18047 TCP2

Introduction to the MPLAB® Harmony TCP/IP Stack

Lab Manual

Bob Smith
Microchip Technology Inc.

v1.0

SOFTWARE:

You may use Microchip software exclusively with Microchip products. Further, use of Microchip software is subject to the copyright notices, disclaimers, and any license terms accompanying such software, whether set forth at the install of each program or posted in a header or text file.

Notwithstanding the above, certain components of software offered by Microchip and 3rd parties may be covered by “open source” software licenses – which include licenses that require that the distributor make the software available in source code format. To the extent required by such open source software licenses, the terms of such license will govern.

NOTICE & DISCLAIMER:

These materials and accompanying information (including, for example, any software, and references to 3rd party companies and 3rd party websites) are for informational purposes only and provided “AS IS.” Microchip assumes no responsibility for statements made by 3rd party companies, or materials or information that such 3rd parties may provide.

MICROCHIP DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, INCLUDING ANY IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY DIRECT OR INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND RELATED TO THESE MATERIALS OR ACCOMPANYING INFORMATION PROVIDED TO YOU BY MICROCHIP OR OTHER THIRD PARTIES, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBLITY OF SUCH DAMAGES OR THE DAMAGES ARE FORESEEABLE.

TRADEMARKS:

The Microchip name and logo, the Microchip logo, dsPIC, FlashFlex, flexPWR, JukeBlox, KeeLoq, KeeLoq logo, Kleer, LANCheck, MediaLB, MOST, MOST logo, MPLAB, OptoLyzer, PIC, PICSTART, PIC32 logo, RightTouch, SpyNIC, SST, SST Logo, SuperFlash and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

The Embedded Control Solutions Company is a registered trademark of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, BodyCom, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, ECAN, In-Circuit Serial Programming, ICSP, Inter-Chip Connectivity, KleerNet, KleerNet logo, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, mTouch, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, RightTouch logo, REAL ICE, SQI, Serial Quad I/O, Total Endurance, TSHARC, USBCheck, VariSense, ViewSpan, WiperLock, Wireless DNA, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

GestIC is a registered trademarks of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

All other trademarks mentioned herein are property of their respective companies.

Introduction to Microchip's TCP/IP Stack

Table of Contents

Lab Exercise 1

Configure the Stack and Join the Network

Purpose

Before we run Labs 2 and 3 we need to start with a functioning network enabled device. We will program the PIC32 with one of the MPLAB® Harmony TCP/IP stack demonstration programs, and verify it has network connectivity.

Requirements

Development Environment:	MPLAB® X IDE v2.10 or later
C Compiler:	MPLAB® XC32 v1.31 or later
Hardware Tools:	PIC32MZ EC Starter Kit (Part # DM320006)
Lab Project Location:	C:\Masters\18047\Microchip\harmony\v0_80_02b \apps\tcpip\web_server_nvm_mpfs\firmware\pic32_eth_web_server.X

Objective

- Program the PIC32 with the web server demo application
- Verify the HTTP server is connected to the network by controlling and monitoring its web page from a PC

Procedure

1 Open the web server demonstration project

MPLAB® Harmony comes with a number of TCP/IP demonstration projects. The one we will use for these labs is named "web_server_nvm_mpfs". This implements an HTTP web server running in the PIC32. The demo webpages are compressed into a single file using the Microchip file system (mpfs) utility and they are stored in the PIC32's flash non-volatile memory (nvm).

Start the MPLAB® X IDE then open the "**web_server_nvm_mpfs**" project:

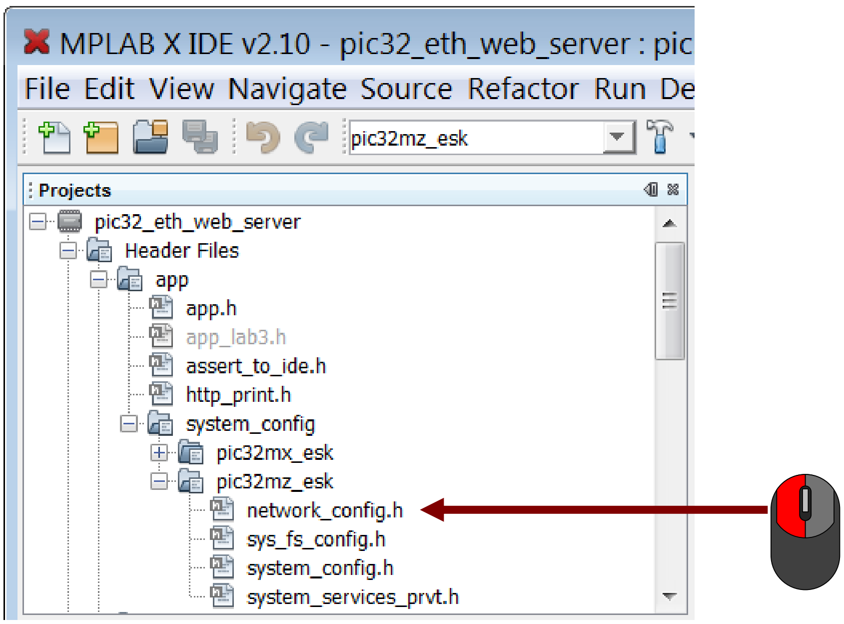
File ▸ Open Project ▸
C:\Masters\18047\Microchip\harmony\v0_80_02b\apps\tcpip
web_server_nvm_mpfs\firmware\pic32_eth_web_server.X

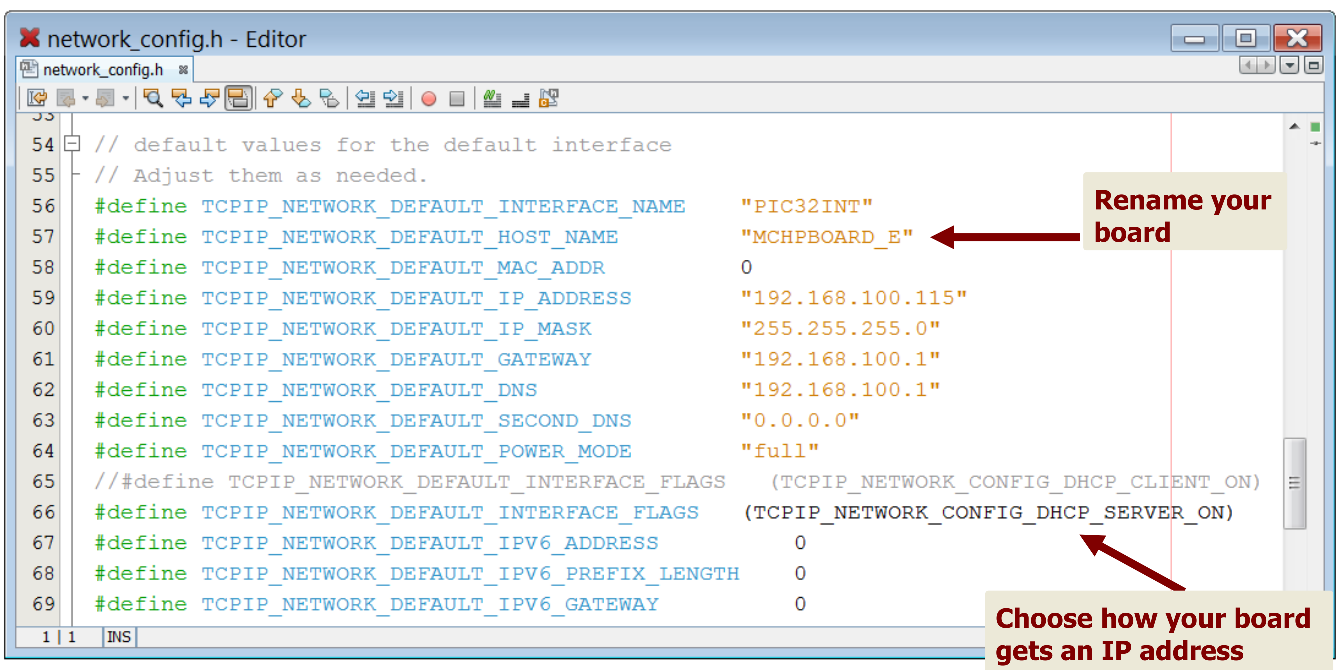
Software used in this class is based on Microchip's MPLAB® Harmony Integrated Software Framework. You will need to download and install this for use in your own projects after the class. (**www.microchip.com/harmony**).

