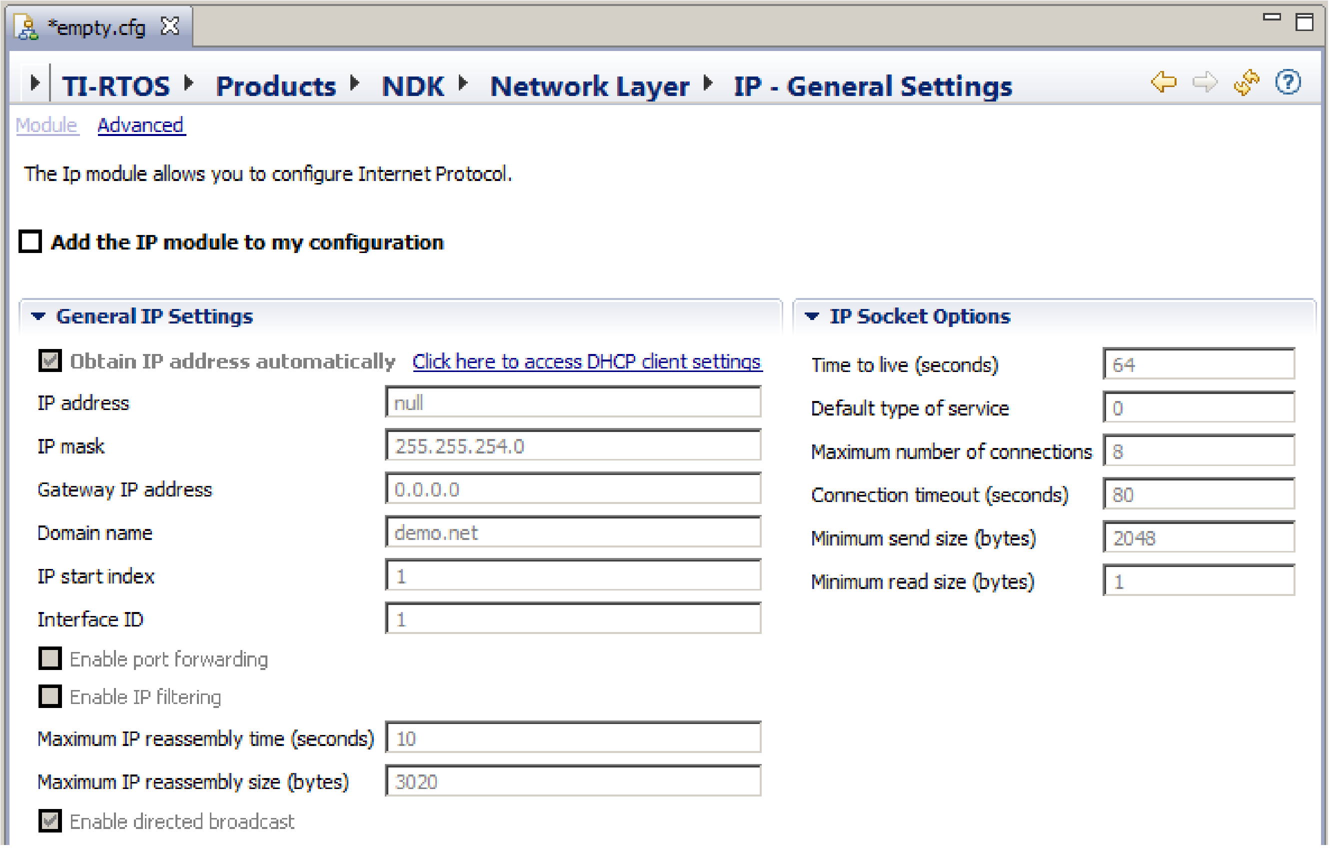


- In “System Overview”, you can now deselect IPv6 (below the IP box) because it is not used in this example.



Adding Networking Stack

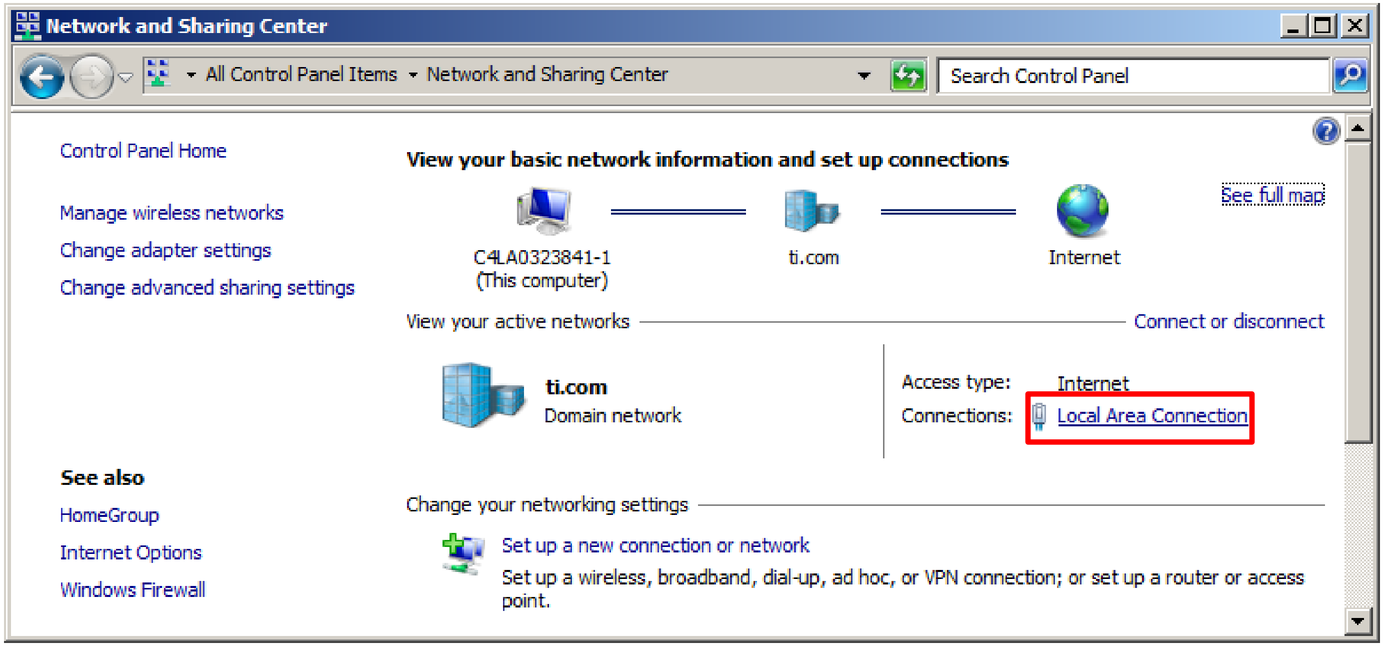
- If you select the IP box, it goes to the IP configuration. You can specify whether to use a static IP address or DHCP. The default is DHCP which is what we will use for this lab. (The next four slides go over how to use cross-over cable and static IP addresses).



