Masters 2014 Class Abstract:

Welcome to the MPLAB® Harmony TCP/IP Stack! If you plan to use a PIC32 in an embedded TCP/IP application, you will need to know how to use the MPLAB® Harmony TCP/IP stack. You will learn the parts of the stack fundamental to all TCP/IP applications and will learn how to interface your application to the stack.

This class will show you the supported protocols, example demo code and support utilities provided by the stack. We will describe the architecture of the stack and how it works, and show some common stack APIs used to interface your application with the stack (socket programming).

Note: This class is not relevant for Microchip's stand-alone Roving Networks (RN) TCP/IP modules.

Objectives

When you walk out of this class you will be able to:

- Describe what the TCP/IP stack is and how it works
- Configure the TCP/IP stack for your specific needs
- Prepare your application's source code to work cooperatively with the TCP/IP stack
- Use commonly used APIs to communicate with the stack

Agenda
Part 1

- **What does the MPLAB® Harmony TCP/IP stack Provide?**
  - General description
  - Supported protocols & security
  - Stack utilities (MPFS2, TCP/IP discoverer)
  - Debugging with Console
    - **Dynamic Startup/Shutdown, Configuration, Benchmarking**
  - What demos are provided?
    - **Supported Devices and demo boards**
- **Stack Architecture**
  - Show directory structure and describe how it works

MICROCHIP MASTERs 2014

Agenda
Part 2

- **Using the Stack**
  1. Start with a known-good system
     - **Verify successful network connection.**
     - **Verify functional TCP/IP applications.**
  2. Add your existing code to the system
     - **Verify your application is functional.**
  3. Verify your code doesn't block the stack
  4. Communicate with the stack to enable data transfer
  5. Port the code to your own PCB

Optional Labs After Class

Lab 1 **Connect to an evaluation board via TCP/IP**

- Configure the stack
- Program the development board
- Interact with the TCP/IP demo webpage

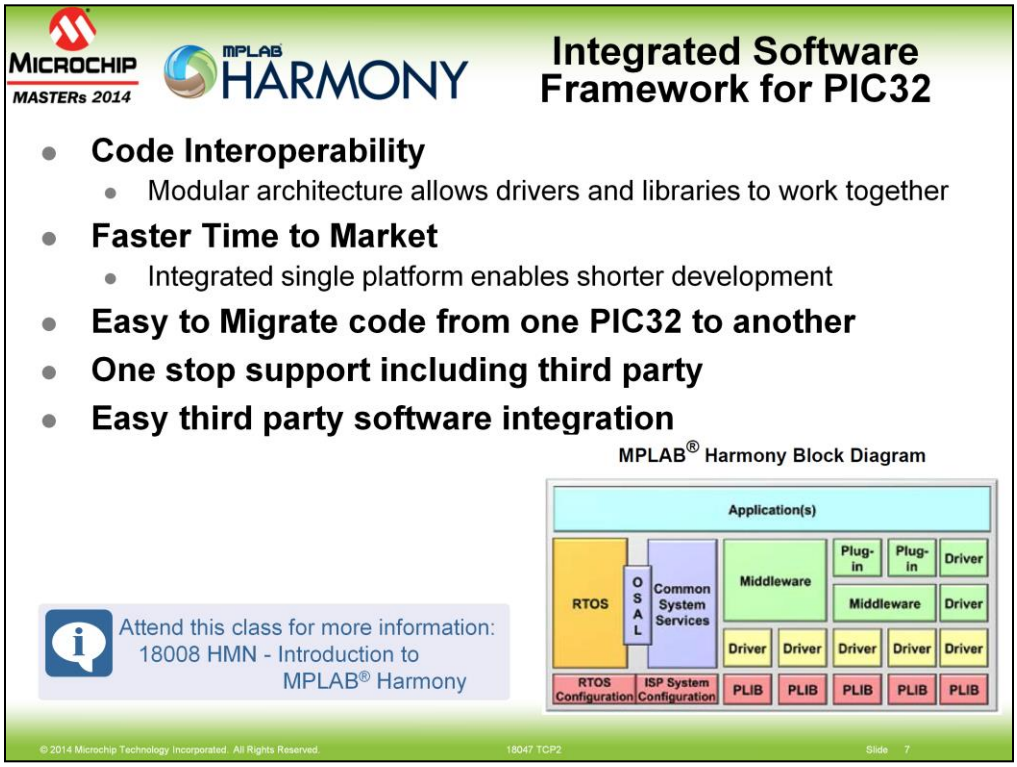
Lab 2 **Integrate an example application with the stack**

Lab 3 **Remove blocking code from example application**

MICROCHIP
MASTERs 2014

PART 1

WHAT DOES THE MPLAB® HARMONY TCP/IP STACK PROVIDE?



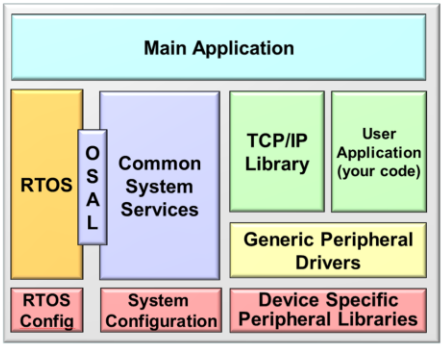
Microchip has invested expertise, resources and focus into the Next Generation Middleware & Software Ecosystem Development for over 3 years and we continue. It is a ground up design, development, review, testing, release and documentation process.

MPLAB Harmony is more than just libraries and middleware, it is a revolution in Microchip's approach to creating software. It is a new process for software design, development, testing and documentation. It consists of a set of peripheral libraries, drivers, system services, middleware, and third-party code that will make it easier to create all types of applications. It comes with examples that show proper abstraction and use of the new software tools, such as device drivers, that is new to the Microchip software development platform. The code is written in a highly reusable format, MPLAB Harmony will save everyone using it significant time and effort. As a direct result of the MPLAB Harmony process, users now have the capability of running more than one stack or more than one instance of a stack interface at a time. MPLAB Harmony also creates standard Application Programmer Interfaces (APIs), and naming conventions to improve consistency and ease of use. MPLAB Harmony implements an Operating System Abstraction Layer (OSAL), which allows seamless integration of commercial RTOS such as FreeRTOS, Micrium and others.

MLA Middleware will continue to be available on Microchip WEB site indefinitely. MLA support will continue for both 8- and 16-bit MCUs HOWEVER for PIC32 100% development is focused on MPLAB Harmony.

MPLAB® Harmony TCP/IP Stack

- **Supports PIC32 devices only**
  - TCP/IP for 8 and 16 bit devices use older MLA versions
- **IPv6 support**
- **Fully dynamic:**
  - Stack initialization/de-initialization
  - Interface up/down
  - Resource management
  - Module configuration
- **Multiple interfaces (Ethernet and WiFi)**
- **Run-time configuration (console)**
- **Interrupt driven operation**
- **RTOS friendly**



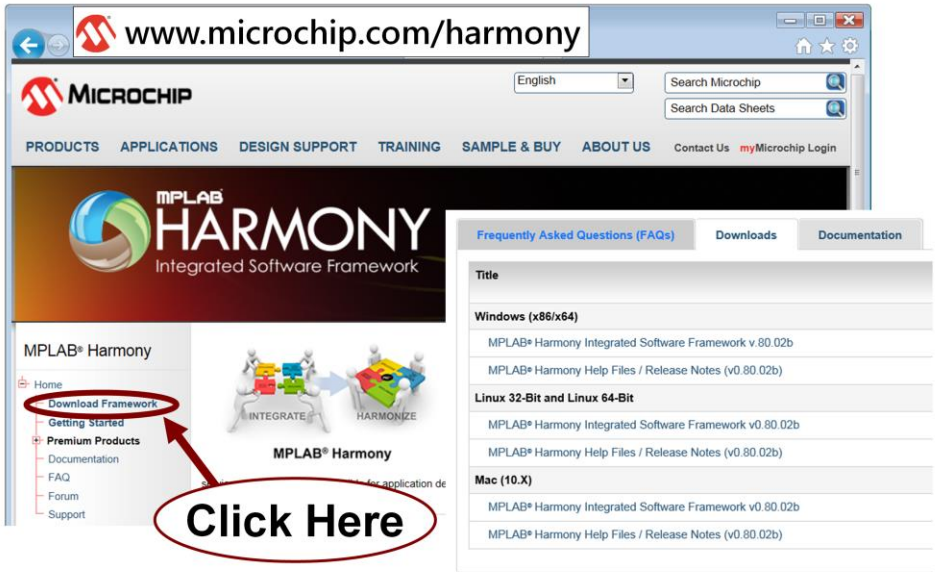
The MPLAB Harmony TCP/IP Stack is designed as a part of a system running other applications, middleware, etc. Therefore, it

makes use of the system services that are available to other modules in the system or to the application, such as:

- File system
- System interrupts
- System driver services
- System timers
- System device drivers
- System command processor
- System console
- System debug services

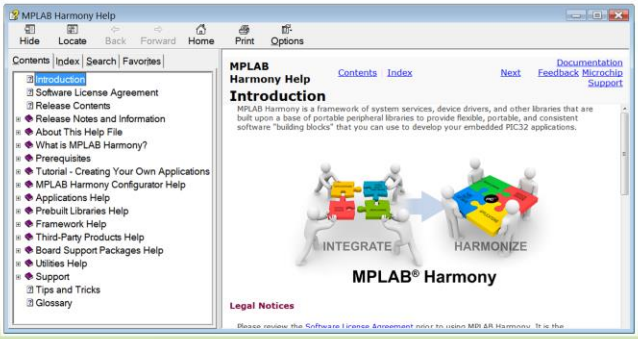
Refer to the ./framework/tcpip/src/system folder for the header files that expose the system wide available API.