Sorry for the crazy long directory names. These are the same names used in the Harmony TCP/IP stack. They need to be long to be descriptive.

2 Configure the network interface

Double click on “network_config.h” (in the “Header Files\app\system_config\pic32mz_esk” folder) to open the network configuration file. Feel free to provide a new hostname for your board by renaming “MCHPBOARD_E”.
We will be using the DHCP server running in the PIC32 to assign itself an IP address. Make sure “TCPIP_NETWORK_CONFIG_DHCP_SERVER _ON” is uncommented.



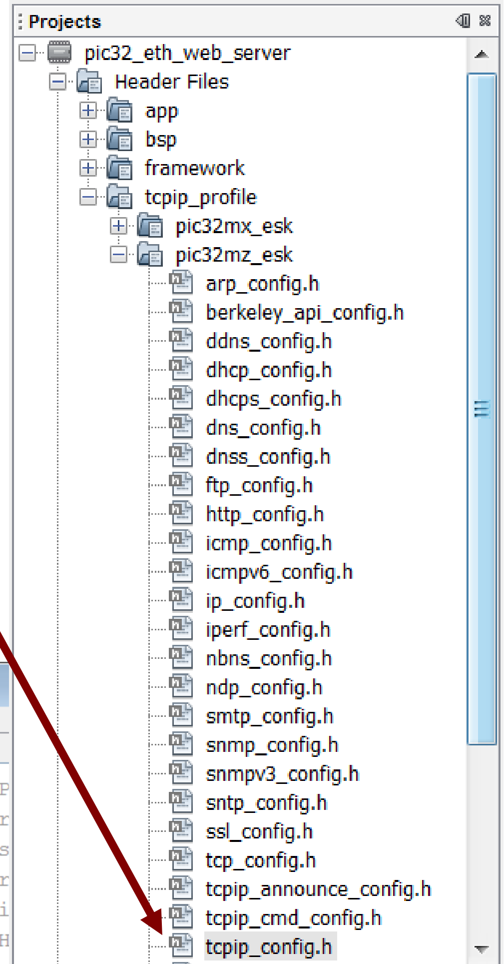


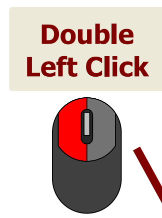
If you are connecting to the evaluation board through a router make sure to use “TCPIP_NETWORK_CONFIG_DHCP_CLIENT _ON” instead of “...DHCP_SERVER_ON”.

The IP address shown above is just the default. The DHCP server running in the PIC32 will assign itself an IP address.

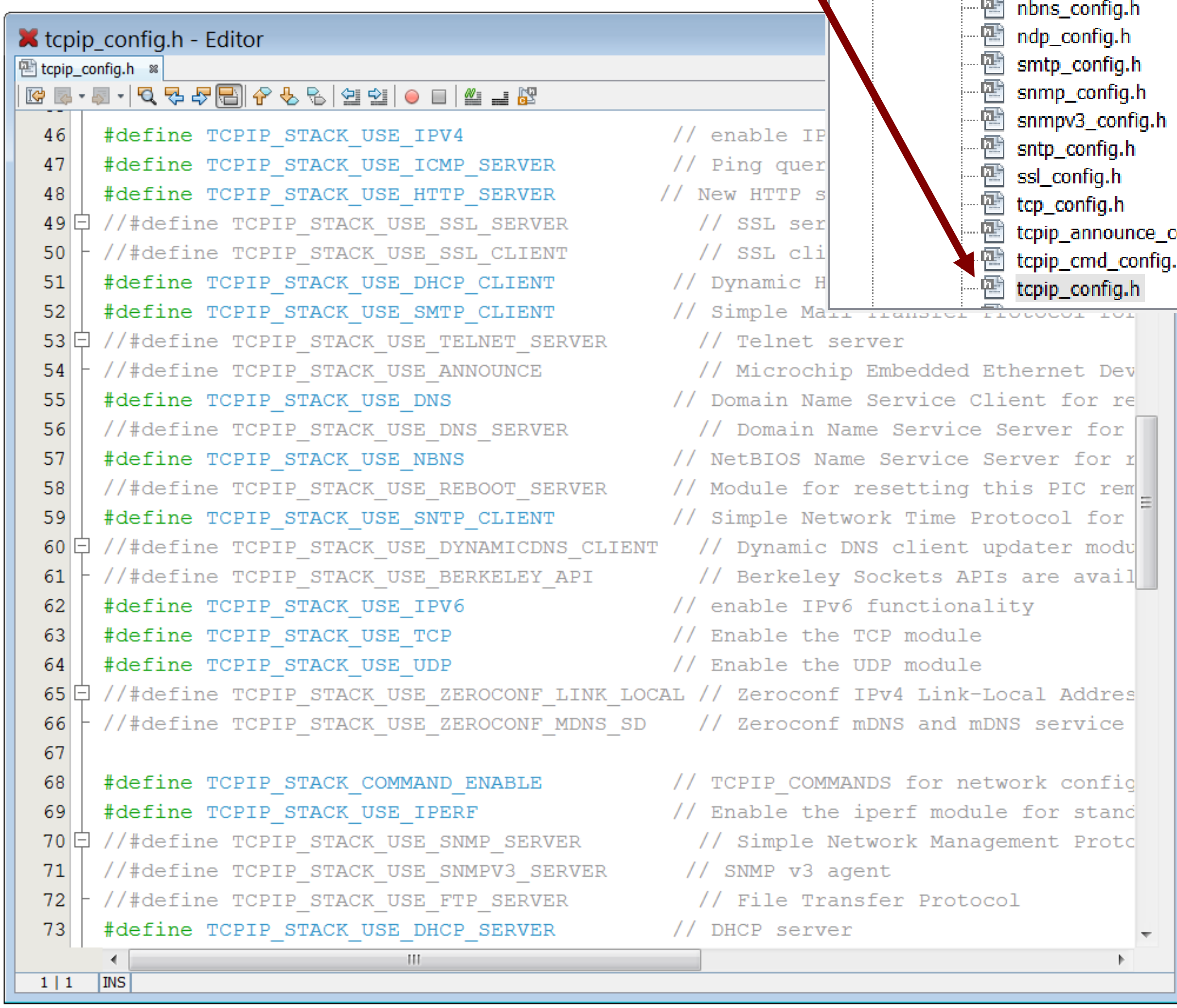
3 Enable the TCP/IP applications you need and disable those you don't.

Double click on "tcpip_config.h" (in the "Header Files\tcpip_profile\pic32mz_esk" folder) to open the TCP/IP configuration file. Observe how to enable the applications required for your project. For this lab you can leave this file unchanged.



