MICROCHIP MASTERs 2019

23075 IoT6

Simplifying TCP/IP Applications with MPLAB® Harmony

SMART | CONNECTED | SECURE

MICROCHIP

Class Objectives

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

- Understand the fundamentals of TCP/IP and how applications use them to create TCP/IP connections.
- Understanding Network analysis with the well-known tool Wireshark
- Learn the architecture and the fundamentals of the Harmony TCP/IP stack to interface your application with some common stack API's in a FreeRTOS Task Schedular enviroment

Agenda – Lecture

Part 1: TCP/IP Fundamentals as a Refresher

Part 2: MPLAB® Harmony TCP/IP Stack Overview

- TCP/IP Layers and Features
- Network Interface Options
- Processor Requirements

Part 3: Using the MPLAB® Harmony TCP/IP Stack

- Creating a new project, stack configuration and **connectivity check** (**Lab 1**)
- MPLAB® Harmony TCP/IP Stack APIs
- Application integration for **local access**, using the example of a Vending machine (**Lab 2**)
- Application integration for **external access** using the example of a Weather Service (**Lab 3**)

Agenda – Labs

- **Lab 1:** Creating a new project, Stack Configuration and **Connectivity Check**
- **Lab 2:** Application integration for **local access**, using the example of a Vending machine
- **Lab 3:** Application integration for **external access**, using the example of a Weather Service

MICROCHIP MASTERs 2019

Part 1: TCP/IP Fundamentals as a Refresher

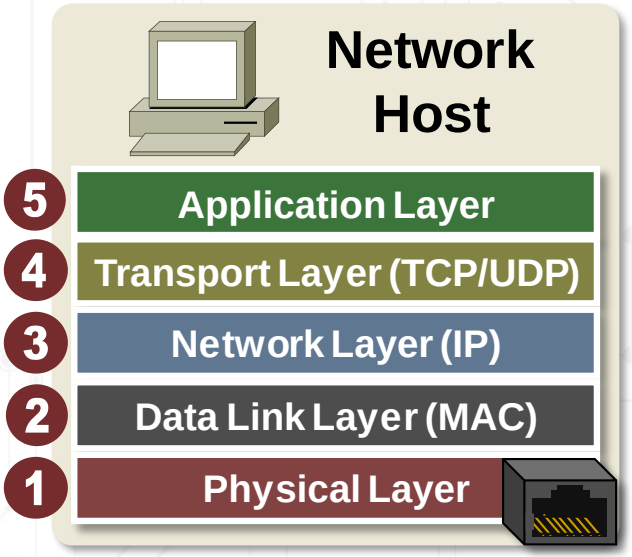
- Five Layer Model and Applications
- TCP vs UDP

SMART | CONNECTED | SECURE

MICROCHIP

Agenda
Five Layer Model and Applications

- **Five Layer Software Model**
  - Application Layer
  - Transport Layer
  - Network Layer
  - Data Link Layer
  - Physical Layer

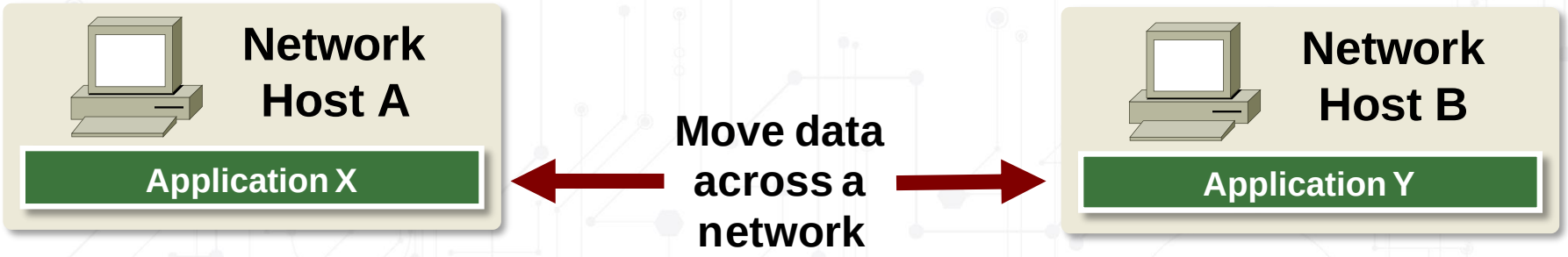


- **TCP vs. UDP**
- **TCP/IP Applications**
  - DNS, NBNS, SNTP, DHCP, SNMP, Telnet, SMTP, HTTP