Static IP Address PC Side (optional)

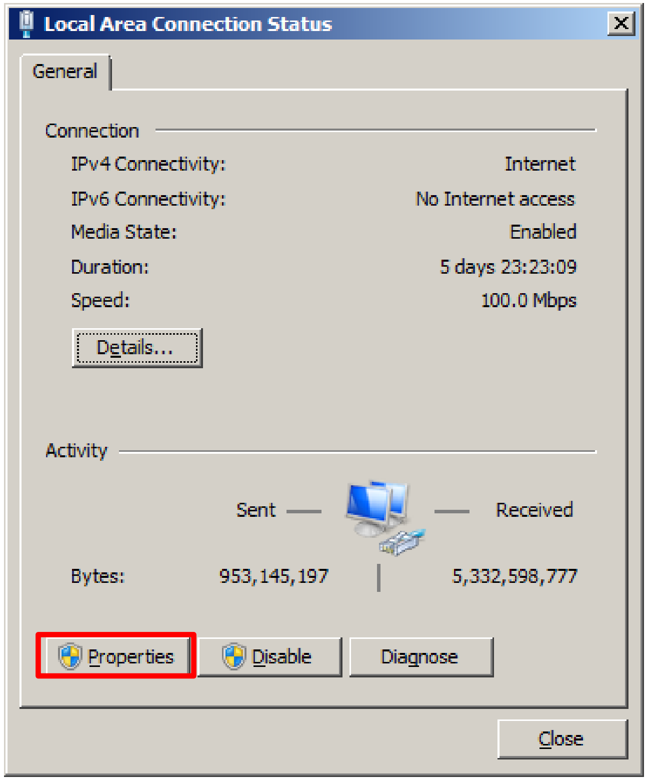
If you do not have a router, you can run the lab via static IP addresses and a cross-over Ethernet cable.

1. Turn off your PC's wireless connection (if you have one).
2. Open your PC's Network Settings (Control Panel->"Network and Sharing Center"…assuming Windows 7, the names might be slightly different if you have something else).
3. Select the "Local Area Connection" (the wired connection).

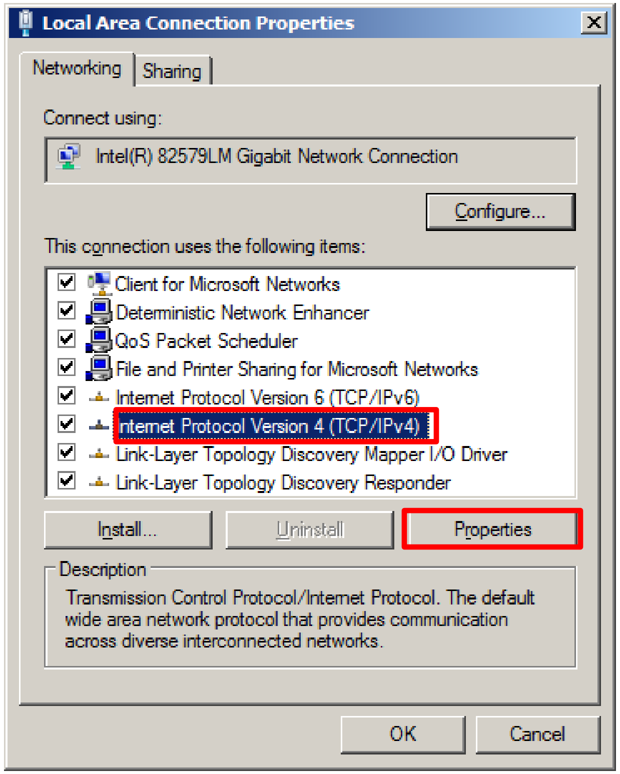


Static IP Address PC Side (optional)

4. Select Properties



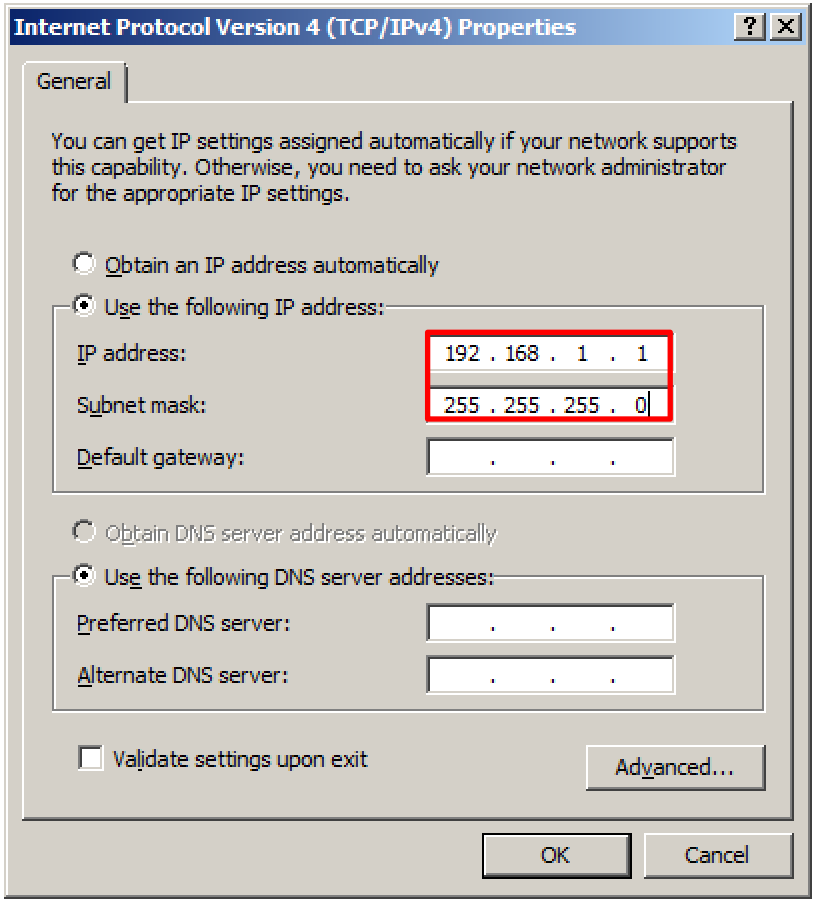
5. Select TCP/IPv4 Properties



Static IP Address PC Side (optional)