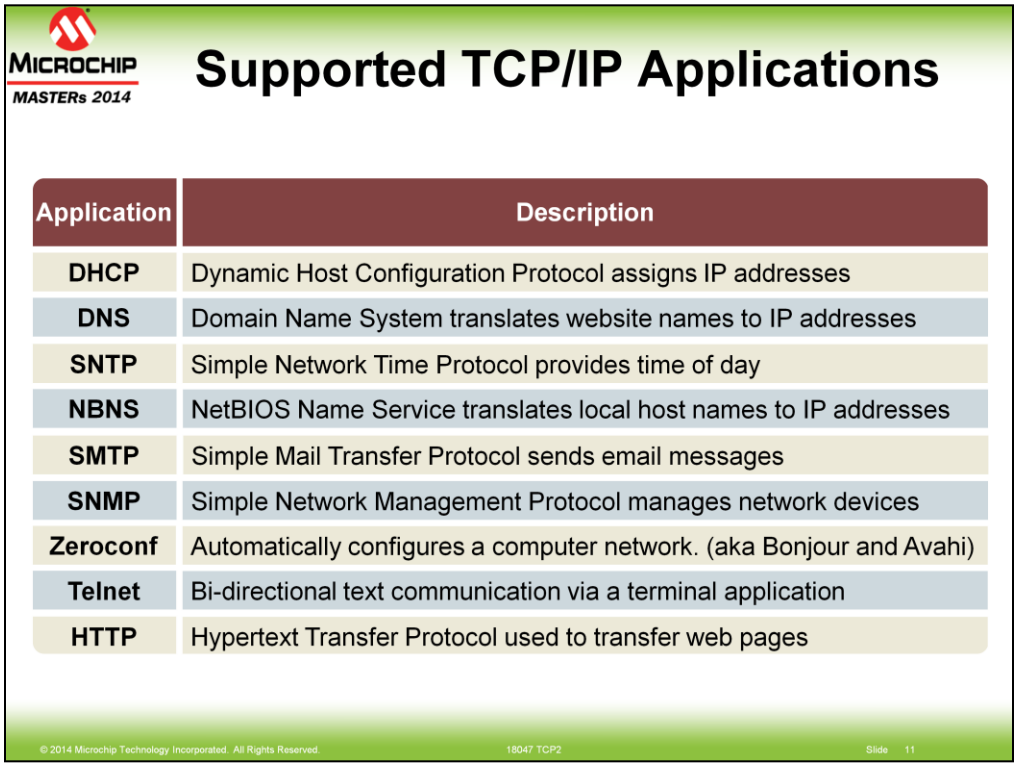
Downloading the Stack



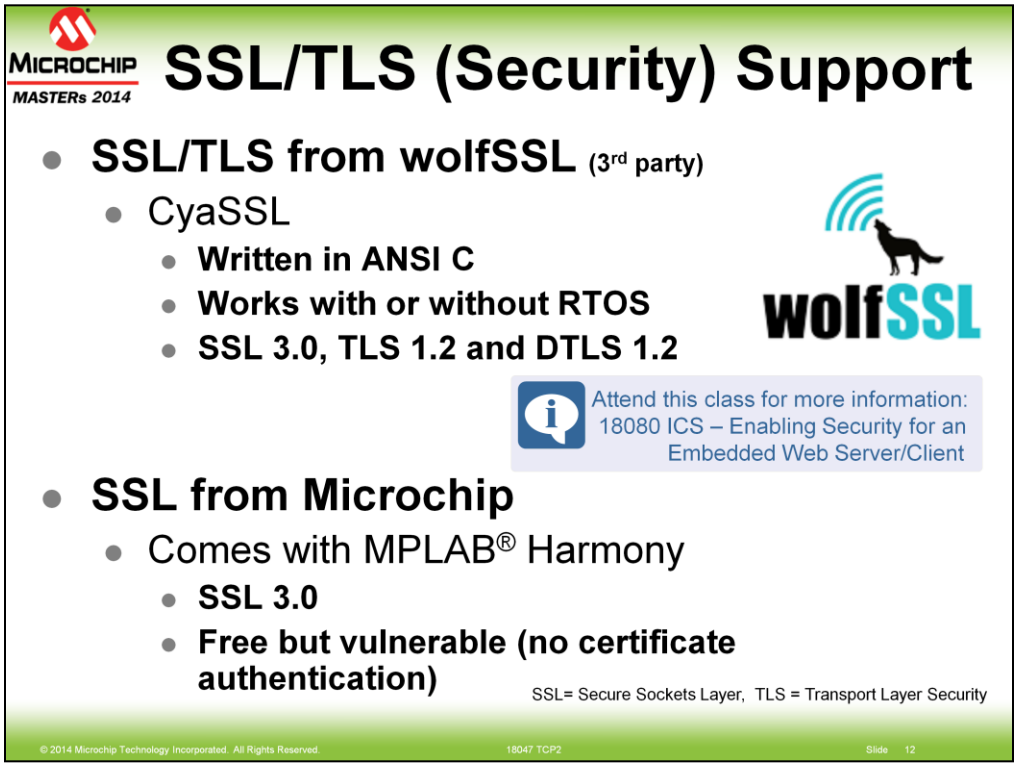
Stack Help and Documentation

- **Stack Documentation is available in the "doc" directory:**
  - help_harmony.chm
  - help_harmony.pdf





Zero Configuration (Zeroconf) IPV4 Link Local Addressing Module for the Microchip TCP/IP Stack. Zeroconf provides a mechanism to ease the configuration of a device on a network. It also provides for a more human-like naming convention, instead of relying on IP addresses alone. Zeroconf also goes by the names Bonjour (Apple) and Avahi (Linux), and is an IETF standard. Zeroconf is built on three core technologies: assignment of numeric network addresses for networked devices, automatic distribution and resolution of computer hostnames, and automatic location of network services, such as printing devices.

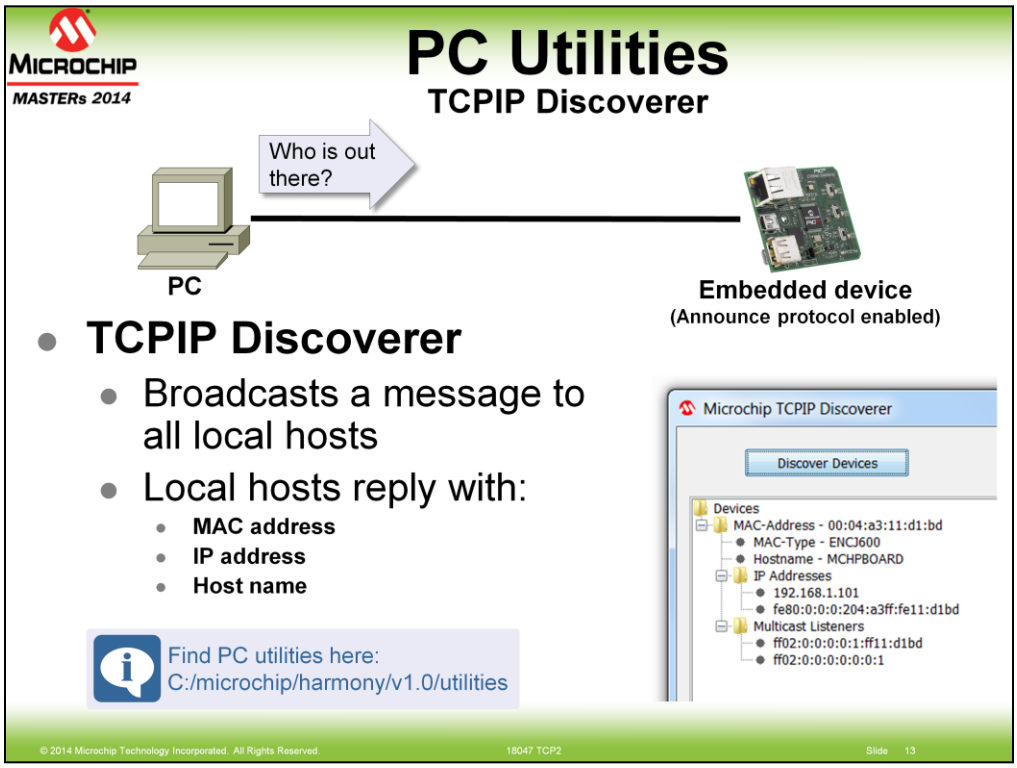


wolfSSL's CyaSSL Embedded SSL Library is ***not*** based on the OpenSLL cryptography library and therefore was not impacted by the Heartbleed virus. wolfSLL is here at Masters. See them for pricing details.

Microchip's Embedded SLL Library is also ***not*** based on OpenSLL. This library does not support TLS and is therefore not recommended if security is important. To comply with US Export Control restrictions, the encryption portion of the SSL module must be purchased separately from Microchip. The library of Data Encryption Routines (SW300052) is available for a nominal fee from microchipDIRECT.

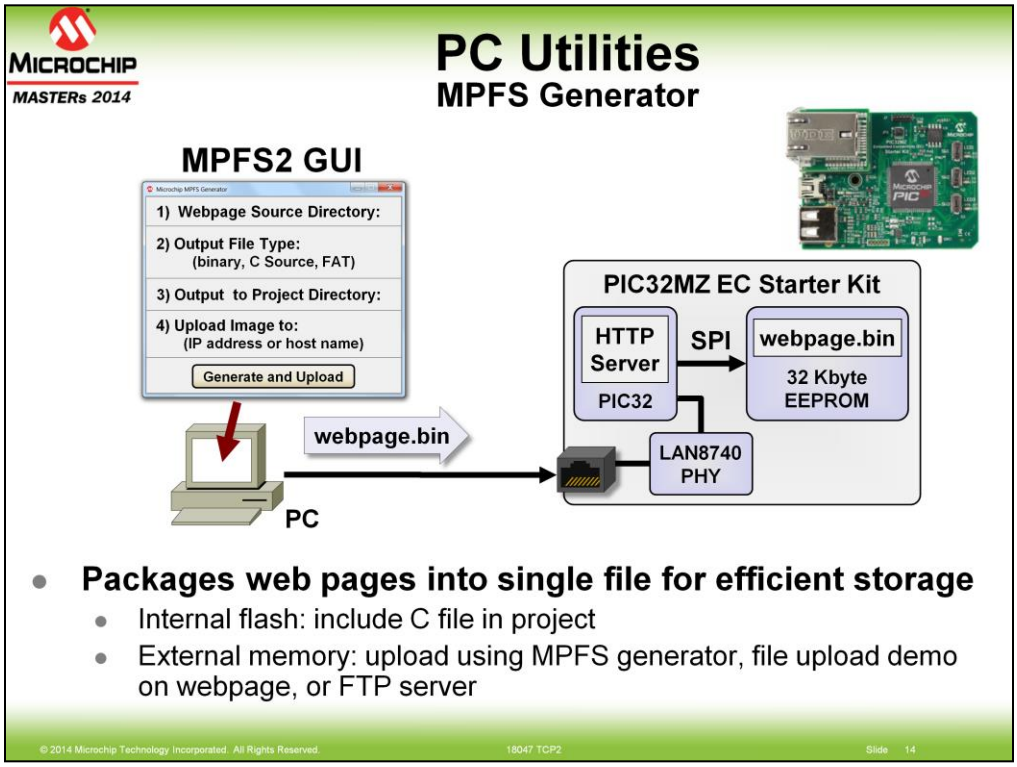
SSL (Secure Sockets Layer) v3.0 encrypts data using a shared key between network hosts. It is the predecessor to TLS and was first released in 1996.

TLS (Transport Layer Security) is based on SSL. Makes it harder for "man-in-the-middle" (MITM) attacks.



The Microchip TCP/IP Discoverer PC project will aid in embedded product device discovery by using the Announce protocol. It will also demonstrate how to write PC applications to communicate to embedded devices.

When the "Discover Devices" button is clicked, this application will transmit a broadcast UDP packet containing the message, "Discovery: Who is out there?" on the local network to port 30303. If any embedded devices with the Announce protocol enabled are connected to the network, they will respond with a UDP packet containing their MAC-Address, MAC-Type (interface type), Hostname (NBNS), IP Addresses (IPv4/IPv6), and Multicast Listeners (IPv6).



The MPFS2 (Microchip File System v2) Utility packages web pages into a format for efficient storage in an embedded system. It is a graphical application for PCs that can generate MPFS2 images for storage in external storage or internal Flash program memory.

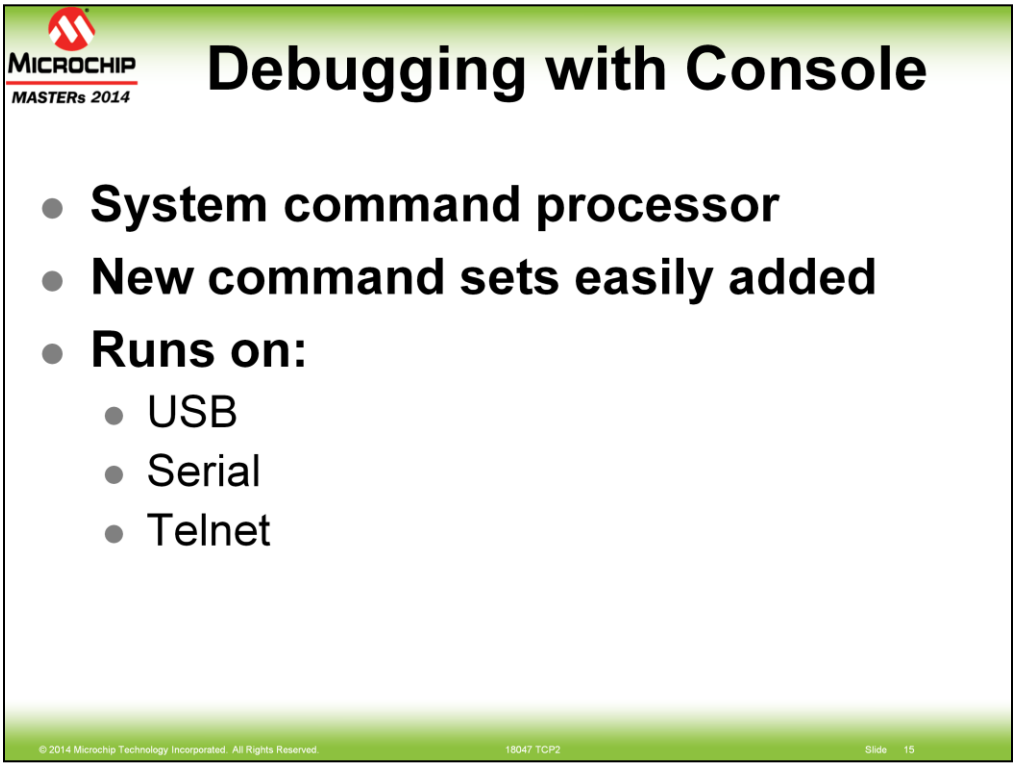
When used to build MPFS2 images, the MPFS2 Utility also indexes the dynamic variables found. It uses this information to generate http_print.h, which ensures that the proper callback functions are invoked as necessary. It also stores this index information along with the file in the MPFS2 image, which alleviates the task of searching from the embedded device.