Basic Needs for TCP/IP Transmissions

- **Need to Specify:**
  - Most reliable or fastest transmissions
  - Where we want the data delivered

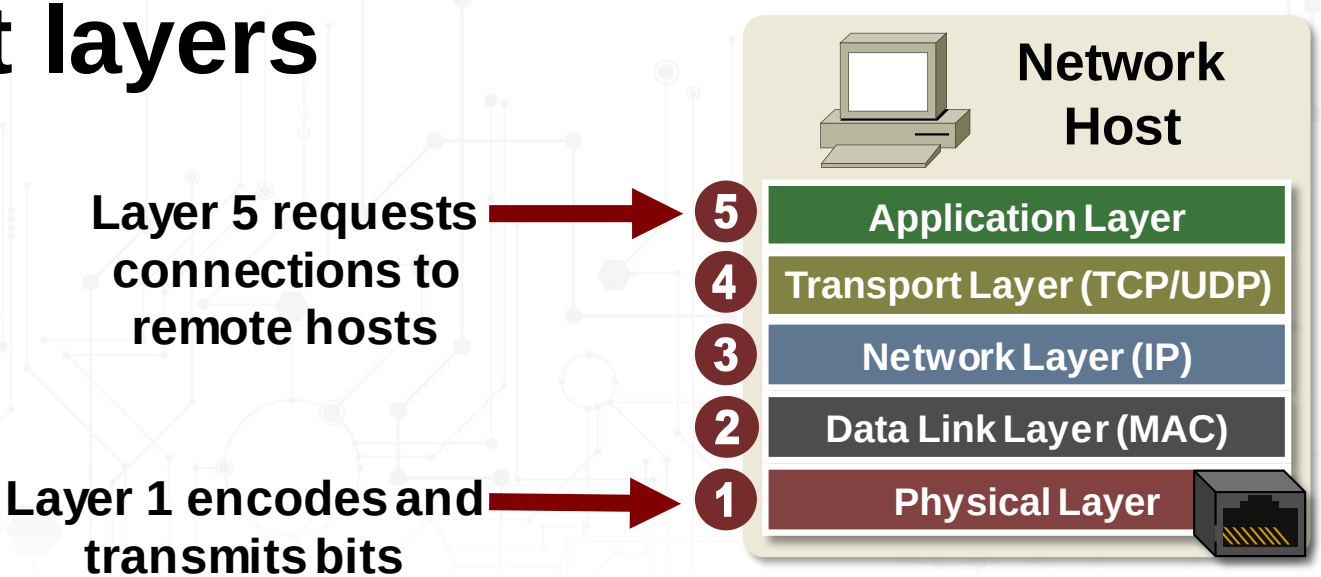
Needs to be sent with all transmitted data.



- **Also need to physically transmit the data**

TCP/IP 5 Layer Model

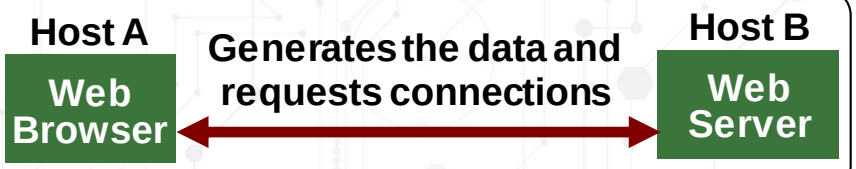
- Each layer performs a specific task needed to move our data across the network
- Each layer only communicates with adjacent layers



What Does Each Layer Do?

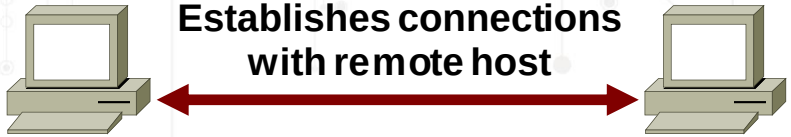
5 Application Layer

The Application layer is the group of applications requiring network communications.