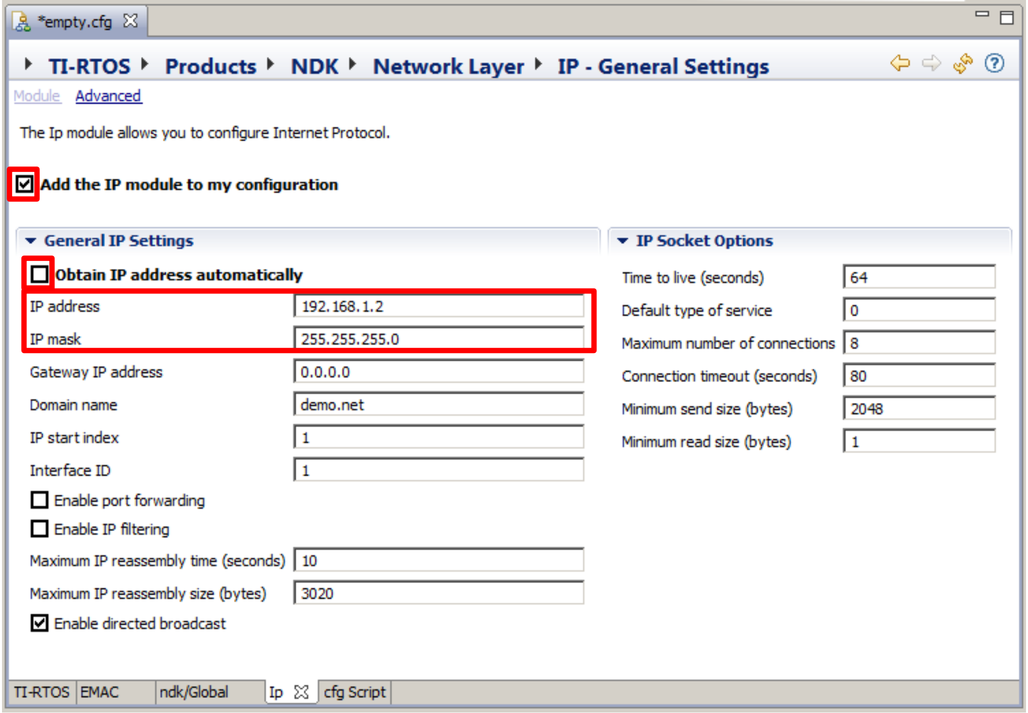
6. Select “Use the following IP address” and set the “IP address” and “Subnet mask” as shown.

   Note: once this lab is over, please remember to go back to your original PC settings!



Static IP Address Target Side (optional)

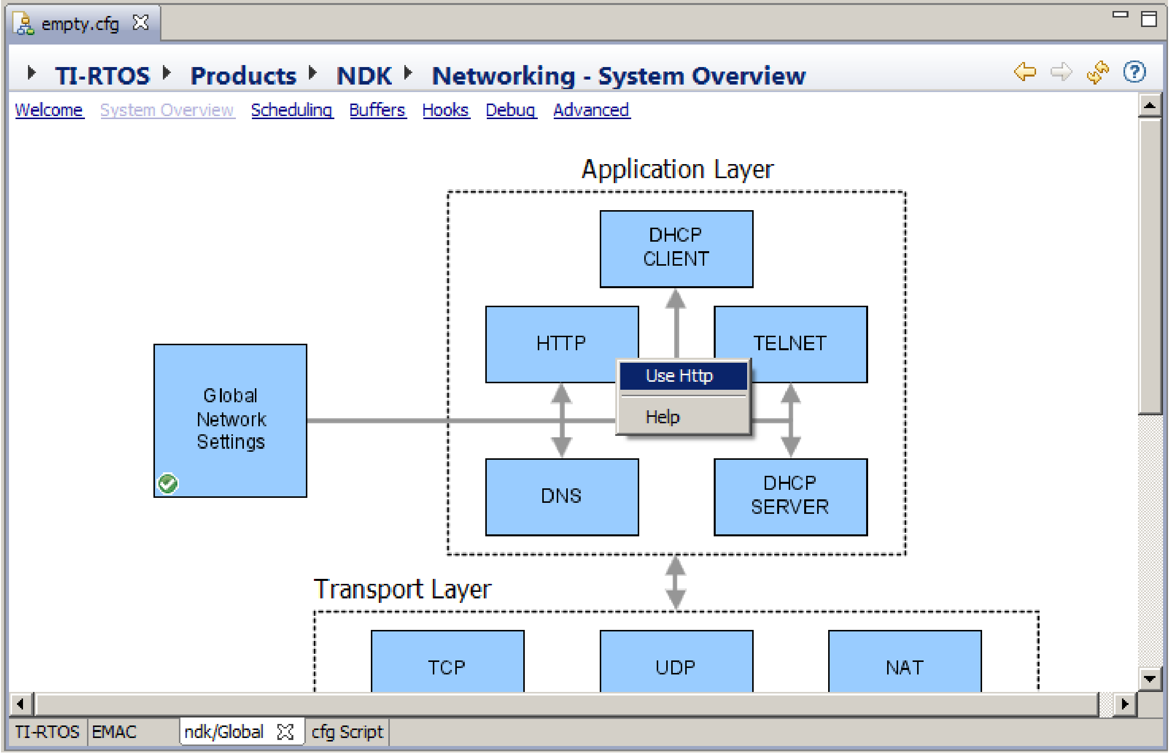
7. Open the empty.cfg file and navigate to the IP General Settings. Un-select the “Obtain IP address automatically” and fill in the IP address and IP mask as shown.