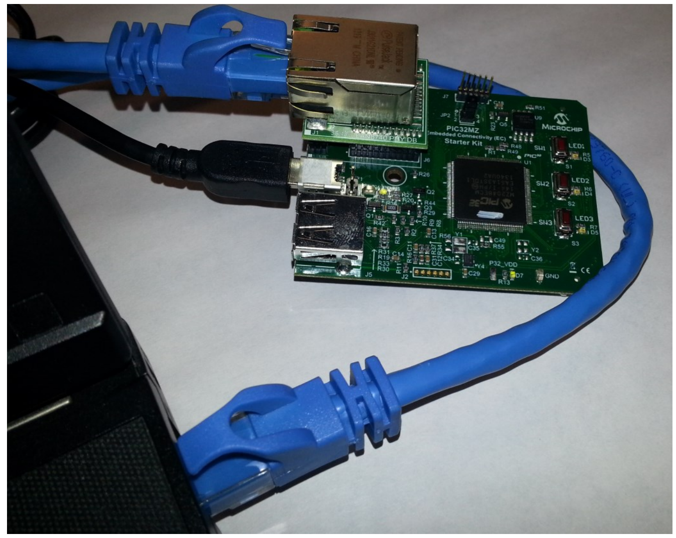


SSL and IPv6 consume a lot of memory. Make sure to disable these applications if you don't need them.



4 Setup and connect your hardware

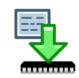
Setup your hardware as shown in the picture. The board is powered and programmed via the USB Mini-B connector. An Ethernet cable directly connects your PC with the PIC32MZ EC Starter Kit



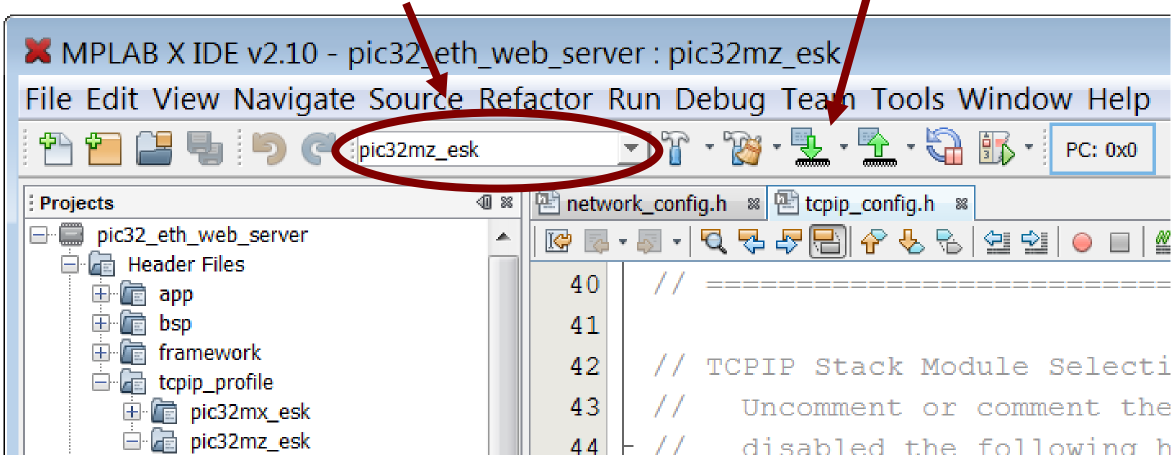
You may want to disable your WiFi interface to avoid confusing it with your Ethernet interface.

5 Build the project and program the device

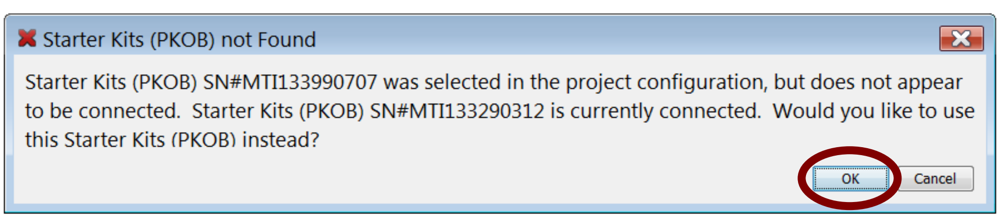
Ensure the Ethernet cable is connected between the laptop and evaluation board before programming. The network interface needs to be connected before the DHCP server starts.

Build the project, program the device, and run the code by clicking on this icon: 

Make sure the project is configured for “pic32mz_esk” using the drop-down menu.



Click “OK” when you see this message.



The TCP/IP demo projects that come with Harmony use “project configurations” to enable one project to be used for multiple hardware platforms. Note that all files in the “pic32mx_esk” folders are grayed-out indicating they are not included in this configuration of the project (we are using the pic32mz_esk evaluation board, *not* the pic32mx_esk).

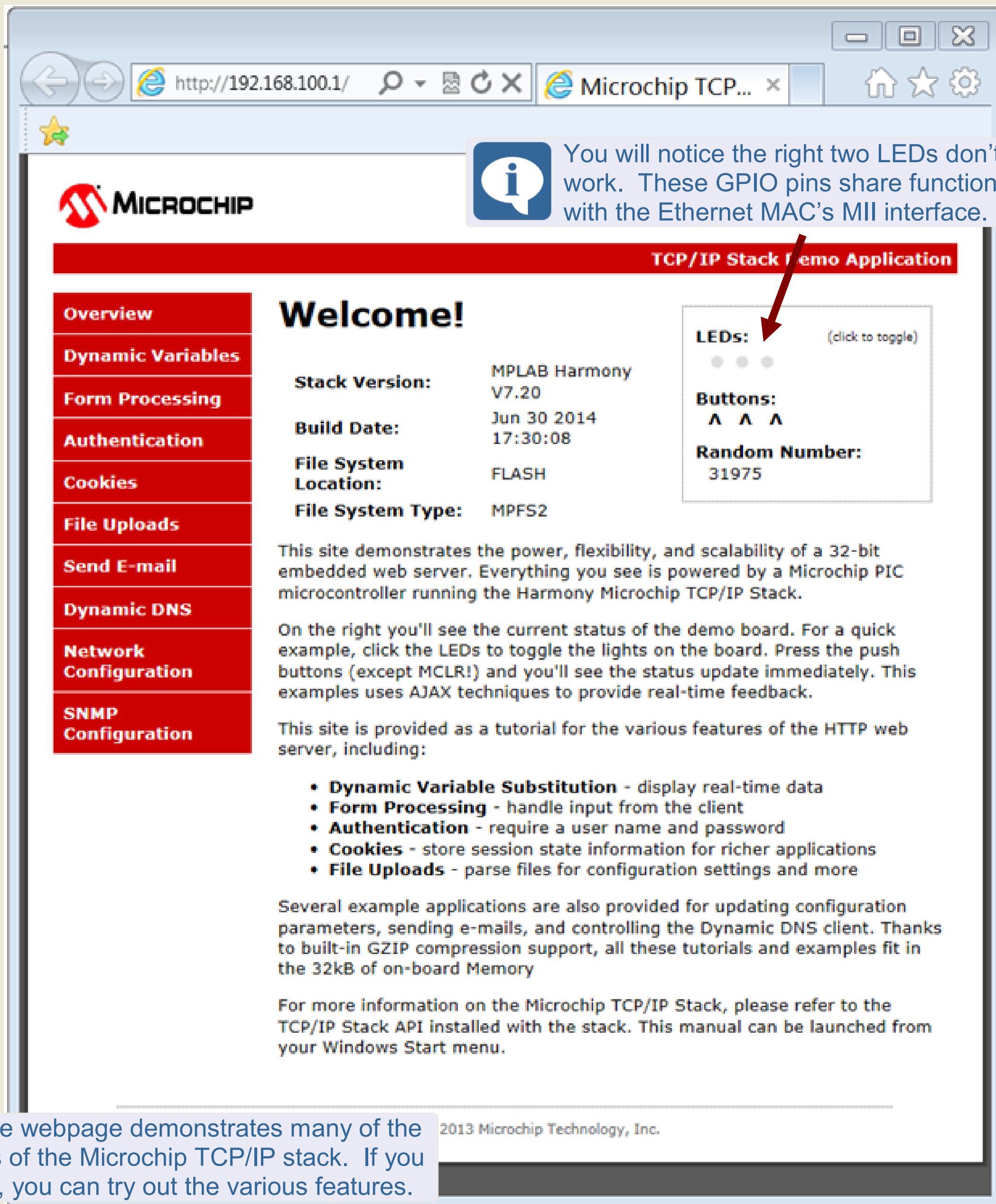
6 Open a web browser and access the web server demonstration web page

Open a web browser (i.e. Internet Explorer) and access web server demo webpage by entering the following text into the address bar:

http://192.168.100.1 or http://MCHPBOARD_E (or whatever you named your board)

> You may need to reset the board after programming to enable the network interface to start. Unplug and re-plug the USB cable to do this.