Finally, when developing an application that uses external storage, the MPFS2 Utility can upload images to the external storage device using the upload functionality built into the HTTP web server or FTP server. http_print.idx is also generated by the utility which keeps information of all the dynamic variable details.

The MPFS2 utility generates the MPFS2SettingDetails.xml file, which contains all of the run-time configured parameters.

Important Note: If there is any change in http_print.idx , the updated MPFS image will be generated. It is recommended to remove the

http_print.idx file before generating a new MPFS image.



The Console System Service routes data or message traffic between a console device and a middleware_layer or application.

The most common use of the Console Service is to route debug or error messages from a PIC32-based device to a terminal

program on a host development system. When fully implemented, the Console Service will be capable of routing data from any

supported console device to a variety of middleware layers.

TCP/IP Console

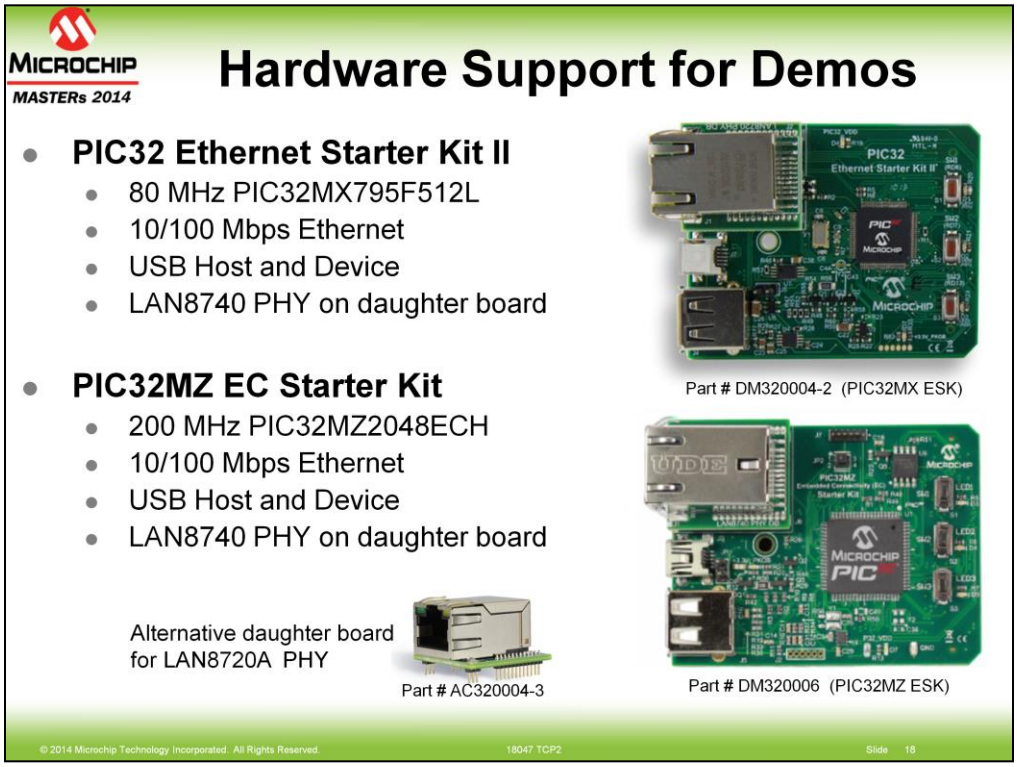
- **Network specific commands:**
  - Network info
  - Stack and interfaces up/down control
    - **Whole stack or one interface at a time**
  - IP addresses set/get
  - DHCP, NetBIOS names, etc.
  - Wi-Fi® commands included
  - TCP/IP Heap statistics

TCP/IP Demo Projects

- **The following demonstration projects come with MPLAB® Harmony:**

berkeley_tcp_client	tcpip_tcp_server
berkeley_tcp_server	tcpip_udp_client
berkeley_udp_client	tcpip_udp_client_server
berkeley_udp_server	tcpip_udp_server
snmpv3_nvm_mpfs	web_server_nvm_mpfs
snmpv3_sdcard_fatfs	web_server_sdcard_fatfs
tcpip_tcp_client	wifi_easy_configuration
tcpip_tcp_client_server	wolf_ssl

Find TCP/IP demo projects here:
C:/microchip/harmony/v1.0/apps/tcpip



Both Ethernet Starter Kits can run stand-alone or can be mounted to the Multimedia Expansion Board (MEB) or Starter Kit I/O Expansion Board to add functionality.

LAN8740 PHY

32 pin 5x5mm

Energy Efficient Ethernet (EEE) with Wake on LAN

MII/RMII

LAN8720 PHY

24 pin 4x4mm

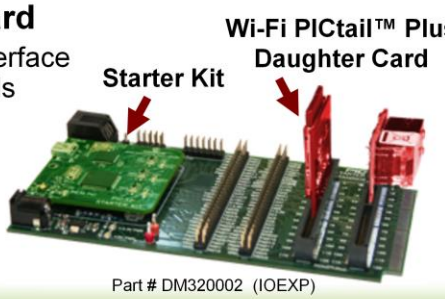
RMII

Hardware Support for Demos

- **Multimedia Expansion Board II**
  - Interfaces with PIC32 Starter Kits
  - 10/100 Mbps Ethernet & 802.11 b/g Wi-Fi
  - 4.3” WQVGA touch display
  - Bluetooth, VGA Camera, microSD slot, 24-bit stereo audio codec, accelerometer

Part # DM320005-2 (MEB-II)

- **Starter Kit I/O Expansion Board**
  - Enables PIC32 Starter Kits to interface with PICtail™ Plus daughter cards
  - Additional debug headers
  - Access to all MCU signals



Part # DM320002 (IOEXP)

Hardware Support for Demos

- **Explorer 16 board**
  - Interfaces with PIC32 and PICtail™ Plus daughter cards

Part # DM240001 (EX16)

Part # MA320012
(PIC32MZ2048ECH)

Part # MA320003
(PIC32MX795F512L)

Ethernet

Part # AC164123
(ENC28)
10 MBPS PICtail Plus

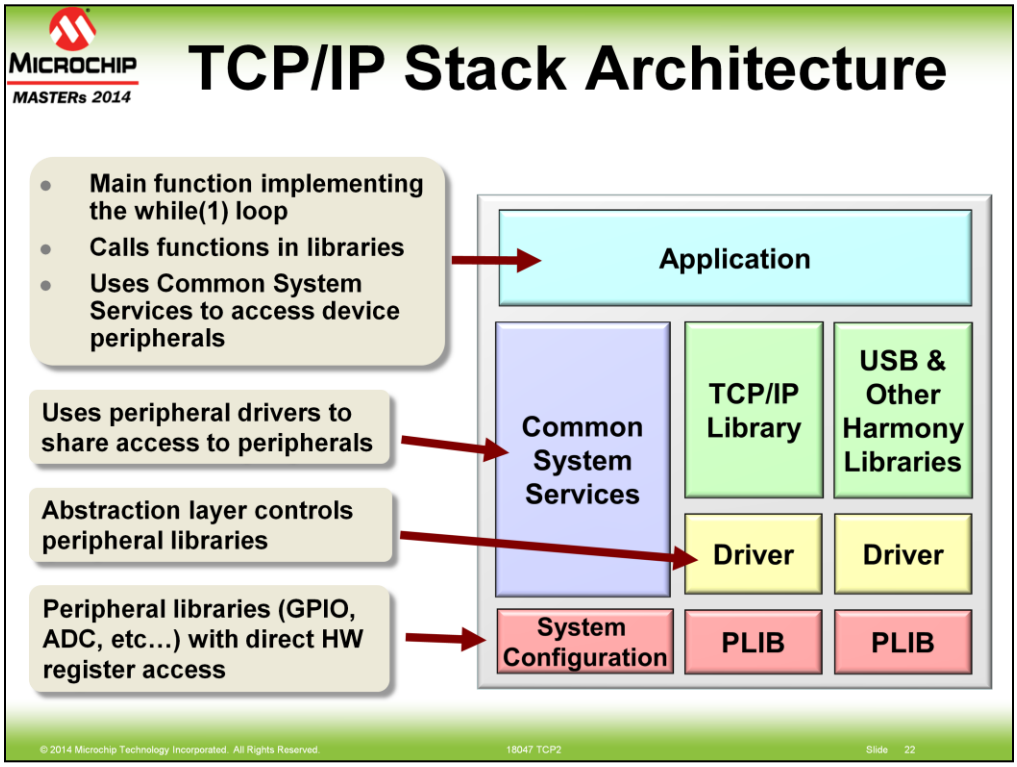
Part # AC164132
(ENC624)
10/100 MBPS PICtail Plus

Wi-Fi

Part # AC164149
(MRF24WG)
IEEE 802.11b/g PICtail Plus

MICROCHIP
MASTERs 2014

TCP/IP STACK ARCHITECTURE



MPLAB Harmony is designed to be a “stack” of interoperable modules.

Application(s)
- Implements desired overall behavior
- No Direct HW Access

Common System Services
- Initialization, Main Loop/Threads
- Manage Shared Resources

OS Abstraction Layer
- Thread Safety
- Memory allocation

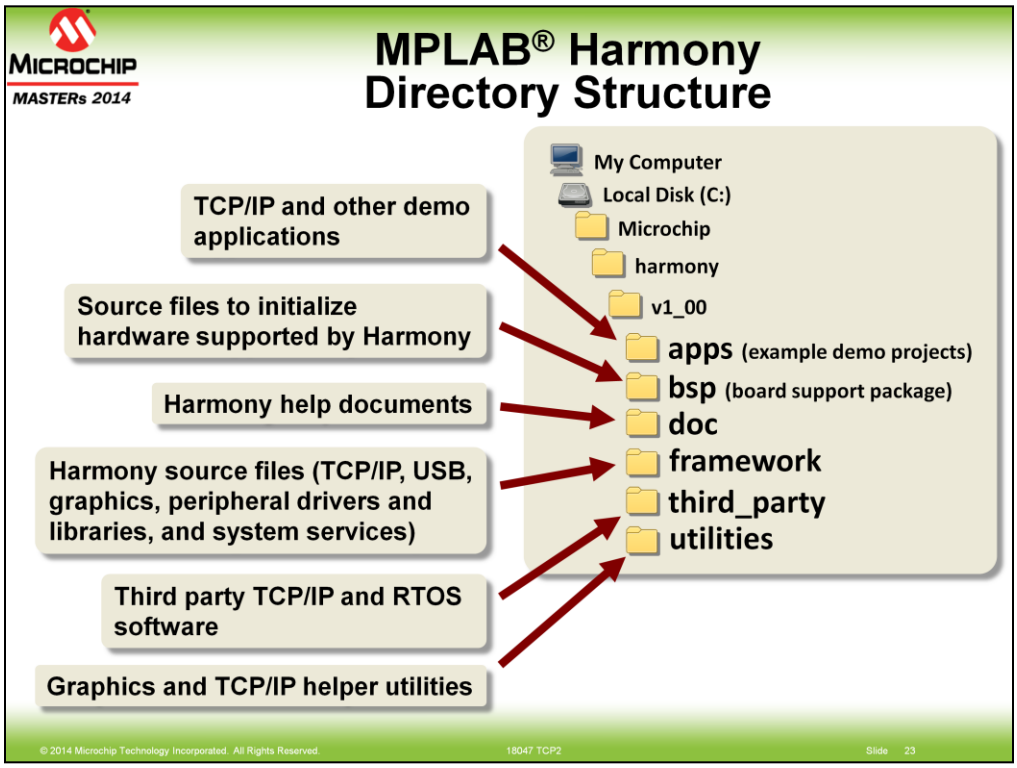
Low-Level Configuration & BSP
- Part-specific (config bits, etc)
- Initialization
- System Tasks
- Raw ISRs