8. Connect the cross-over cable to your PC and target.

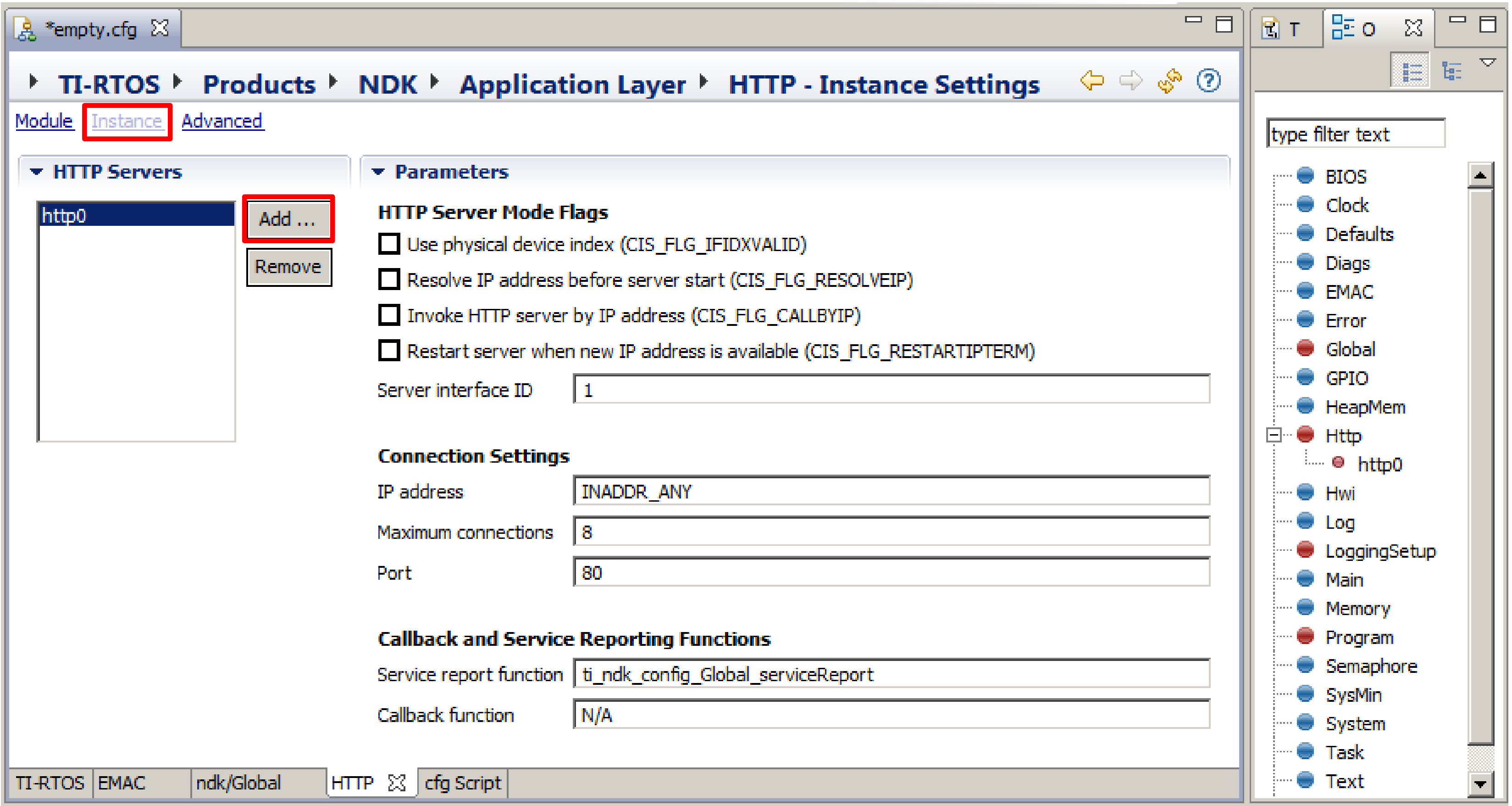
Adding Networking Stack

- Navigate to include HTTP Server by left clicking HTTP box from Networking System Overview.
- Right Click on HTTP and select “Use HTTP”. You should see the following now.



Adding Networking Stack

- Select “Instance” and hit “Add”. We’ll leave the defaults.



Adding Networking Stack

- In main() in empty.c, remove the "//" from the Board_initEMAC() line. This initializes the EMAC driver

```
Board_initGeneral();
Board_initEMAC();
Board_initGPIO();
```

- In TMDXDOCK28M36.c, you need to change the MAC address to match the sticker on **your** board.

```
UInt8 macAddress[6] = {0xff, 0xff, 0xff, 0xff, 0xff, 0xff};
```

Do not use the MAC Address shown to the right! This picture is just to help you locate the MAC address on your TMDXDOCK28M36 board.

Adding Networking Stack

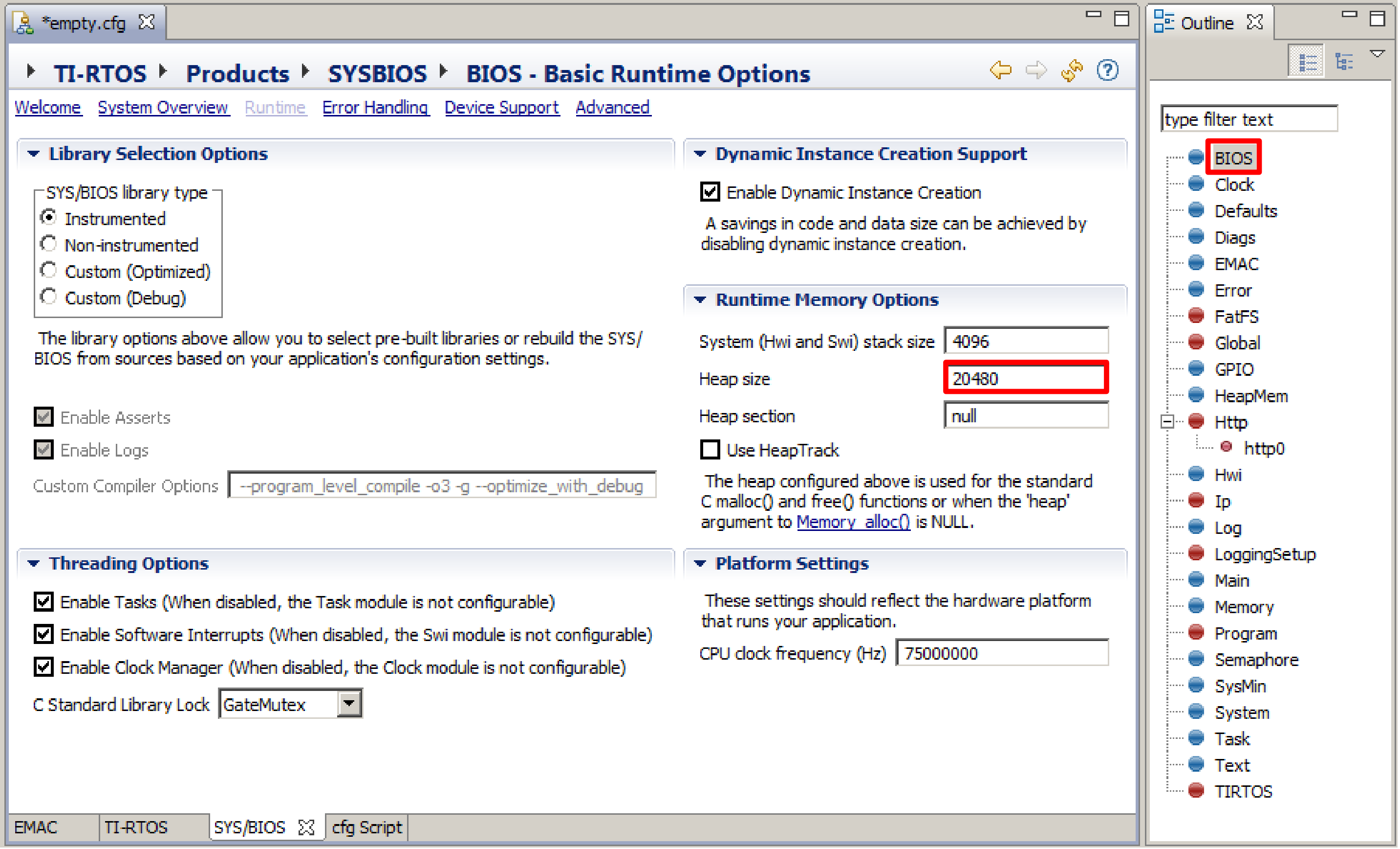
- Save and build application.
- Load and halt at main().
  - Look at ROV's Task, EMAC and Ndk windows
- Now run and look at the console window.

```
Starting the example
System provider is set to SysMin. Halt the target and use ROV to view output.
ti.sysbios.heaps.HeapMem: line 307: out of memory: handle=0x2000912c, size=1804
00000.000 mmBulkAlloc(): could not allocate memory: out of memory: handle=0x%x, size=%u

Service Status: DHCPC    : Failed   :          : 000
ti.sysbios.heaps.HeapMem: line 307: out of memory: handle=0x2000912c, size=2048
xdc.runtime.Error.raise: terminating execution
```

- What happened?
- Look at HeapMem in ROV. There is not enough memory. The empty example does not have a big enough heap…let's fix this.