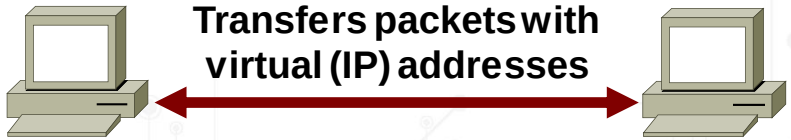
4 Transport Layer (TCP/UDP)

The Transport layer establishes the connection between applications on different hosts.



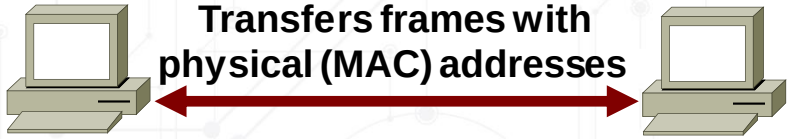
3 Network Layer (IP)

The Network layer is responsible for creating the packets that move across the network.



2 Data Link Layer (MAC)

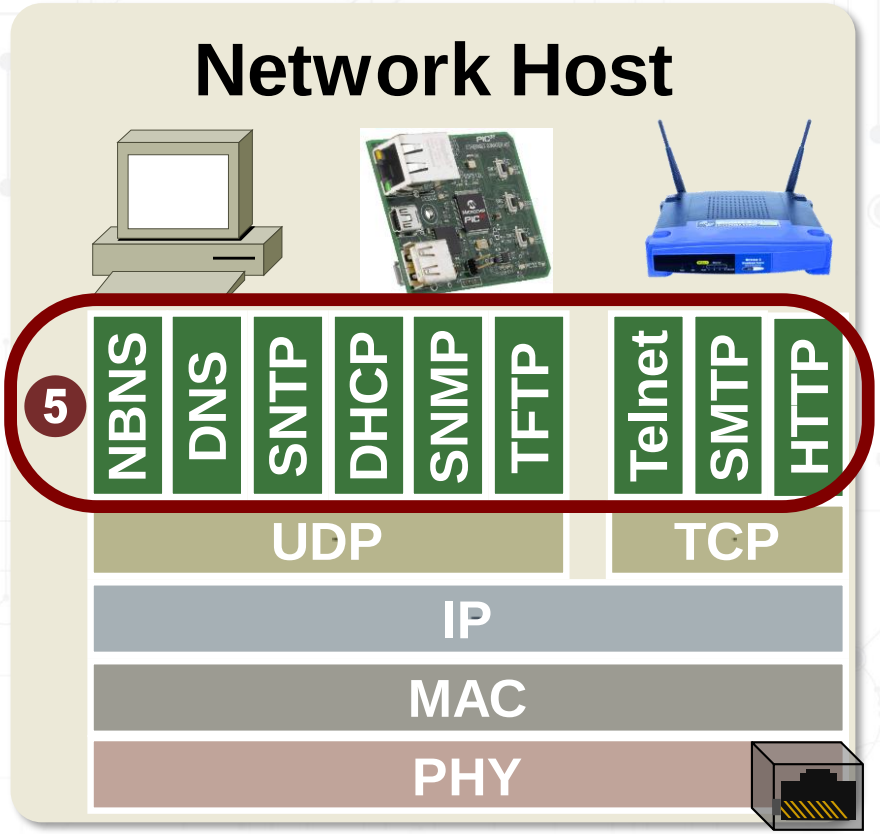
The Data Link layer is responsible for creating the frames that move across the network.



1 Physical Layer

The Physical layer is the transceiver that drives the signals on the network.