Welcome to the TCPIP Stack Demo web page! The upper right corner of this page displays the status of the PIC32MZ EC Starter Kit buttons, LEDs and random number generator in real time using dynamic variables. Click on the left LED to turn it on and off. Press one of the three buttons beside the LEDs and see their state change on the web page.



> You will notice the right two LEDs don't work. These GPIO pins share functionality with the Ethernet MAC's MII interface.

> This sample webpage demonstrates many of the capabilities of the Microchip TCP/IP stack. If you finish early, you can try out the various features.

Results

You have just set up the TCPIP Demo App project on your development board and uploaded a sample set of web pages. When complete, you should see a page similar to the screen shown on the previous page.

Conclusions

This lab has confirmed that your network setup and development board are operational.

The IP address used for the webpage has been assigned by the DHCP server running in the PIC32. If we were to connect to the evaluation board through a router, the DHCP server running in the router would have assigned the board's IP address.

Lab Exercise 2

Integrating an Application with the Stack

Purpose

For many designs, you will be integrating an existing non-internet enabled application with the stack. It is important to understand the fundamental design of the stack and where to place your application's code.

Let's assume we have an existing product that automatically dispenses pet food into a container when a button is pressed. Every button press activates a motor to dispense food for one second. This product could be connected to the internet to allow us to feed our pets while we are at work or on vacation.

In this lab, we will integrate this existing code with Microchip's TCPIP stack. Due to the complexity of the stack, it is much easier to integrate your application into the stack than the other way around. This lab will show you how to do this. When the lab is complete, the pet food dispenser code and the TCP/IP stack will be running on the same PIC32.

Requirements

Development Environment:	MPLAB® X IDE v2.10 or later
C Compiler:	MPLAB® XC32 v1.31 or later
Hardware Tools:	PIC32MZ EC Starter Kit (Part # DM320006)
Lab Project Location:	C:\Masters\18047\Microchip\harmony\v0_80_02b \apps\tcpip\web_server_nvm_mpfs\firmware\pic32_eth_web_server.X

Objective

- Experience some of the steps for combining an existing application with the TCP/IP stack
- Identify where to put your application's code (pet food dispenser) within the TCPIP stack
- Verify both the TCP/IP stack and your application are functioning

Procedure

1 Observe the main() function

There isn't much to the main () function.

The first function called in main() is **SYS_Initialize()**. As you may have guessed, this function is used to initialize the system:

- Development board (i.e. switch and LED connections to pins)
- PIC32 Device (i.e. core configuration bits, interrupts, clocks)
- Harmony middleware modules (i.e. TCP/IP stack)
- Any initialization required by your application

After that, it drops into the top-level "super" loop where it repeatedly calls **SYS_Tasks()**. This function calls the state machine for each module in the system. Some example modules include:

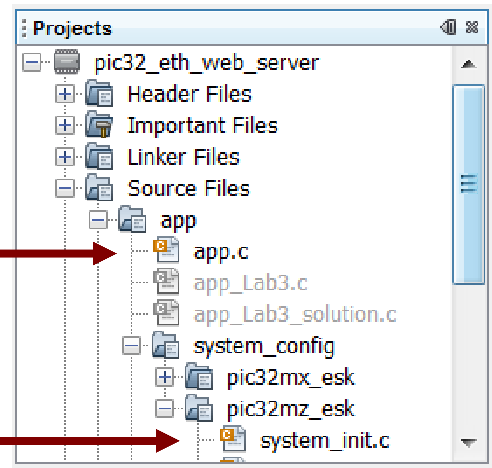
- TCP/IP Stack
- System timer
- System console (run-time system status/debug messages)
- Your application's state machine

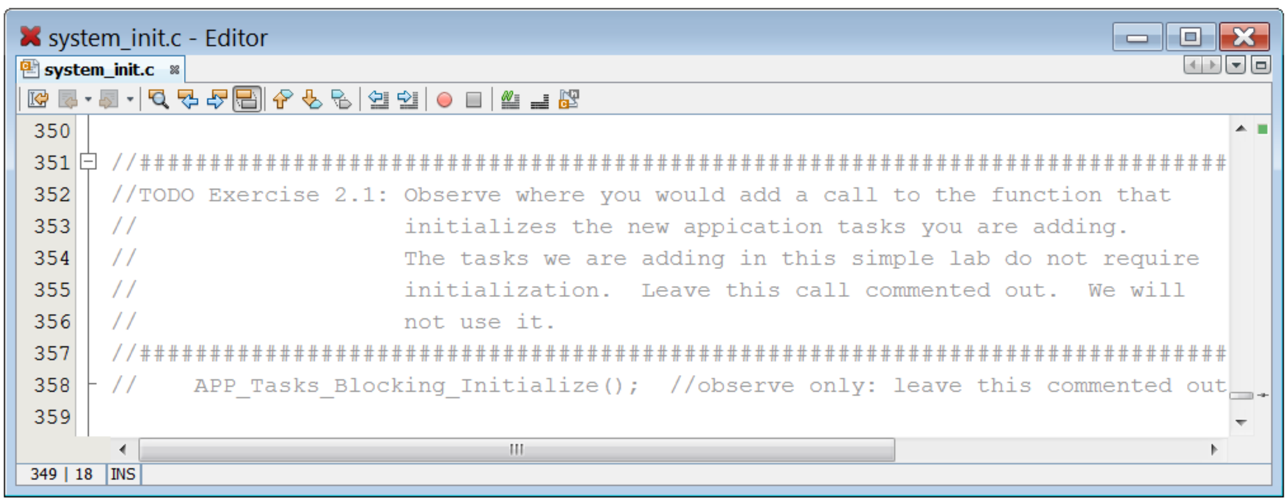
2 Add code to initialize your application

You will need to initialize your application before it starts running. Somewhere in the SYS_Initialize() function you need to call your initialization function to do this.

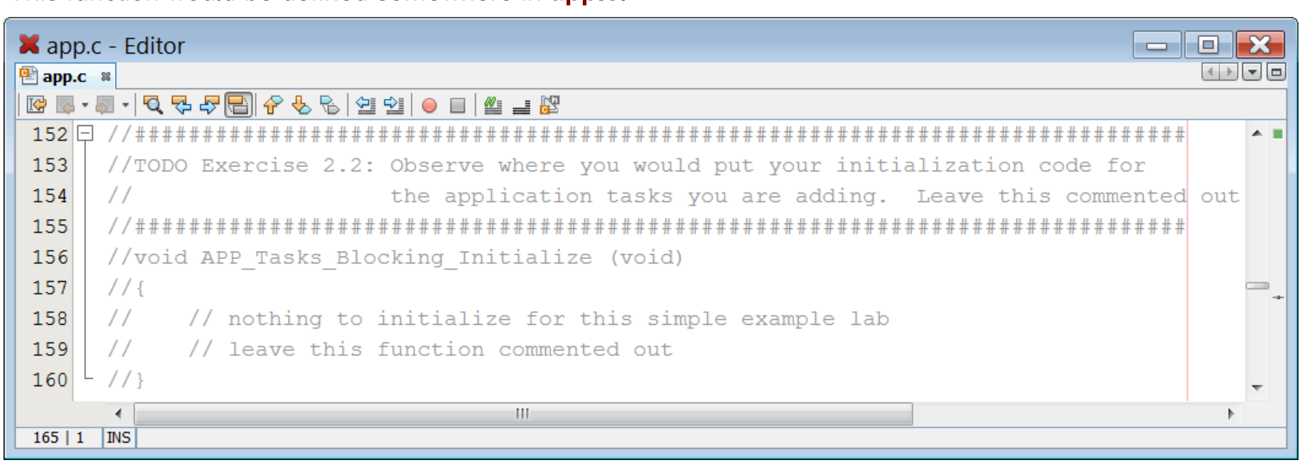
In this case, our pet food dispenser is a very simple application that doesn't require initialization. It's likely the application you plan to web enable will require initialization.