Fitting Networking Stack into Target

First, the default heap is too small. Select the BIOS page in the outline window and navigate to the "Runtime" page. Change the heap to 20480 (0x5000).



NDK Size

Why increase the default heap to 20480 (0x5000)?

The HTTP Server runs in it's own Task. So it needs a stack (2048 bytes by default). There are going to be a socket for each connected client, so that's another 4096 bytes by default (2048 bytes each both a Rx and Tx buffer).

For more details about the Networking Stack's memory usage and ways to customize it, please refer to http://processors.wiki.ti.com/index.php/TI-RTOS_Networking_Stack_Memory_Usage

Running HTTP Server

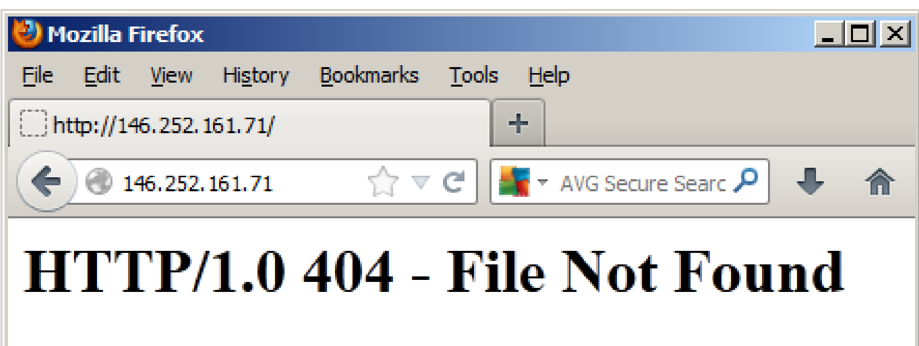
- Make sure you have the Ethernet cables connected!

- Build and load the application. Once you run it, you should see the following:

```
[Cortex_M3_0] Service Status: DHCPC    : Enabled  :          : 000
[Cortex_M3_0] Service Status: HTTP     : Enabled  :          : 000
[Cortex_M3_0] Service Status: DHCPC    : Enabled  : Running  : 000
[Cortex_M3_0] Network Added: If-1:146.252.161.71
[Cortex_M3_0] Service Status: DHCPC    : Enabled  : Running  : 017
```

- Look at ROV's Task, Ndk, and EMAC

- Open your browser of choice on your PC to the IP Address…



What's wrong??

Let’s Make a WebServer!

Steps

- Create and build new “empty” project
- Loading and debugging the empty project
- Add in Networking stack
- **Add HTTP pages**

Adding HTTP Server Content

The following slides are going to summarize the
<tirtos>/products/<ndk>/docs/spru524h.pdf's
"Appendix E: Web Programming with the HTTP Server" section…

To help lab the lab run smoothly, all the necessary HTML, converted header and source files can be found on the following wiki:
http://processors.wiki.ti.com/index.php/TI-RTOS_HTTP_Example

Adding HTTP Server Content

Let’s add a simple html page for the target. Steps 1 and 2 can be skipped if you use the supplied default.h. These two steps are used to create this file.

1. Create a simple default.html file that contains “Hello World” in the same directory as your project.

2. Convert this to a Char array that can be included into the application. Run the binsrc.exe that is in the <tirtos>\products\<ndk>\packages\ti\ndk\tools\binsrc directory.

   ```
   >binsrc.exe default.html default.h DEFAULT
   ```

   This creates a default.h file, which has the Char array representation of the webpage.

default.h

```
#define DEFAULT_SIZE 254
static const unsigned char DEFAULT[] = {
    0x3C, 0x21, 0x44, 0x4F, 0x43, 0x54, 0x59, 0x50, 0x45, 0x20, 0x68, 0x74,
    0x6D, 0x6C, 0x20, 0x50, 0x55, 0x42, 0x4C, 0x49, 0x43, 0x20, 0x22, 0x2D,
```

Adding HTTP Server Content

3. Add default.h into your project.

4. Add the following functions into empty.c. The NDK's HTTP has an embedded file system (efs).

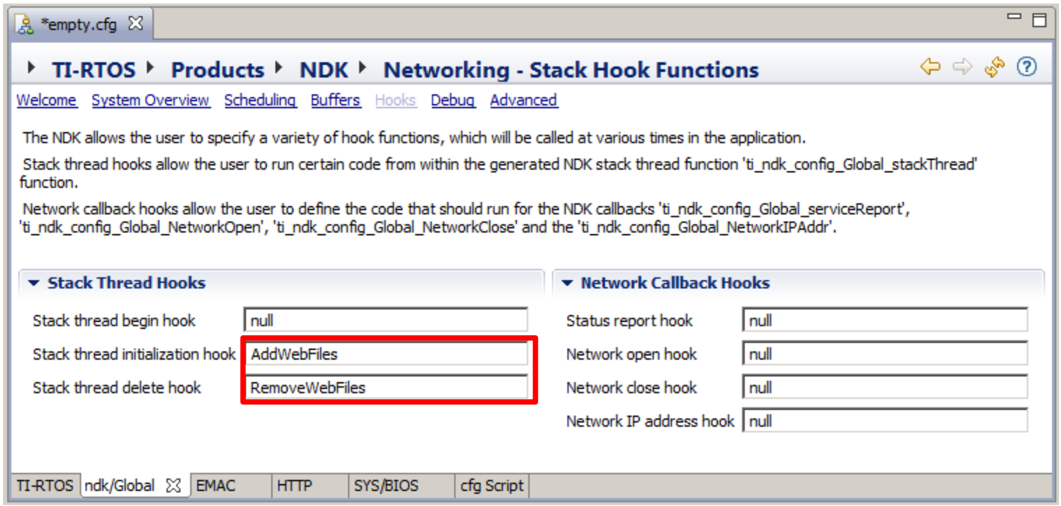
```
Void AddWebFiles(Void)
{
    //Note: both DEFAULT_SIZE and DEFAULT are defined in default.h
    efs_createfile("index.html", DEFAULT_SIZE, (UINT8 *)DEFAULT);
}

Void RemoveWebFiles(Void)
{
    efs_destroyfile("index.html");
}
```

5. Add the following lines at the top of the empty.c file. This is needed to include the webpage and resolve the efs APIs.