Middleware Layers
- Complex Libraries & Protocols (USB, TCP/IP, FS, Graphics,…)
- Built on Drivers, PLIBS, SYS Services

Drivers
- Simple Peripheral Interface
- Manages Peripheral State
- Multiple Clients
- Multiple Peripheral Instances
- HW Access Via PLIB
- Blocking or Non-blocking

Peripheral Library (PLIB)
- Common Functional Interface for MCHP Cross-Micro Compatibility
- Part-Specific Implementations
- Exposes All Peripheral Features

- Direct HW Register Access
- No Calling Outside PLIB
- No State Management
- No Access Control



RTOS third parties that come with Harmony:
Wittenstein's FreeRTOS and OPENRTOS

Other RTOS third parties:
Micrium's MicroC/OS

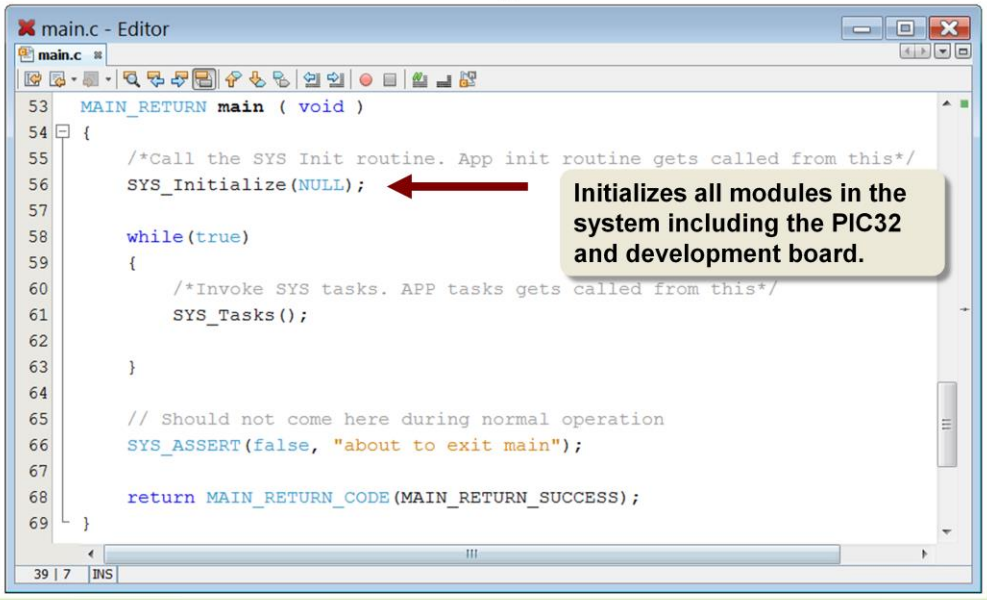
In process RTOS:
ThreadX
embOS

TCP/IP third parties that come with Harmony:
wolfSSL CyaSSL (SSL and TLS)
This is a demo only.

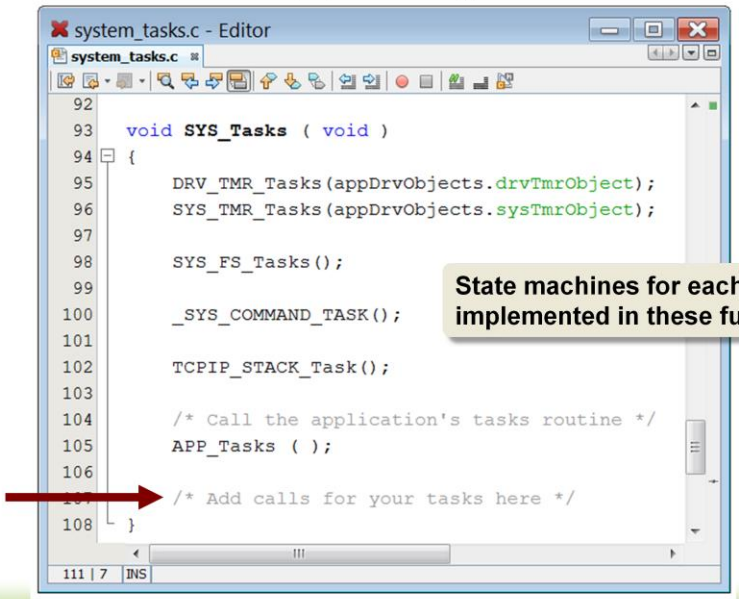
Other TCP/IP third parties:
InterNiche TCP/IP stack

TCP/IP utilities that come with Harmony:
MPFS Generator (webpage compression and upload)
TCP/IP Discoverer (display connected devices using Announce protocol)

MPLAB® Harmony main() function



MPLAB® Harmony SYS_Tasks() function

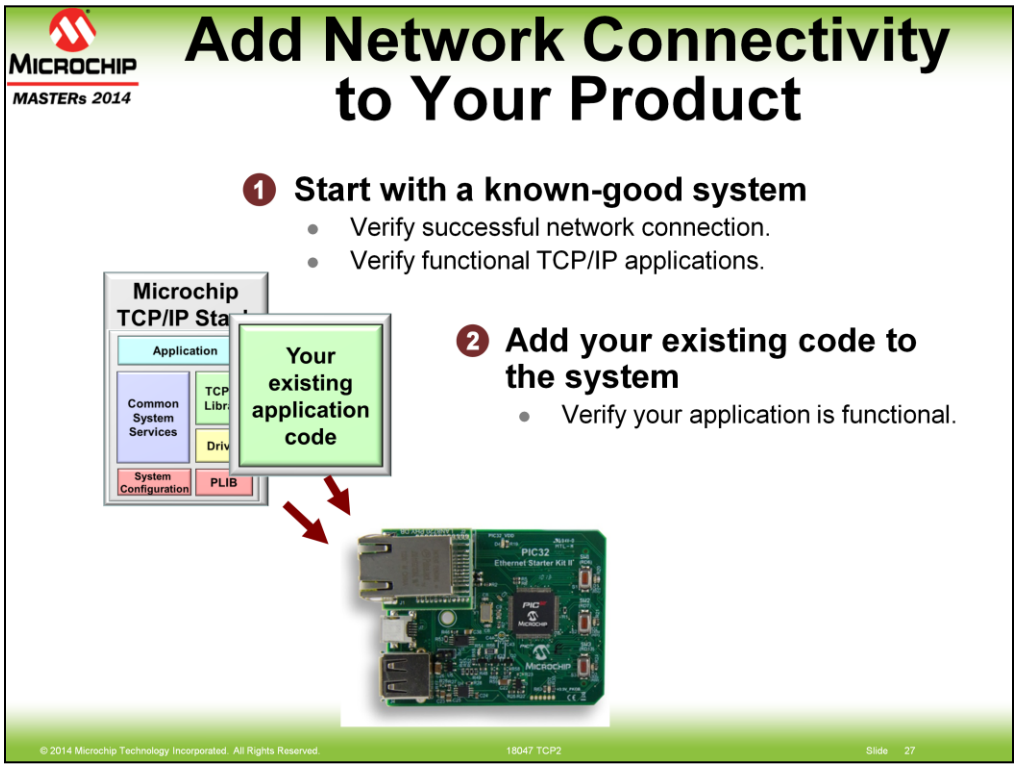


State machines for each task implemented in these functions

MICROCHIP
MASTERs 2014

PART 2

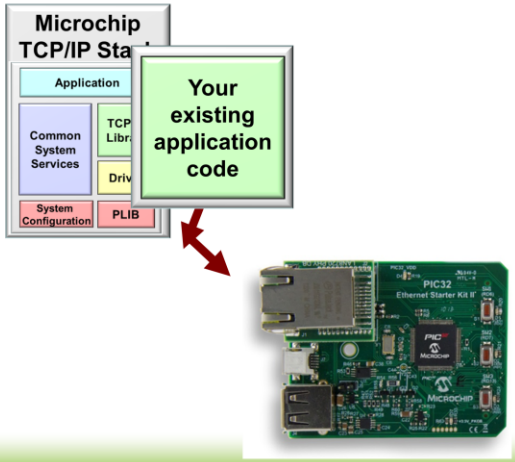
USING THE TCP/IP STACK



It's always easier to add functionality to an existing functional system than it is to design one from scratch. Prototype your application using Microchip evaluation boards and TCP/IP software. The hardware shown above is the "PIC32 Ethernet Starter Kit" with the "Starter Kit I/O Expansion Board". Refer to the TCP/IP stack documentation or "Hardware Support for Demos" slides for other supported hardware platforms.

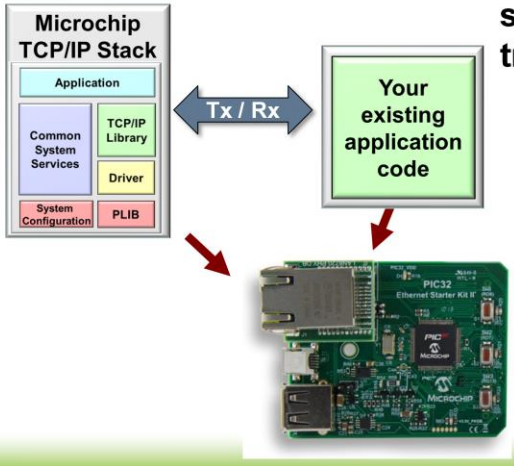
Add Network Connectivity to Your Product

3. Verify your code doesn't block the stack



Add Network Connectivity to Your Product

3. Verify your code doesn't block the stack



4. Communicate with the stack to enable data transfer

5. Port the code to your own PCB

MICROCHIP
MASTERs 2014

Using the TCP/IP Stack

1 START WITH A KNOWN-GOOD SYSTEM

Start with a Known-Good System

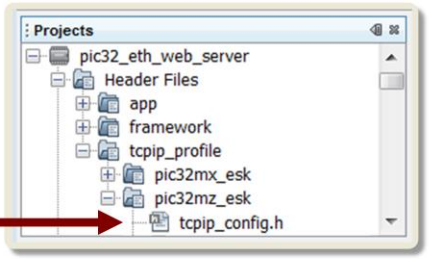
- **Find a Harmony demo project similar to your needs**

berkeley_tcp_client	tcpip_tcp_server
berkeley_tcp_server	tcpip_udp_client
berkeley_udp_client	tcpip_udp_client_server
berkeley_udp_server	tcpip_udp_server
snmpv3_nvm_mpfs	web_server_nvm_mpfs
snmpv3_sdcard_fatfs	web_server_sdcard_fatfs
tcpip_tcp_client	wifi_easy_configuration
tcpip_tcp_client_server	wolf_ssl

See the Harmony Help documentation for full descriptions of the demonstration projects above:
(Chapter 10, Applications Help: TCP/IP Demonstrations)

Configure the TCP/IP Stack

- **Enable or disable TCP/IP modules**
  - HTTP Server, NBNS, DHCP Client, DNS, etc…
  - Uncomment or comment lines in **tcpip_config.h**



tcpip_config.h

```
42  // TCPIP Stack Module Selection
43  //   Uncomment or comment the following lines to enable or
44  //   disabled the following high-level application modules.
45
46  #define TCPIP_STACK_USE_IPV4             // enable IPv4 functionality
47  #define TCPIP_STACK_USE_ICMP_SERVER      // Ping query and response capability
48  #define TCPIP_STACK_USE_HTTP_SERVER      // New HTTP server with POST, Cookies,
49  //#define TCPIP_STACK_USE_SSL_SERVER       // SSL server socket support (Require
50  //#define TCPIP_STACK_USE_SSL_CLIENT       // SSL client socket support (Require
51  #define TCPIP_STACK_USE_DHCP_CLIENT      // Dynamic Host Configuration Protocol
52  #define TCPIP_STACK_USE_SMTP_CLIENT      // Simple Mail Transfer Protocol for se
```