Open the **system_init.c** source file, find the SYS_Initialize() function and observe where this initialization would be called if it were needed.

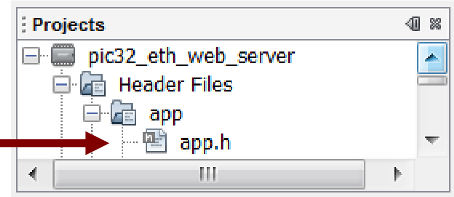




This function would be defined somewhere in **app.c**.



If you did need an initialization function for your application, don't forget to include the initialization function prototype somewhere in app.h.

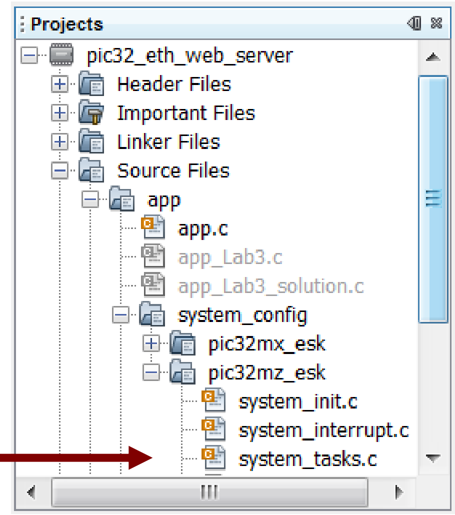


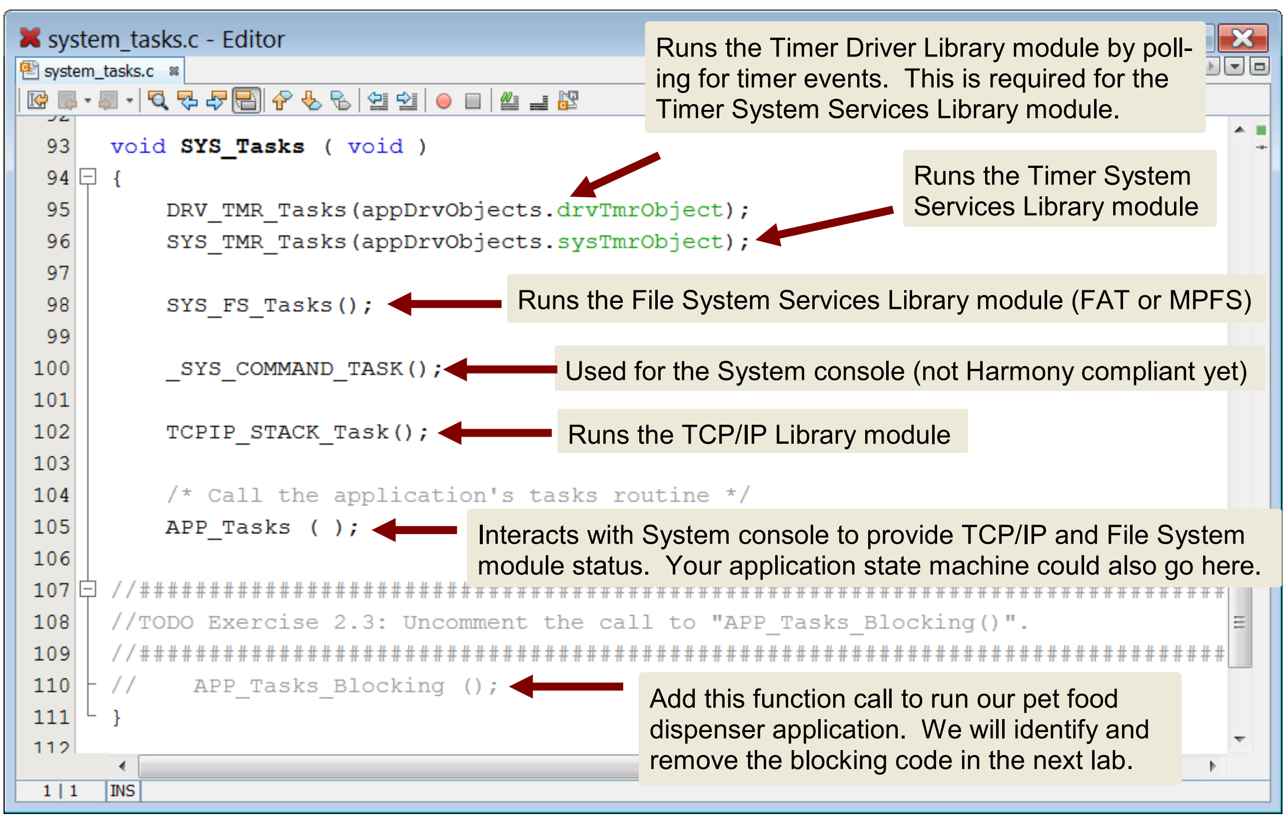
3 Add code to run your application

The next step is to add the code that will actually run our pet food dispenser application.

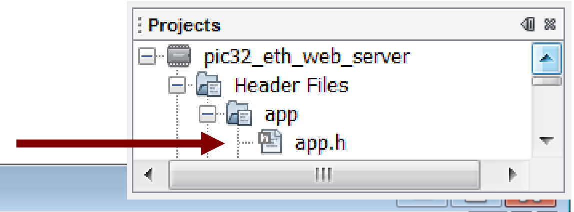
After initializing the system, the main() function drops into the top-level "super" loop where it repeatedly calls **SYS_Tasks()**. We need to add a call to our pet food dispenser tasks in this function.

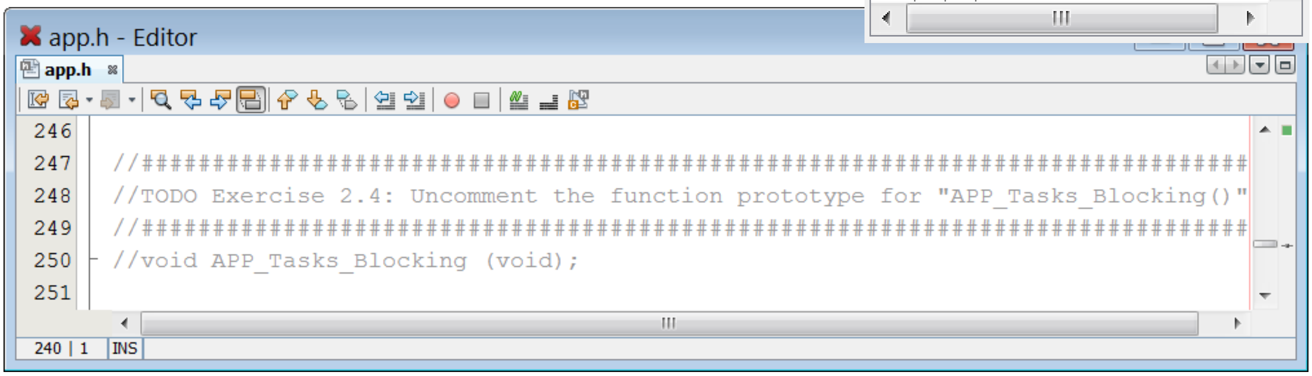
Open the **system_tasks.c** source file and uncomment the "APP_Tasks_Blocking()" function shown below.