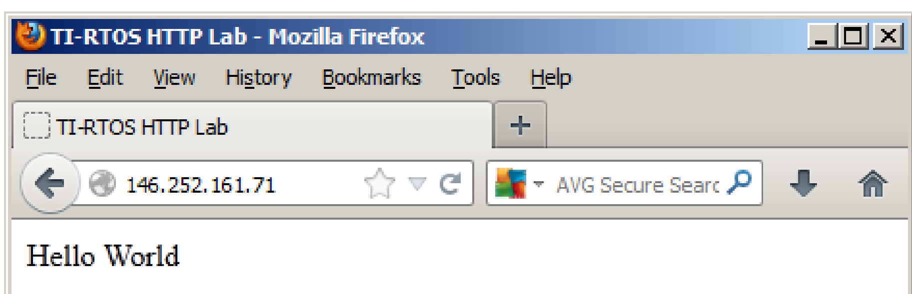
```
#include <ti/ndk/inc/netmain.h>

#include "default.h"
```

Adding HTTP Server Content

6. Now to hook the **AddWebFiles** and **RemoveWebFiles** functions into the stack. These functions will be called when the stack is started up and shutdown respectively. In the ndk/Global's Hooks page, add the two functions as shown:



5. Build, load, run and now point your browser at the IP address.



Adding HTTP Server Content: CGI

Pages are great, but how do you add “smarts” to the page? For example, how long has the target been up?

The NDK supports CGI scripts to be on the target. The basic idea is that you write a CGI function in ‘C’ that plugs into the EFS.

Let’s add the time that have occurred since the system has started using the Clock_getTicks() API.

Note: steps 1 and 2 can be skipped if you use the supplied default_withCGI.h. Add the file to your project (and remove the default.h). Then rename the #include line from “default.h” to “default_withCGI.h” in empty.c

1. Add a getTime.cgi command onto the default.html page.

```
<body>
Hello World<br>
<br>
Get current time  <a href="getTime.cgi">getTime.cgi</a><br>
```

2. Convert the default.htm via binsrc

```
>binsrc.exe default.html default.h DEFAULT
```

Adding HTTP Server Content: CGI

Now add the following code into empty.c (above the `AddWebFiles()` function)

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

Int getTime(SOCKET s, int length)
{
    Char buf[200];
    static UInt scalar = 0;

    if (scalar == 0) {
        scalar = 1000000u / Clock_tickPeriod;
    }
    httpSendStatusLine(s, HTTP_OK, CONTENT_TYPE_HTML);
    httpSendClientStr(s, CRLF);
    httpSendClientStr(s,
        "<html><head><title>SYS/BIOS Clock "\
         "Time</title></head><body><h1>Time</h1>\n");
    System_sprintf(buf, "<p>Up for %d seconds</p>\n",
        ((unsigned long)Clock_getTicks() / scalar));
    httpSendClientStr(s, buf);
    httpSendClientStr(s, "</table></body></html>");
    return (1);
}
```

Adding HTTP Server Content: CGI

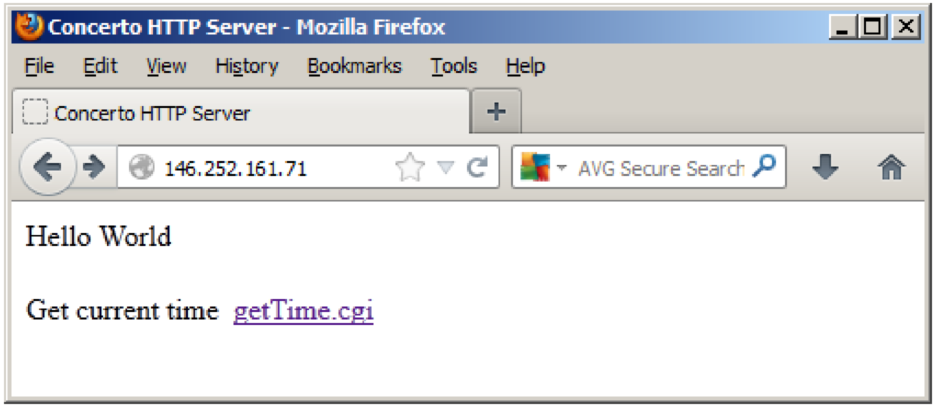
Now hook the cgi into the EFS in empty.c (new lines in bold)

```
Void AddWebFiles(Void)
{
    efs_createfile("index.html", DEFAULT_SIZE, (UINT8 *)DEFAULT);
    efs_createfile("getTime.cgi", 0, (UINT8 *)&getTime);
}