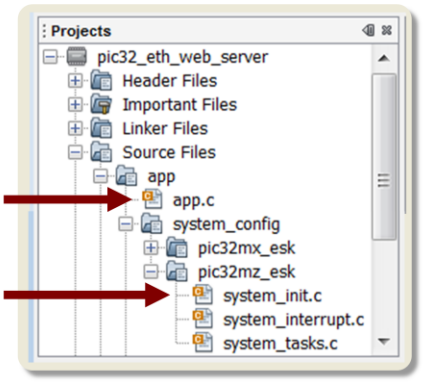
MICROCHIP
MASTERs 2014

Using the TCP/IP Stack

❷ ADD YOUR EXISTING CODE TO THE SYSTEM

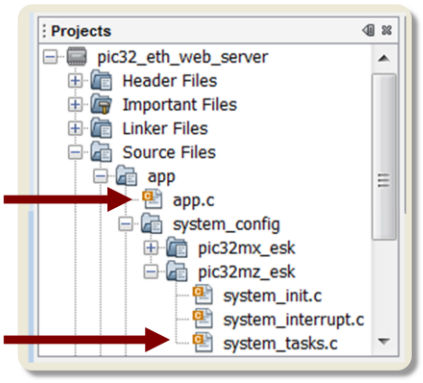
Integrate Your Code with the Stack

- **Include your source files in the project directory**
- **Initialize your code**
  - Add function to initialize your code in **app.c**
  - Call this function in SYS_Initialize() in **system_init.c**



Integrate Your Code with the Stack

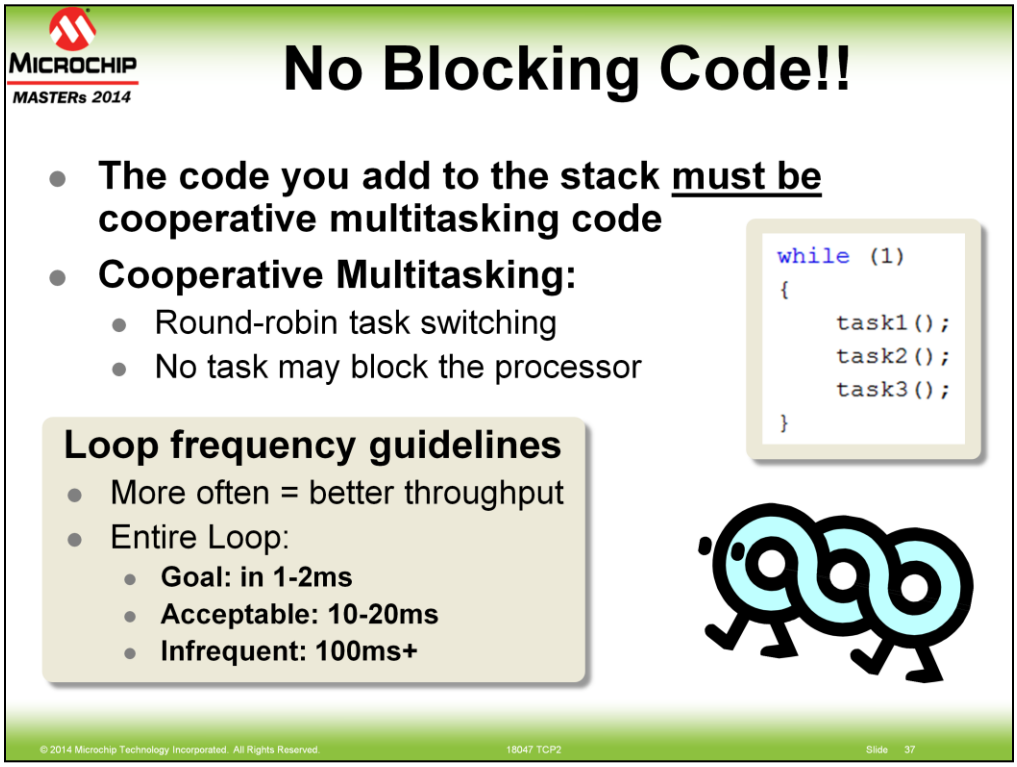
- Add functions for your code in app.c
- Add function prototypes in app.h
- Call these functions in SYS_Tasks() in system_tasks.c



MICROCHIP
MASTERs 2014

Using the TCP/IP Stack

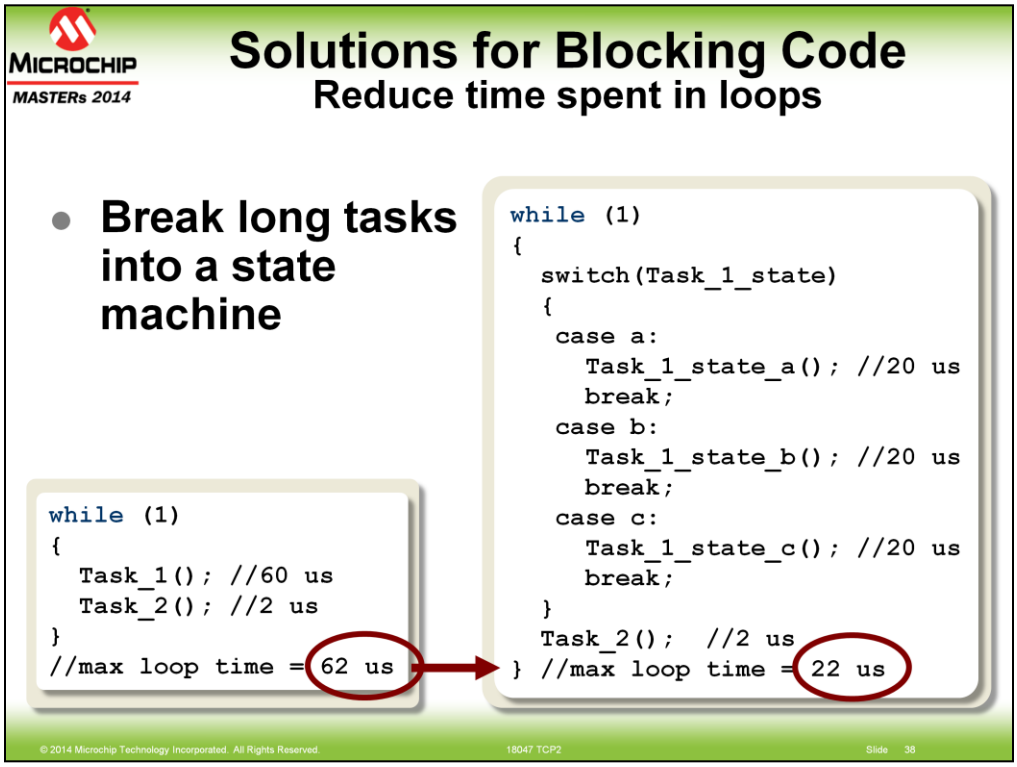
❸ VERIFY YOUR CODE DOESN'T BLOCK THE STACK



The Microchip TCP/IP Stack uses a cooperative multitasking model to share the processor between the stack and your application. This model utilizes round-robin task switching, and relies on each task to be fair and avoid monopolizing the processor. Since there is no pre-emption, blocking code is to be avoided because it would prevent the other tasks from completing their duties.

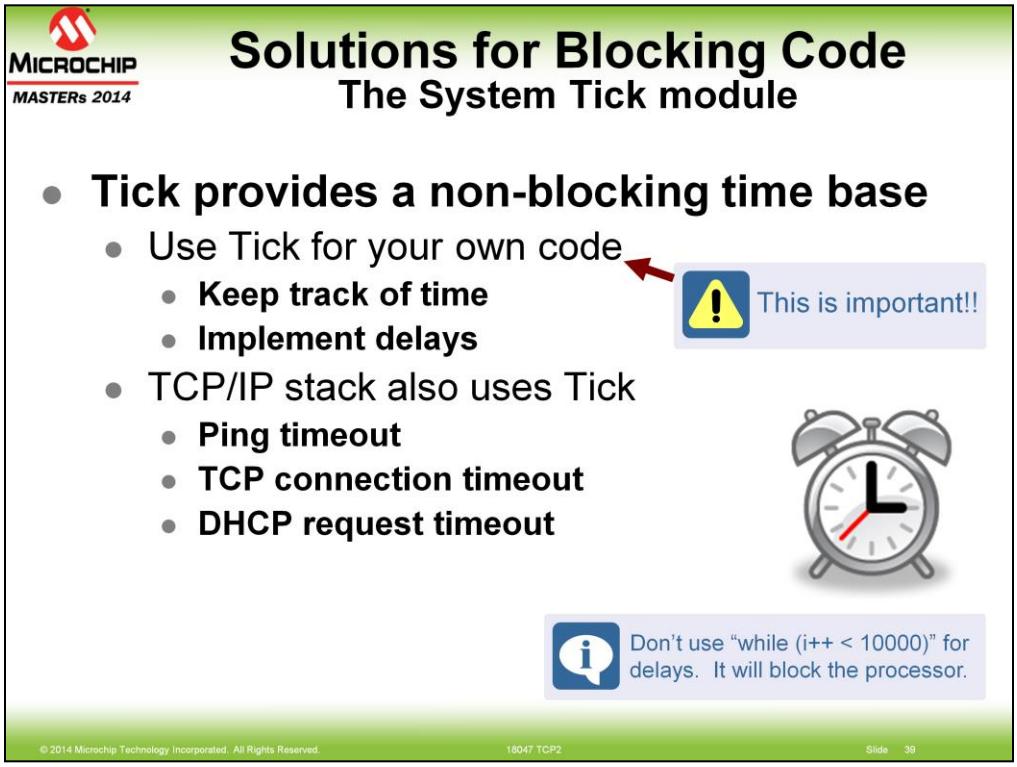
The super loop in main() will continuously execute all stack-related tasks, as well as your own application's functions. Every task performs its tasks (whether all in one shot or part of it) and returns so that other tasks can do their job. If a task needs very long time to do its job, it must be broken down into smaller pieces so that other tasks can have CPU time.

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 however, so 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.)



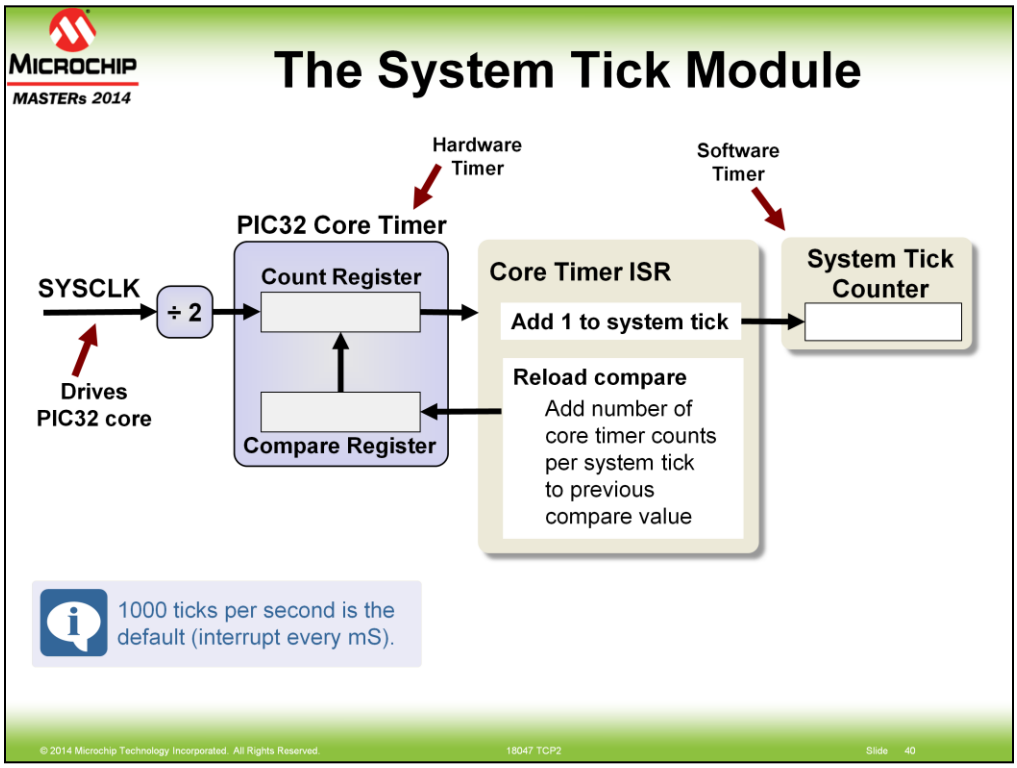
Check your application thoroughly for tasks that may require a significant amount of processing. Whenever possible, break these tasks into smaller pieces with a state machine. Process one state at a time, then return to the main stack before continuing.