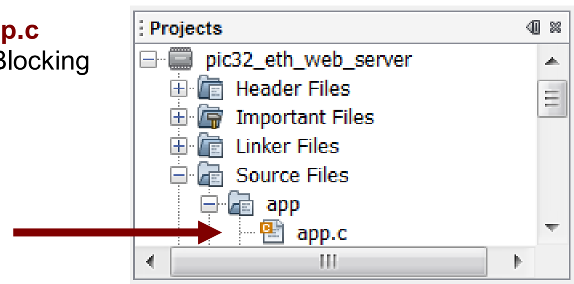


Open the **app.h** header file and uncomment the APP_Tasks_Blocking() function prototype.

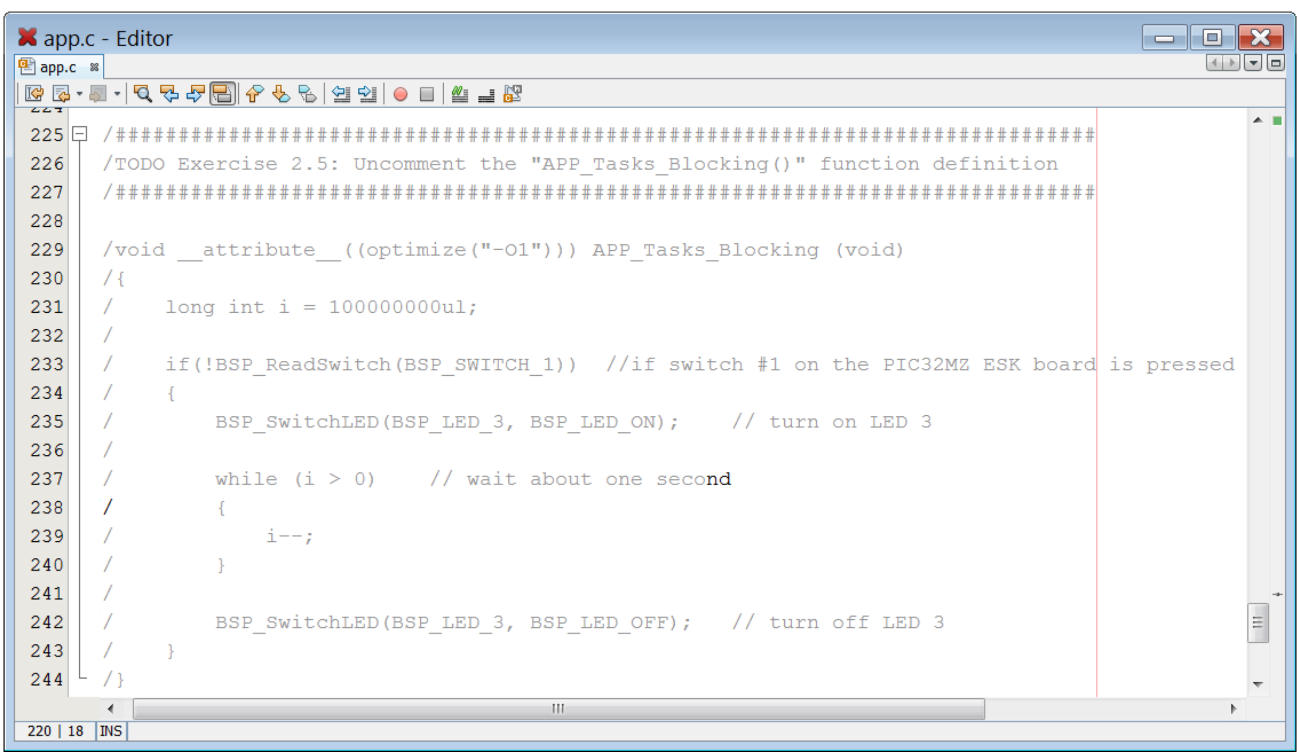




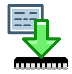
The APP_Tasks_Blocking() function will be defined in the **app.c** source file. Open this file and uncomment the APP_Tasks_Blocking () function definition .

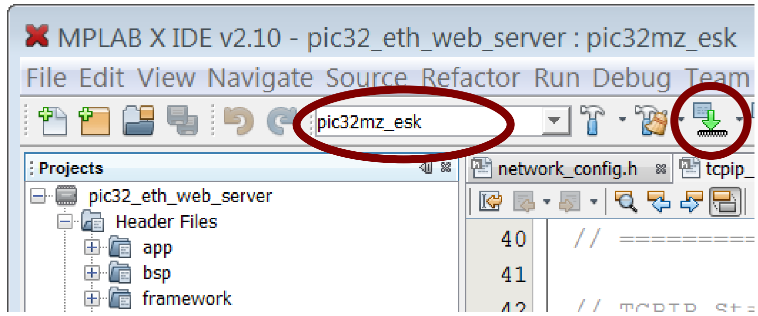


We will use one of the LEDs on the PIC32MZ EC Starter Kit board to simulate a motor turning on for one second. This motor will be used to dispense the pet food into a container. When the top switch on the board (SW1) is pressed, the motor (LED3) will turn on.



4 Build the project and program the device

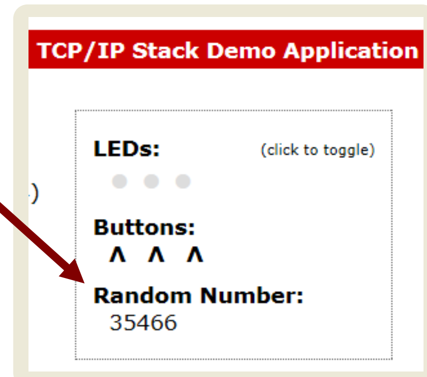
First verify the project is still configured for “pic32mz_esk”. Build the project, program the device, and run the code by clicking on this icon: 



If you run into trouble for some reason, the “pic32mz_esk_lab3” project configuration includes all code modifications that should have be made in this lab.

5 Verify the TCPIP demo web page and pet food dispenser code are both functional.

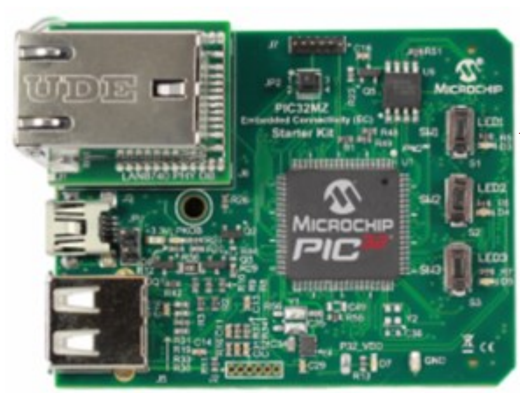
Verify the webpage is active as it was in the first lab. You should see the Random Number field constantly displaying new random numbers.



Remember you may need to reset the board (unplug/re-plug the USB cable) after programming if the webpage is not active.

Verify the code we added to simulate the pet food dispenser is operational.

Press the top push button (SW1) to light the bottom LED (LED3) for one second (dispensing food for one second).



Press this button

To light this LED

Results

Your project now has the pet food dispenser code operating simultaneously with the TCP/IP Stack.

Conclusions

Although no new functionality has been gained, the pet food dispenser is now much closer to being network-enabled. The pet food dispenser code was not originally intended to work cooperatively with other code on the same device. Integration with the stack is the most challenging task because in many cases it requires a new way of thinking about the application. We will address this issue in the following lab.

Lab Exercise 3

Remove Blocking Code

Purpose

Networked applications must handle a significant amount of background processing in order to keep the network connection alive. To accomplish these time-sensitive events, the Microchip TCP/IP Stack operates in a co-operative multi-tasking fashion. The application code you add to the stack has the potential to prevent the stack from running when it needs to.