Void RemoveWebFiles(Void)
{
    efs_destroyfile("index.html");
    efs_destroyfile("getTime.cgi");
}
```

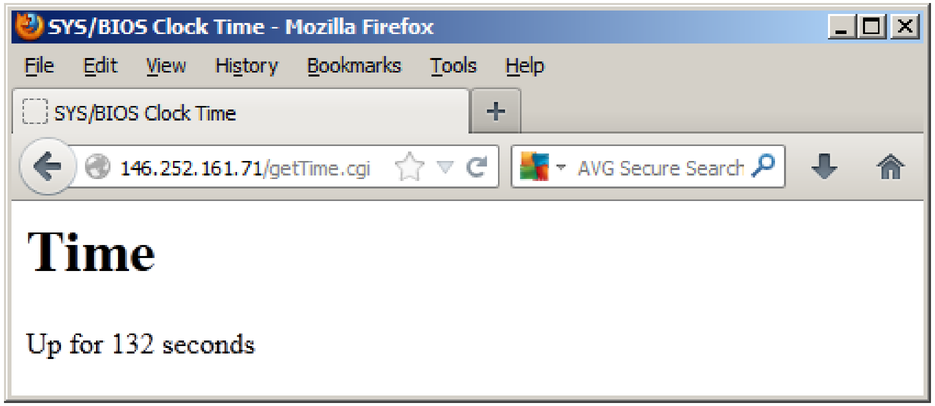
Build, load and run.

Adding HTTP Server Content: CGI

Point browser to the IP address.



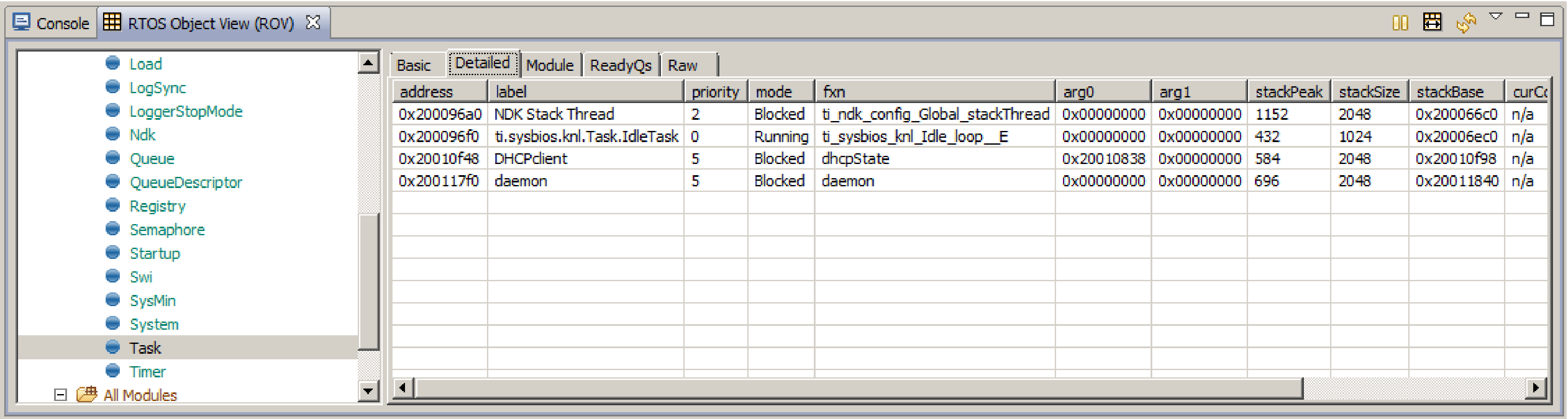
Click the getTick.cgi link and see how long the target has been up!



Refresh and time will update.

ROV Debug Information

Suspend the target and look at Task's Detailed view in ROV now.



address	label	priority	mode	fxn	arg0	arg1	stackPeak	stackSize	stackBase	curC
0x200096a0	NDK Stack Thread	2	Blocked	ti_ndk_config_Global_stackThread	0x00000000	0x00000000	1152	2048	0x200066c0	n/a
0x200096f0	ti.sysbios.knl.Task.IdleTask	0	Running	ti_sysbios_knl_Idle_loop__E	0x00000000	0x00000000	432	1024	0x20006ec0	n/a
0x20010f48	DHCPclient	5	Blocked	dhcpState	0x20010838	0x00000000	584	2048	0x20010f98	n/a
0x200117f0	daemon	5	Blocked	daemon	0x00000000	0x00000000	696	2048	0x20011840	n/a

The NDK Stack Thread is the main networking Task

The DHCPClient Task is responsible for renewing the IP Address.

The daemon is the HTTP server.

Looks like we probably can reduce the stacks a little and save some memory.

System Analyzer Debug Information

Let's look at System Analyzer…

Open System Analyzer (Tools->System Analyzer->Live) and select Start

Console | RTOS Object View (ROV) | *Live Session | Analyze

	Type	Time	Error	Master	Message	Event	EventClass	Data1
66		10207268633806		Cortex_M3_0	LM_setPri: tsk: 0x200096a0, func: 0x208e1d, oldPri: 9, newPri 2	Task_LM_setPri	Unknown	ti_ndk_config_Global_stackThread()
67		10207268642260		Cortex_M3_0	LM_pend: sem: 0x2000ff20, count: 0, timeout: -1	Semaphore_LM_pend	Unknown	
68		10207268649326		Cortex_M3_0	LD_block: tsk: 0x200096a0, func: 0x208e1d	Task_LD_block	Unknown	ti_ndk_config_Global_stackThread()
69		10207268660113		Cortex_M3_0	LM_switch: oldtsk: 0x200096a0, oldfunc: 0x208e1d, newtsk: 0x200096f0, newfunc: 0x2161b7	CtxChg	TSK	ti_sysbios_knl_Idle_loop__E()
70		10207349134206		Cortex_M3_0	LM_post: sem: 0x2000ff20, count: 0	Semaphore_LM_post	Unknown	
71		10207349141006		Cortex_M3_0	LD_ready: tsk: 0x200096a0, func: 0x208e1d, pri: 2	Task_LD_ready	Unknown	ti_ndk_config_Global_stackThread()
72		10207349153620		Cortex_M3_0	LM_switch: oldtsk: 0x200096f0, oldfunc: 0x2161b7, newtsk: 0x200096a0, newfunc: 0x208e1d	CtxChg	TSK	ti_ndk_config_Global_stackThread()
73		10207349164953		Cortex_M3_0	LM_setPri: tsk: 0x200096a0, func: 0x208e1d, oldPri: 2, newPri 9	Task_LM_setPri	Unknown	ti_ndk_config_Global_stackThread()
74		10207349177380		Cortex_M3_0	LM_setPri: tsk: 0x200096a0, func: 0x208e1d, oldPri: 9, newPri 2	Task_LM_setPri	Unknown	ti_ndk_config_Global_stackThread()
75		10207349185833		Cortex_M3_0	LM_pend: sem: 0x2000ff20, count: 0, timeout: -1	Semaphore_LM_pend	Unknown	
76		10207349192900		Cortex_M3_0	LD_block: tsk: 0x200096a0, func: 0x208e1d	Task_LD_block	Unknown	ti_ndk_config_Global_stackThread()
77		10207349203686		Cortex_M3_0	LM_switch: oldtsk: 0x200096a0, oldfunc: 0x208e1d, newtsk: 0x200096f0, newfunc: 0x2161b7	CtxChg	TSK	ti_sysbios_knl_Idle_loop__E()

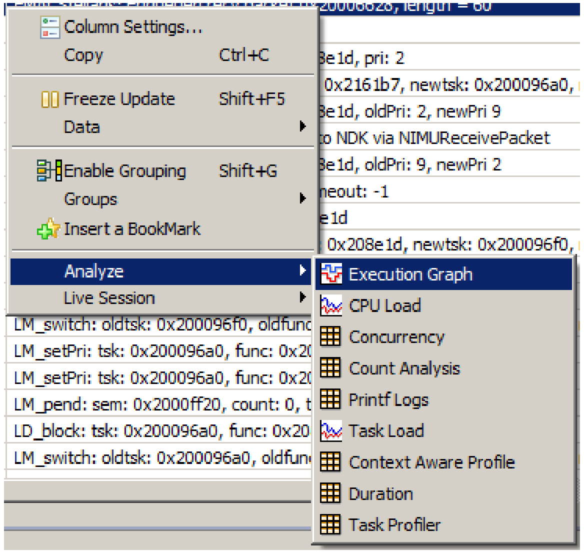
Showing 77 records

UIA's LoggingSetup is setting up basic logging in the example's .cfg file.

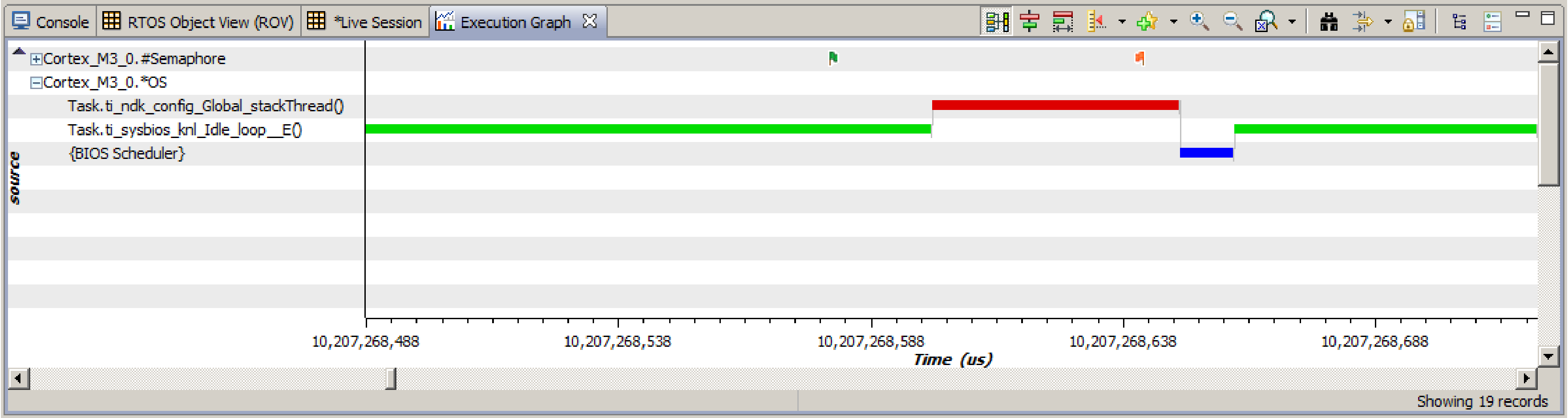
Note: This lab is showing stopmode System Analyzer. TI-RTOS also supports getting the Log events via UART and USB (refer to examples in the TI-RTOS product for more details).

System Analyzer Debug Information

By right clicking in the "Live Session", you can select more displays. For example, you can get the execution graph.



Then you can zoom in and see what is going on…ok…this one's a little boring!



Texas Instruments

Adding More HTTP Server Content [Extra Credit]

Create a greetings.html file that contains “Greets Everyone” and a picture (e.g. chip.jpg) (you can skip the file creation and step 1 if you use the provide files greetings.h and chip.h).

1. Run the binsrc.exe convertor on the page and the image.