Use caution when transferring or receiving data over the UART. The low baud rate means that calls to write or read strings spend a significant amount of time waiting for bytes to transfer. Consider reading and writing data to a buffer in RAM, then using an interrupt to transmit or receive the next byte when the previous byte completes.



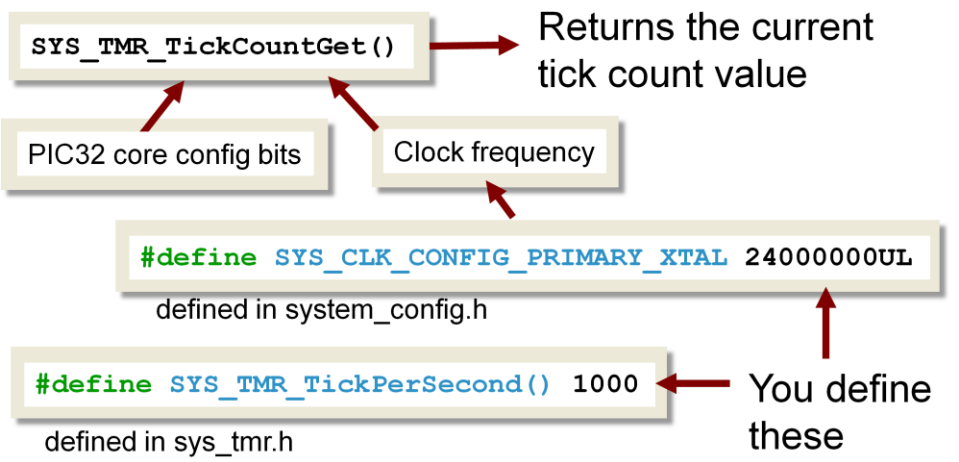
You just learned about the cooperative multitasking model, and why blocking code is to be avoided. However, many applications have a need to implement delays. A common method for doing this is to cause the processor to complete some meaningless task, such as counting to 10,000, or to call one of the Delay() functions that does this internally. Code like this blocks the processor though, and so it should be avoided.

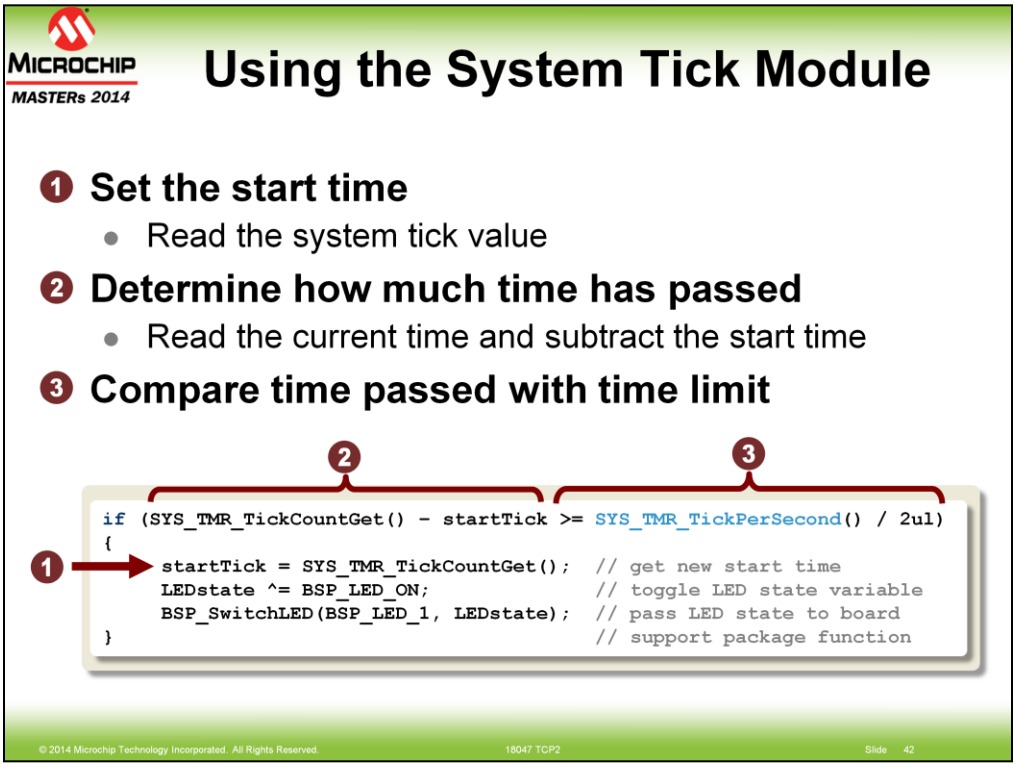
Instead, use the provided Tick module. The Tick module is interrupt-driven, and uses the PIC32's core timer to drive it. It is stable and accurate, and can be used to implement non-blocking delays by comparing current times to timeouts.



The system tick counter is 32 bits. With a system tick rate of 1000 Hz, it will overflow after 50 days.

Using the System Tick Module





This is the code that blinks an LED once every second in the demo applications while the stack is running. Every time this code runs it checks the value of the current system tick counter. It subtracts the system tick value at the start of the waiting period from the current value and compares the result the number of system ticks there are in a second divided by 2. If more system ticks have occurred than there are in half a second, the code will execute. Note this method lets us know when at least (not exactly) half a second has passed.

If (current system tick count – starting system tick count >= system ticks per second/2)

{

set starting system tick count

toggle “LEDstate” variable

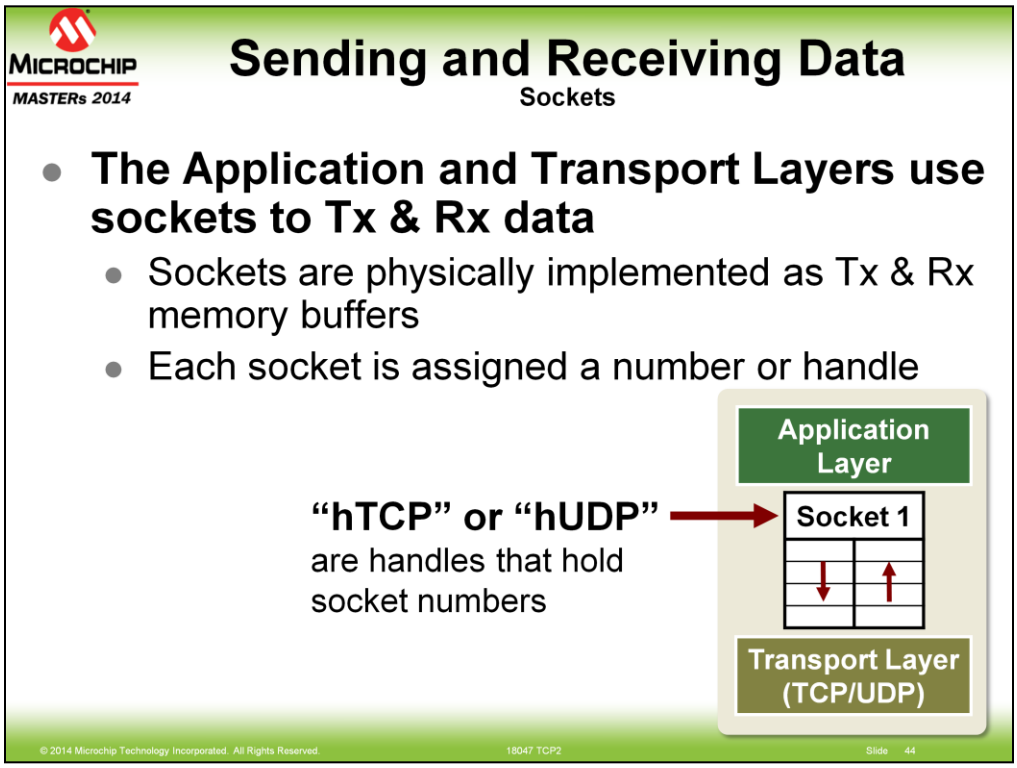
Pass “LEDstate” to system function to toggle LED 0

}

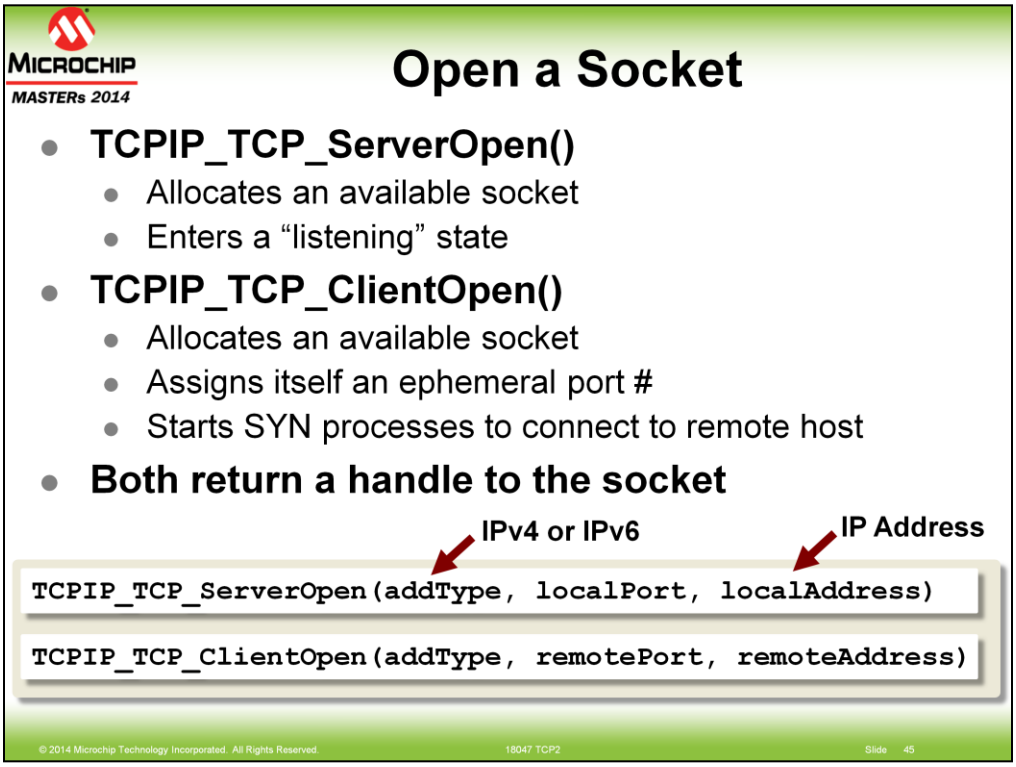
MICROCHIP
MASTERs 2014

Using the TCP/IP Stack

4 COMMUNICATE WITH THE STACK TO ENABLE DATA TRANSFER



When a socket is opened, the application is provided with a handle to reference that socket later. (Similar to the manner in which opening a file yields a file pointer handle.) The hTCP variable in all of these functions is that socket handle. One application (i.e. HTTP server) may have several sockets and own several of these handles. The application opens these sockets.