In this lab, you will witness how blocking code can affect the TCP/IP Stack, and will learn how to remove this code from your application if it exists.

Requirements

Development Environment:	MPLAB® X IDE v2.10 or later
C Compiler:	MPLAB® XC32 v1.31 or later
Hardware Tools:	PIC32MZ EC Starter Kit (Part # DM320006)
Lab Project Location:	C:\Masters\18047\Microchip\harmony\v0_80_02b \apps\tcpip\web_server_nvm_mpfs\firmware\pic32_eth_web_server.X

Objective

- Observe how blocking code prevents the TCP/IP stack from running
- Identify and remove blocking code in your application and substitute with multi-tasking code
- Verify blocking code has been removed from your application

Procedure

1 Verify the pet food dispensing code interferes with the TCPIP stack.

Note the webpage stops working while the LED is on (observe the Random Number pause). That is because the code we added in lab 2 is not compatible with the multi-tasking code found in the TCP/IP stack. It is blocking all other code from running while it implements the one second delay. We will fix this problem in this lab.

Cooperative Multitasking

The MPLAB® Harmony integrated software framework uses a cooperative multitasking model to share the processor between all software modules. This model utilizes round-robin task switching, and relies on each task to be fair and avoid monopolizing the processor. Blocking code is to be avoided because it prevents the other tasks from completing their duties in a timely manner. If a task needs a long time to do its job it must be broken down into smaller pieces so that other tasks can have CPU time.

How long is too long?

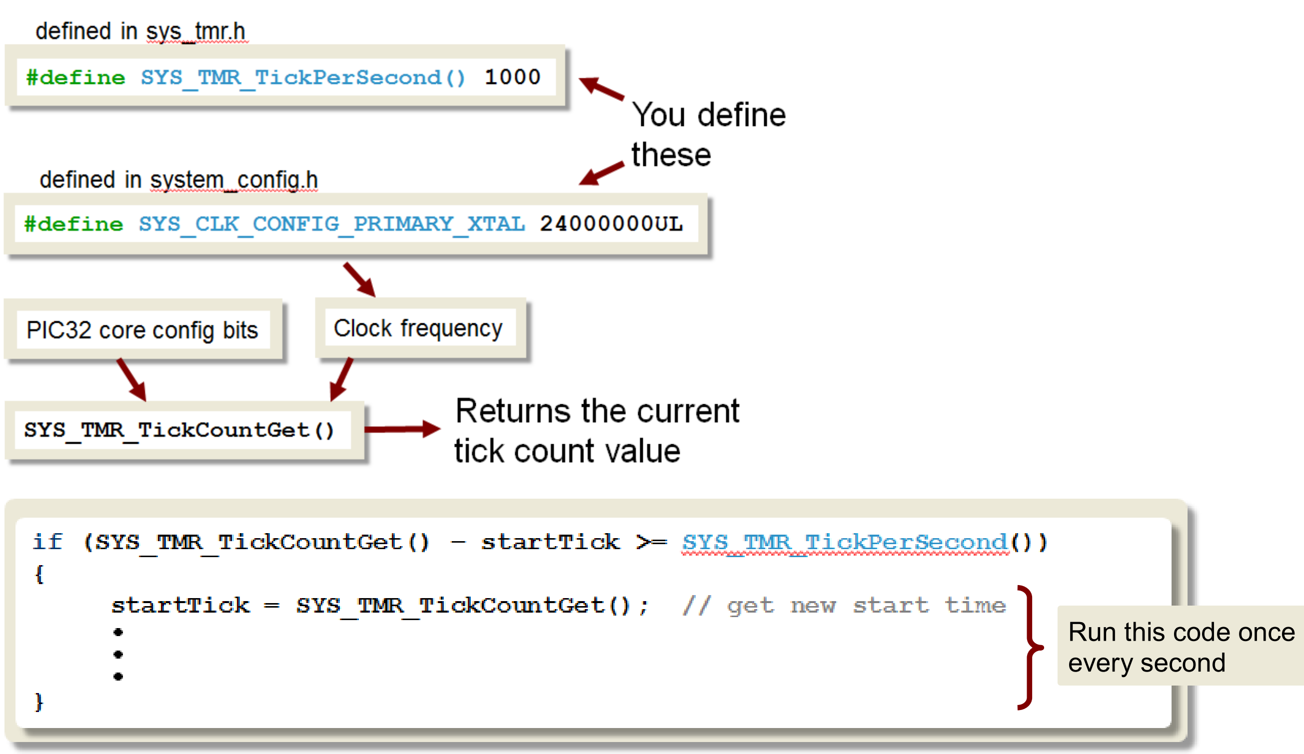
A frequent question is, “How often should the main stack loop run?” There is no strict answer, but the best performance is achieved when the stack runs as frequently as possible. Since only a fixed amount of data can be sent in one loop, frequent loops will provide the best throughput. The TCP/IP Demo App tries to keep the loop around 1 millisecond when idle, and you should try to ensure that your application does not stray too far from this goal when idle. Applications that raise this time to 10 or 20 milliseconds will still function just fine, but throughput will be lowered. Most application protocols will not time out until transfer has been interrupted for several seconds. If your application occasionally needs to consume several hundred milliseconds that will be fine. Just be aware that the stack cannot process incoming packets at this time, so ensure that your application can recover from lost packets if the Ethernet buffer becomes full. (Applications such as HTTP and SMTP that are built on TCP will recover automatically from this after a brief delay.)

“While” loops are bad

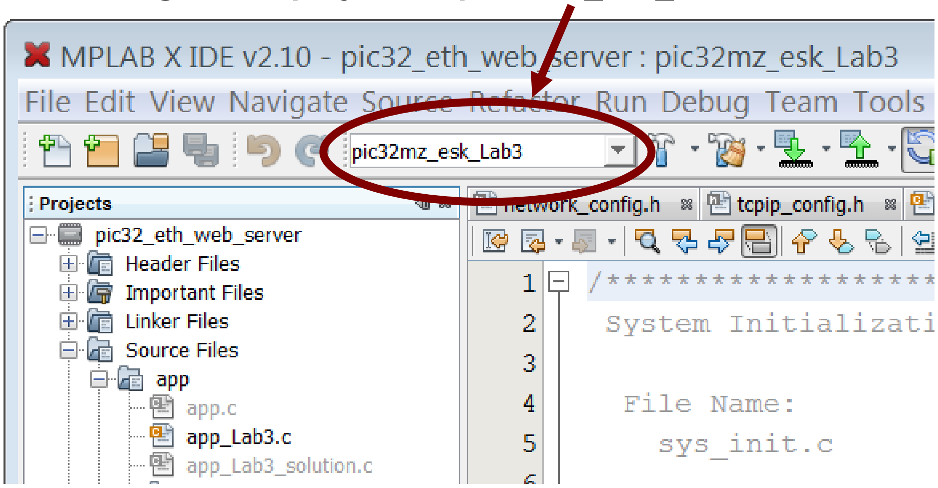
A common method for implementing delays is to cause the processor to complete some meaningless task (i.e. while (i < 100000) { i++;}). However, code constructs such as these block the processor and waste MCU resources that could otherwise be spent handling incoming Ethernet packets or other network functions. Loops of this type longer than 200 microseconds should be avoided.

Using MPLAB® Harmony’s Timer System Service Library

Instead of implementing a delay with a “while()” loop, use Harmony’s Timer System Service Library. This module is interrupt-driven, and is based on a hardware clock. It is stable and accurate, and can be used to implement non-blocking delays by comparing current times to timeouts.



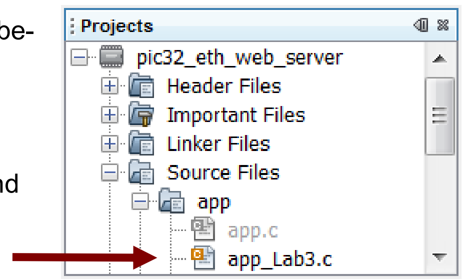
2 Use the drop-down menu to configure the project for pic32mz_esk_Lab3.