```
>binsrc.exe greetings.html greetings.h GREETINGS
>binsrc.exe chip.jpg chip.h CHIP
```

2. Include the greetings.h and chip.h files into your project.

```
#include "greetings.h"
#include "chip.h"
```

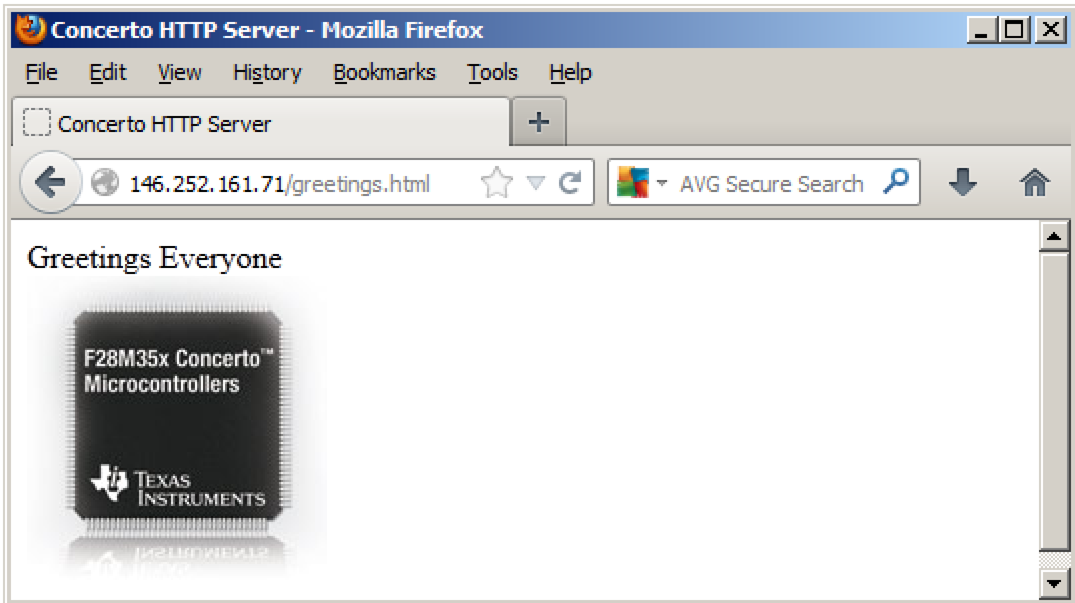
3. Add greetings and chip into the AddWebFiles/RemoveWebFiles (bolded lines)

```
Void AddWebFiles(Void)
{
    …
    efs_createfile("greetings.html", GREETINGS_SIZE, (UINT8 *)GREETINGS);
    efs_createfile("chip.jpg", CHIP_SIZE, (UINT8 *)CHIP);
}
Void RemoveWebFiles(Void)
{
    …
    efs_destroyfile("greetings.html");
    efs_destroyfile("chip.jpg");
}
```

Adding more HTTP Server Content [Extra Credit]

Rebuild, load and run the target now. Point your browser at the IP address. It should still be the “Hello World” page.

Now point to <ipaddr>/greetings.html. You should see the following.



It is also possible to pull the pages from an SD card or USB flash drive. The details are on the http://processors.wiki.ti.com/index.php/TI-RTOS_HTTP_Example page.

Lab Summary

At this stage you should have been able to:

- Make a simple webpage

Please remember to return your PC network configuration to its original settings. Also turn your wireless back on (if so desired).

Resources and Summary

Support Resources

- e2e Forum - TI-RTOS currently uses the SYS/BIOS e2e Forum:
  - External: http://e2e.ti.com/support/embedded/bios/default.aspx
- Wiki: http://processors.wiki.ti.com/index.php/Main_Page
  - Select 'TI-RTOS' category
- Download page:
  - http://software-dl.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/tirtos/index.html

Sales Resources

- www.ti.com Web Page:
  - www.ti.com/tool/ti-rtos
  - Includes link for product downloads for customers
  - Includes link for **product bulletin**
- Youtube Overview Videos:
  - **Overview:** http://www.youtube.com/watch?v=Vrs-o8HsMs8
  - **Components:** http://www.youtube.com/watch?v=nkA8ss5FAqE
  - **Tools:** http://www.youtube.com/watch?v=_F2bVVqaeFk

Summary

- TI-RTOS enables MCU software developers to focus on their specific areas of applications expertise
  - TI-RTOS provides widely required connectivity software such as TCP/IP and USB stacks
  - TI-RTOS provides an integrated set of proven embedded software components that are known to work together
- TI-RTOS provides standard APIs to device drivers to abstract applications from HW specifics
  - Applications are easily ported to the latest devices
- TI-RTOS is augmented by graphical configuration and system-level debug tools
- TI-RTOS no-cost licensing removes commercial barriers to deployment
- TI-RTOS is developed and supported by TI