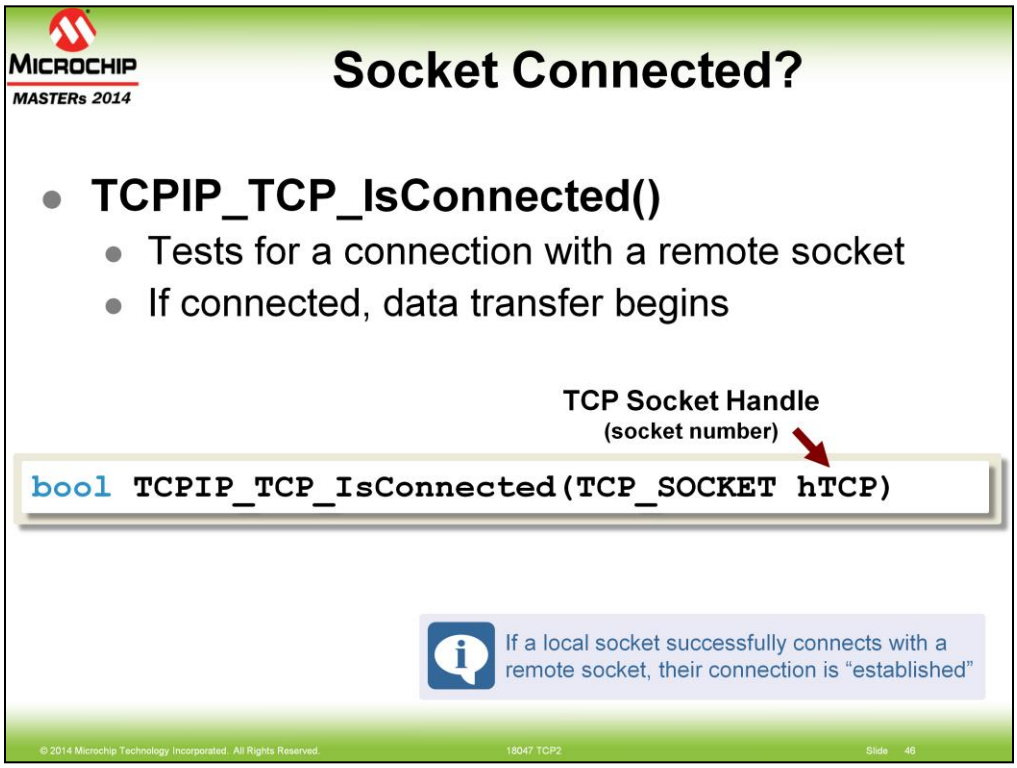


Allocate a socket to be used with your application.

The "SYN" process is a standard TCP/IP synchronization request to connect a local socket with a remote one.

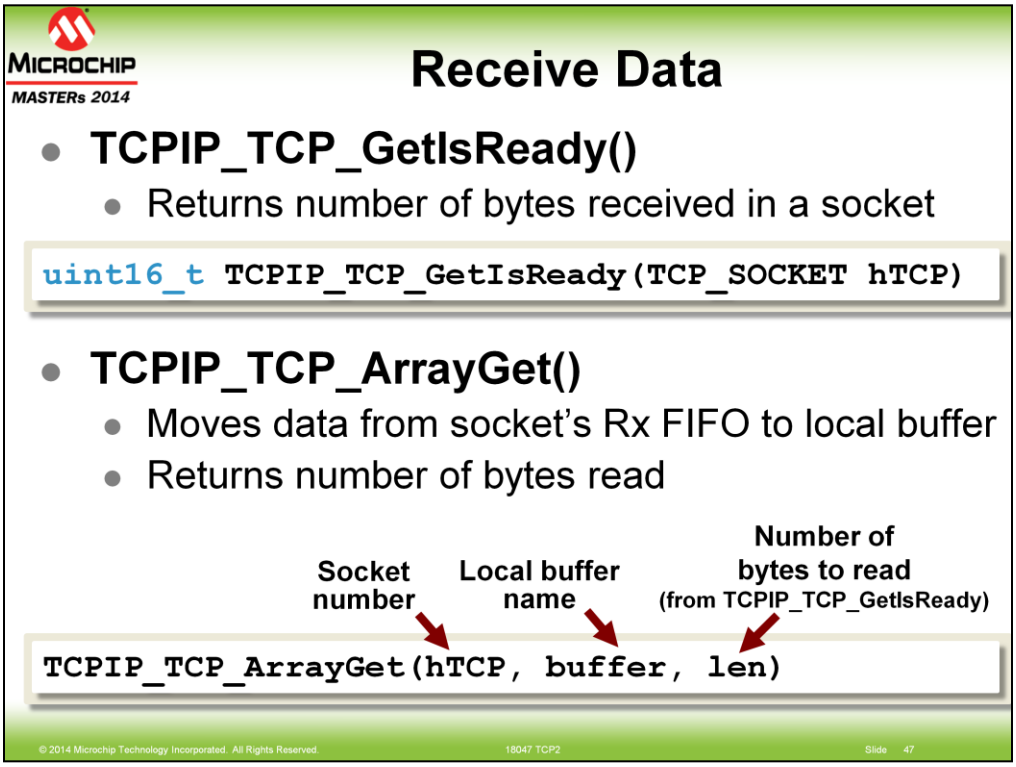
1) The client sends a request to connect with a server
2) The server ACKs the client
3) The client ACKs the server

After this SYN process the connection is "established" and data transfer can begin.

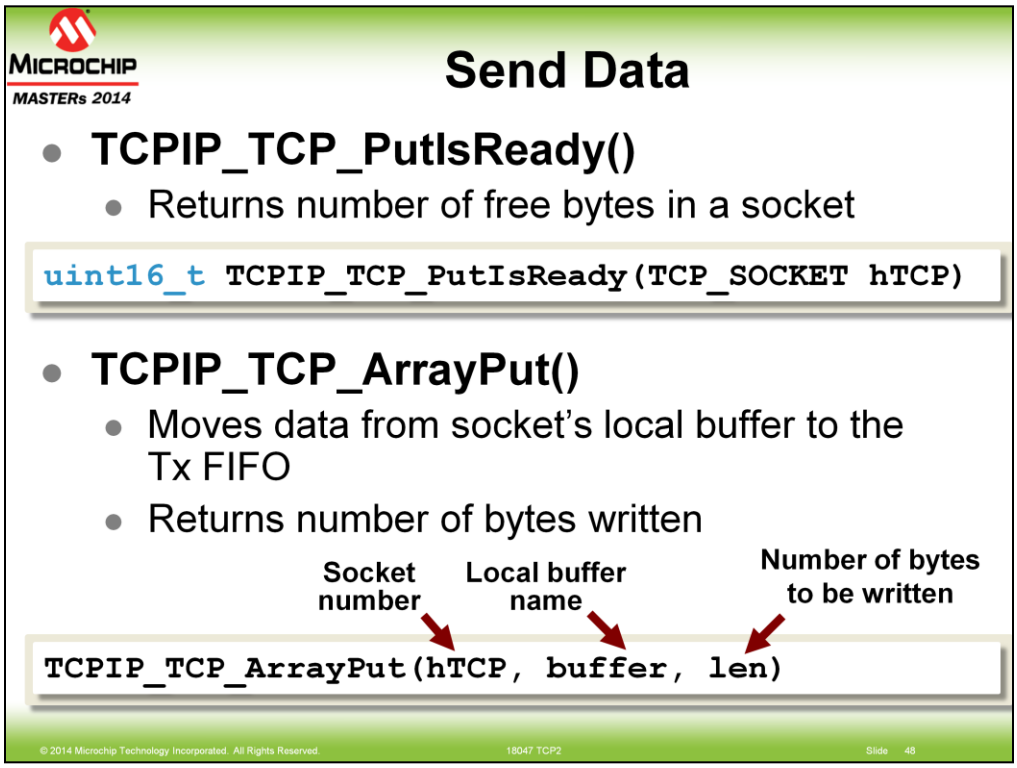


TCP_SOCKET is an "int16_t" data type. This is a C99 standard indicating this variable will always be 16 bits regardless of the compiler being used.

Returns true if connected.



uint16_t TCPIP_TCP_ArrayGet(TCP_SOCKET hTCP, uint8_t* buffer, uint16_t len) returns the # of bytes read.

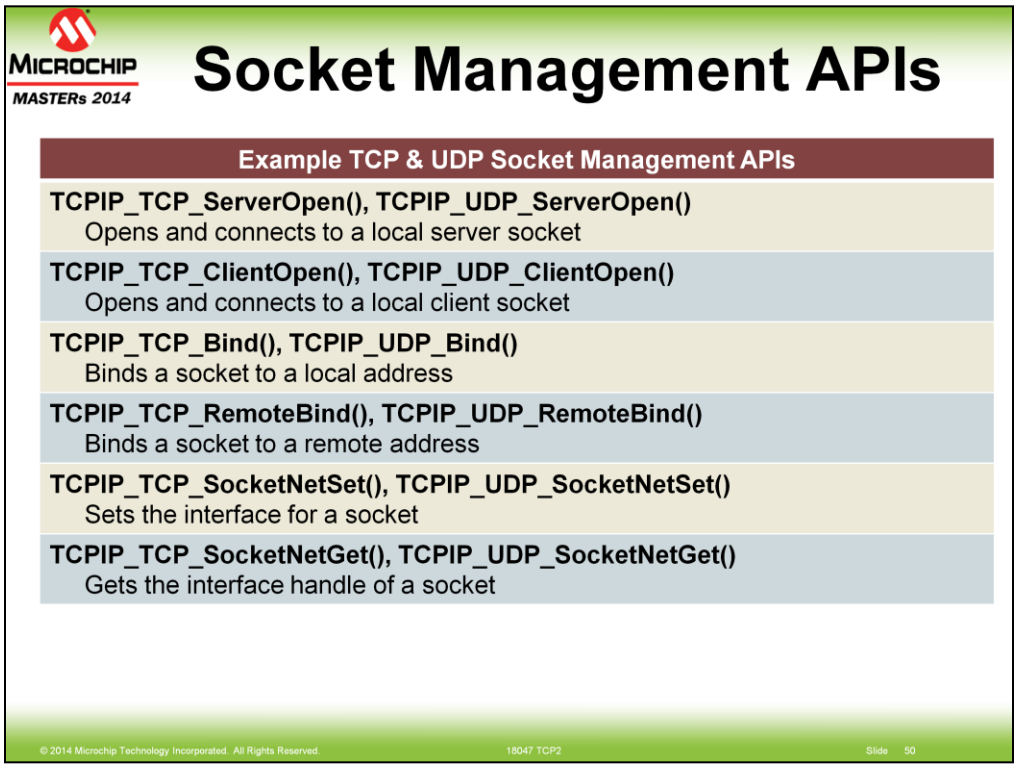


uint16_t TCPIP_TCP_ArrayPut(TCP_SOCKET hTCP, const uint8_t* data, uint16_t len) returns the # of bytes written.

Common TCP/IP Stack APIs

Example Stack APIs
TCPIP_STACK_NetDefaultGet() Used to obtain a handle to the current interface - needed for all stack queries
TCPIP_STACK_NetAddressMac (myNetHandle) Used obtain mac address of current interface

Example Application APIs
TCPIP_DHCP_IsBound(myNetHandle) Check whether DHCP process is complete (do I have an IP address?)
TCPIP_DNS_UsageBegin(myNetHandle) Start DNS resolution



When server sockets are closed, they retain their handle (socket number), and automatically go into listening mode. When client sockets close, the handle is relinquished to the stack.