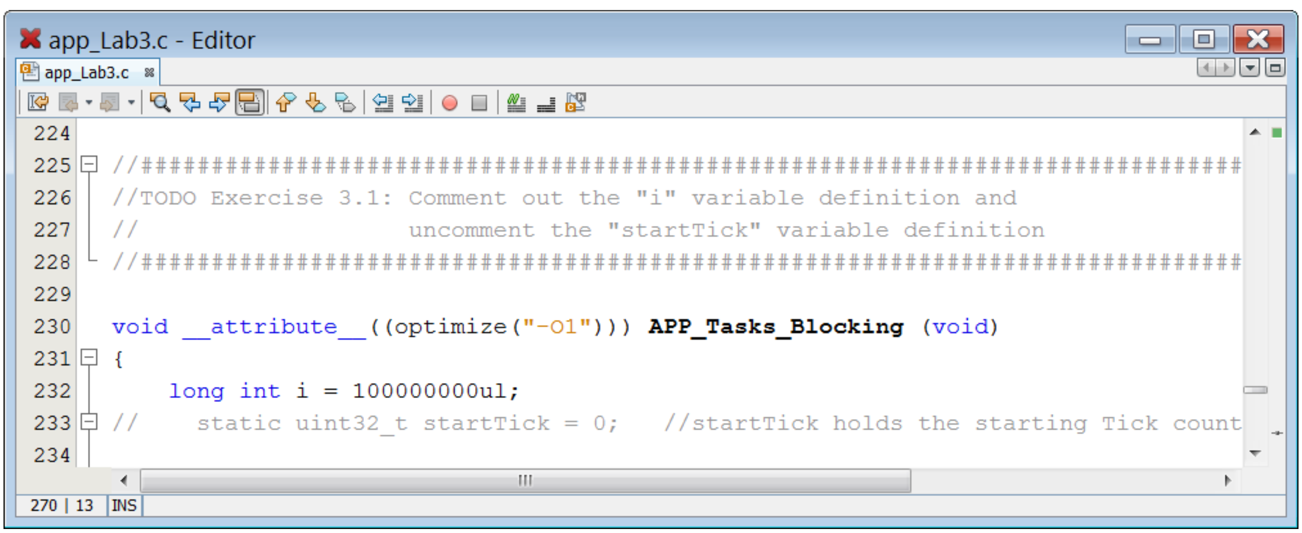


3 Modify the APP_Tasks_Blocking() function to remove the blocking code

Open the **app_lab_3.c** source file and find the section of code shown below.

Remove the loop count variable " i " declaration and uncomment the "startTick" variable declaration. The "startTick" variable will be used to hold the value of the system timer counter at the start of the one second period.





Modify the existing " if () " statement in the **app_lab3.c** source file so that it only turns on LED3 and captures the system timer counters value at the beginning of the one second period.

1) Remove the blocking " while() " loop.
2) Comment out (don't delete, we will need this later) the function that turns the LED off.
3) Assign the current value of the system timer counter to "startTick". Use the SYS_TMR_TickCountGet() Harmony function to do this (startTick = SYS_TMR_TickCountGet();).

app_Lab3.c - Editor

```c
//################################################################################
//TODO Exercise 3.2: Modify the following if statement as follows:
//          1) remove the blocking "while" statement
//          2) comment out the "BSP_SwitchLED(BSP_LED_3, BSP_LED_OFF)" function
//             (don't delete it, you will need this in the next step)
//          3) add a statement assigning "startTick" the current Tick count value
//             (use "SYS_TMR_TickCountGet()")
//################################################################################

    if(!BSP_ReadSwitch(BSP_SWITCH_1)) // if the PIC32MZ ESK switch #1 is pressed
    {
        BSP_SwitchLED(BSP_LED_3, BSP_LED_ON);     // turn on LED 3
        // your code here         //### assign current tick count to "startTick"

        while (i > 0)     // wait about one second
        {
            i--;
        }

        BSP_SwitchLED(BSP_LED_3, BSP_LED_OFF);   // turn off LED 3
    }
```

262 | 71 INS

Uncomment the code below in the **app_lab3.c** source file. This new " if() " statement tests if one second has passed. Use the Harmony function removed from the previous " if() " statement to turn off LED3 if true.

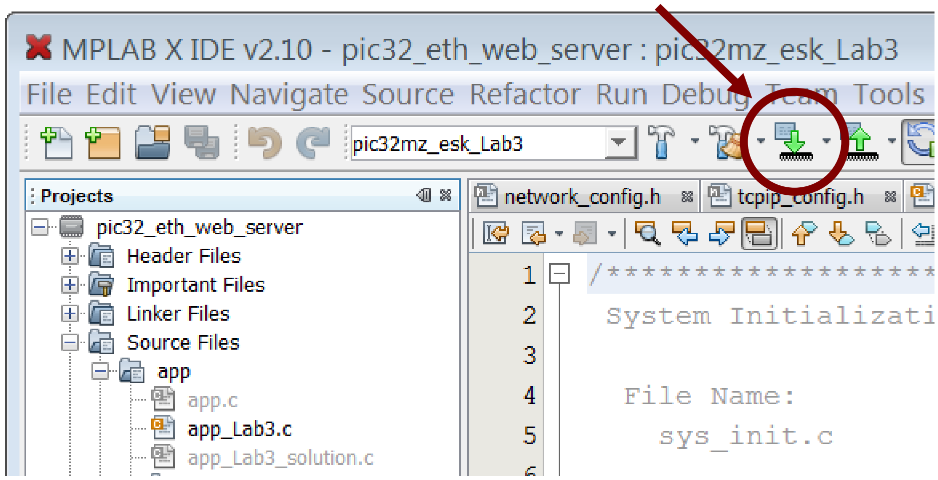
app_Lab3.c - Editor

```c
//################################################################################
//TODO Exercise 3.3: Uncomment the following "if" statement to implement a
//                   non-blocking one second delay.  Insert the function you
//                   removed from the previous "if" statement to turn off LED #3
//                   if more than one second has passed.
//################################################################################
//    if(SYS_TMR_TickCountGet() - startTick >= SYS_TMR_TickPerSecond())
//    {                                         // if more than 1 sec has passed
//        // your code here   //### turn off LED 3
//    }
}
```

250 | 10 INS

4 Build the project and program the device

Build the project, program the device, and run the code by clicking on this icon: 



5 Verify the pet food dispenser code no longer blocks the TCP/IP demo webpage

First verify the pet food dispenser code is still functioning as before. Press the top switch to see the bottom LED light for one second.

The TCP/IP demonstration webpage should continue to function while the LED is on. You should also note that unlike the previous lab, the webpage shows the LED turn on for one second.

6

Just for fun

Use the Harmony functions and macros used in the previous steps to turn on LED3 for five seconds after switch #2 (SW2 in the middle) is pressed.

Results

When the lab is completed correctly, the code that turns the LED on for one second will no longer prevent the TCP/IP stack from functioning.

Conclusions

The pet food dispenser code is now completely integrated with the Harmony TCP/IP stack, and neither task will interfere with the other. With this complete, you now have the ability to add communications between the pet food dispenser and the TCP/IP stack (Was there already food in the bowl? Are you about to run out of food?).

This was an exceedingly simple application. A more complex application would need to be broken up into states and a state machine (switch statement) would have to be implemented.

The principles learned in these past two labs apply to all stack applications:

1. Break long tasks into states, allowing the stack to execute frequently.
2. Use the Harmony Timer System Service Library module instead of blocking loops for timed events.

At this point we've learned how to integrate existing application code with the Harmony TCP/IP stack. We've learned how to identify blocking code and how to replace it with non-blocking Harmony code.