Socket Read/Write APIs

TCP & UDP APIs for Sending and Receiving Data
TCPIP_TCP_Put(), TCPIP_UDP_Put() Writes a single byte to a socket
TCPIP_TCP_ArrayPut(), TCPIP_TCP_StringPut(), TCPIP_UDP_ArrayPut(), TCPIP_UDP_StringPut() Writes an array or string from RAM to a socket
TCPIP_TCP_Get(), TCPIP_UDP_Get() Reads a single byte from a socket
TCPIP_TCP_ArrayGet(), TCPIP_UDP_ArrayGet() Reads an array or string from a socket
TCPIP_TCP_PutIsReady(), TCPIP_UDP_PutIsReady() Determines number of bytes available in the TX buffer
TCPIP_TCP_GetIsReady(), TCPIP_UDP_GetIsReady() Determines number of bytes available in the RX buffer

Berkeley Sockets APIs

Example Berkeley Socket APIs
socket() Create a new Berkeley socket
bind() Assign a local port number and IP address to a socket
listen() Places a socket in listen mode (listen for a connection request)
accept() Accepts a connection request
connect() Connects a socket to a remote host port number and IP address
send() Sends outgoing data to a socket
recv() Receives incoming data from a socket

MICROCHIP
MASTERs 2014

Using the TCP/IP Stack

5 PORT THE CODE TO USE YOUR OWN PCB

Configure the Stack for Inputs

- Use Harmony to connect your switches to specific PIC32 inputs

Board Support Package Harmony Function

Descriptive name for your switch

```
BSP_ReadSwitch(BSP_SWITCH_1)
```

Associate name with GPIO port bit position #12 (bsp_config.h)

```
BSP_SWITCH_1 = PORTS_BIT_POS_12
```

Associate GPIO PORTB with switch (bsp_sys_init.c)

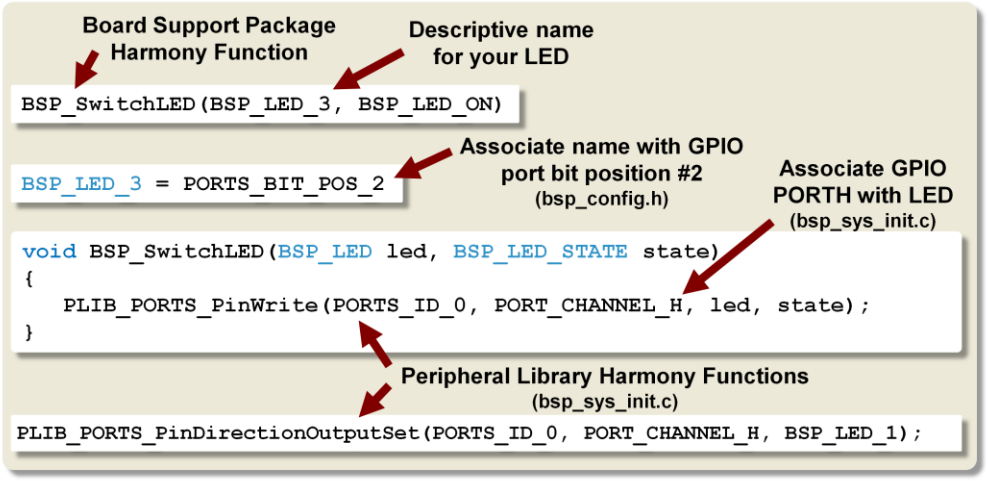
```
BSP_SWITCH_STATE BSP_ReadSwitch(BSP_SWITCH bspSwitch)
{
   return (PLIB_PORTS_PinGet(PORTS_ID_0, PORT_CHANNEL_B, bspSwitch));
}
```

Peripheral Library Harmony Functions (bsp_sys_init.c)

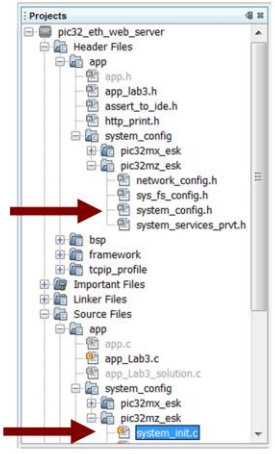
```
PLIB_PORTS_PinDirectionInputSet(PORTS_ID_0, PORT_CHANNEL_B, BSP_SWITCH_1);
```

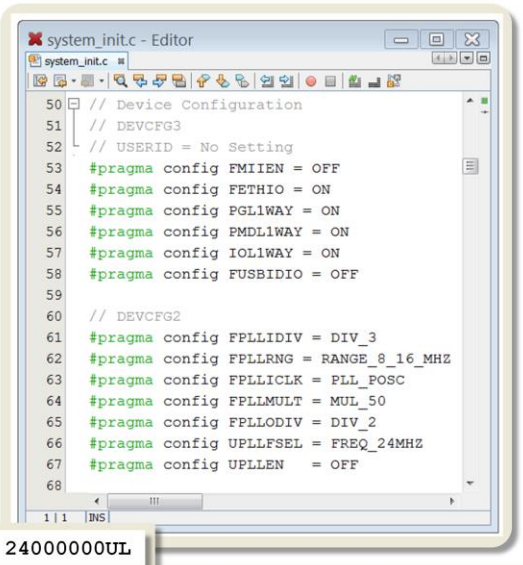
Configure the Stack for Outputs

- Use Harmony to connect specific PIC32 outputs to LEDs



Device Configuration Bits and Oscillator Frequency





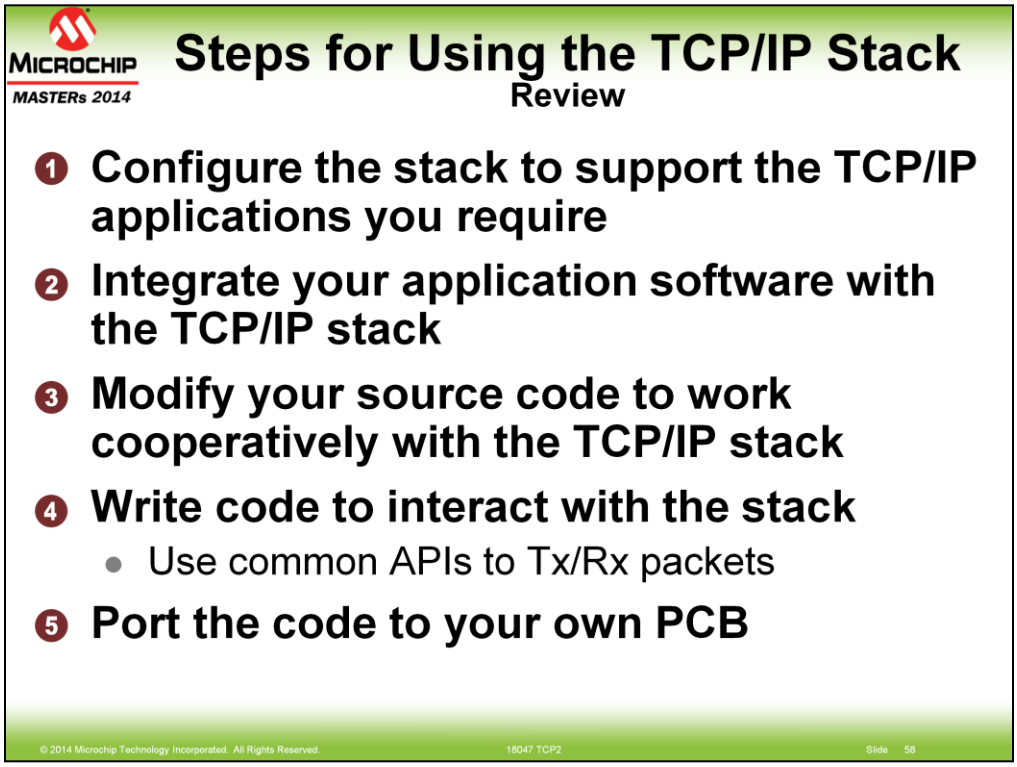
Define External Oscillator Frequency (system_config.h)

```
#define SYS_CLK_CONFIG_PRIMARY_XTAL 24000000UL
```

MICROCHIP
MASTERs 2014

Review

USING THE TCP/IP STACK



In the previous section, we discussed planning for applications. We explained the cooperative multitasking model used by the stack. Remember to ensure that your application conforms to round-robin task switching, and does not block the processor.

Next we talked about the stack's Tick module. Tick lets you implement timers without writing blocking loops. Compare stored times against multiples of TICK_SECOND and you can implement a timeout without blocking.

Finally, we discussed application integration and porting your code to your own PCB.

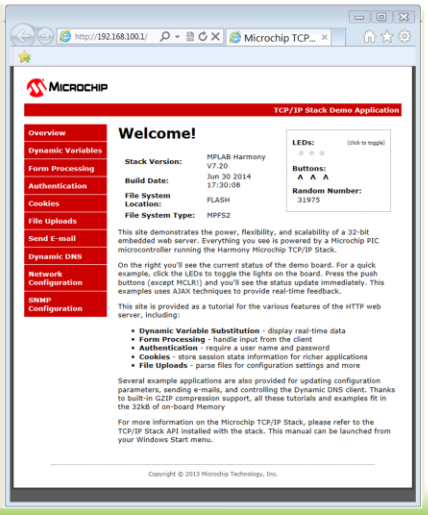
MICROCHIP
MASTERs 2014

Join the Network
LAB 1

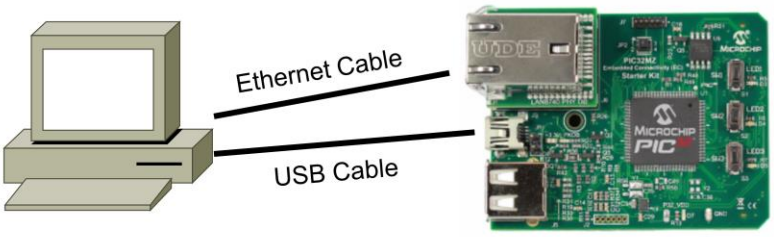
Lab 1
Join the Network

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**



Lab 1
Hardware Setup

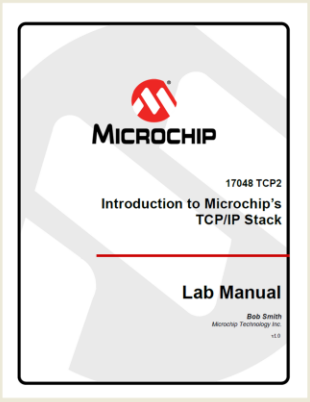


Part # DM320006
(PIC32MZ ESK)

Lab 1
Join the Network

Procedure

Follow the directions in the lab manual on page 1-1



MICROCHIP
MASTERs 2014

Integrating an Application with the Stack

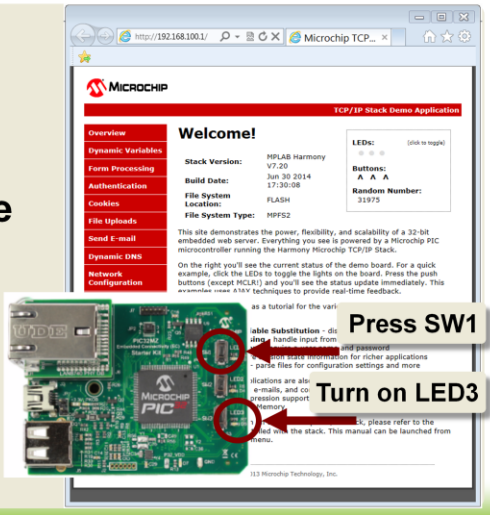
LAB 2

Lab2

Integrating an Application with the Stack

Objective

- Combine an existing application with the TCP/IP stack
- Identify where to put your application's code within the TCPIP stack
- Verify both the TCP/IP stack and your application are functioning

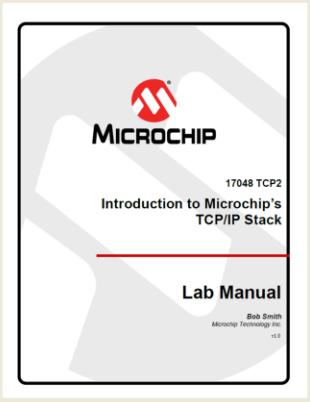


Lab2

Integrating an Application with the Stack

Procedure

Follow the directions in the lab manual on page 2-1



MICROCHIP
MASTERs 2014

Remove Blocking Code

LAB 3

Lab3

Remove Blocking Code

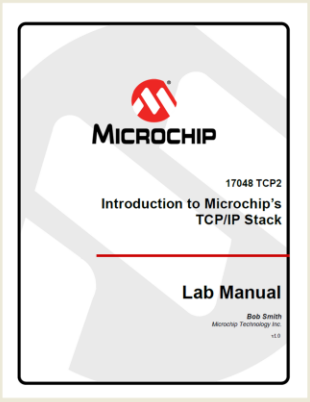
Objective

- 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

Lab3
Remove Blocking Code

Procedure

Follow the directions in the lab manual on page 3-1



MICROCHIP
MASTERs 2014

LEGAL NOTICE

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. PLEASE BE AWARE THAT IMPLEMENTATION OF INTELLECTUAL PROPERTY PRESENTED HERE MAY REQUIRE A LICENSE FROM THIRD PARTIES.

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 and mTouch are registered trademarks 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, 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.