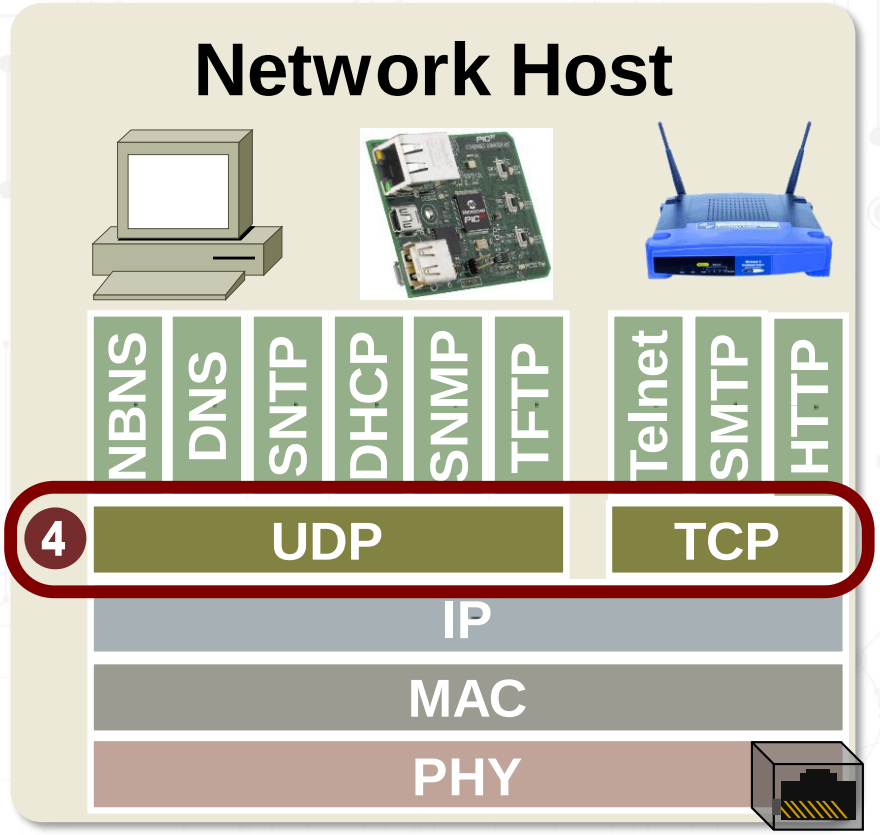
Application Layer
(Layer 5)



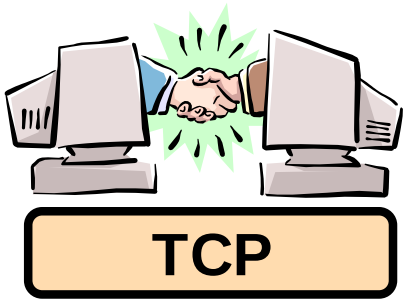
- Layer 5 is where TCP/IP applications live
- Your application typically interacts with these applications

Transport Layer
(Layer 4)

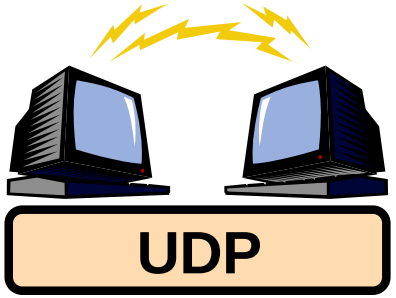
- **Connects to remote hosts using either:**
  - TCP (Transfer Control Protocol)
  - UDP (User Datagram protocol)
- **Delivers data to and from applications**
- **Assigns port numbers to processes running in applications**



TCP vs. UDP



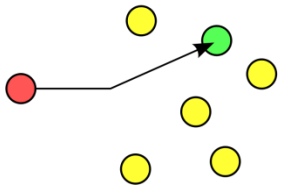
TCP



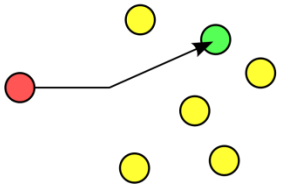
UDP

- **Slower but reliable transfers**
- **Typical applications:**
  - Email
  - Web browsing

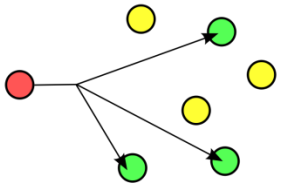
- **Fast but non-guaranteed transfers (“best effort”)**
- **Typical applications:**
  - VoIP
  - Internet Radio



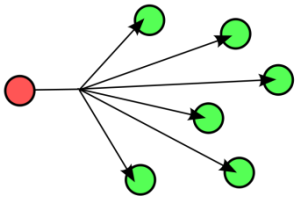
unicast



unicast



multicast



broadcast

TCP and UDP Header Formats

Reference Slide

TCP Segment Header Format

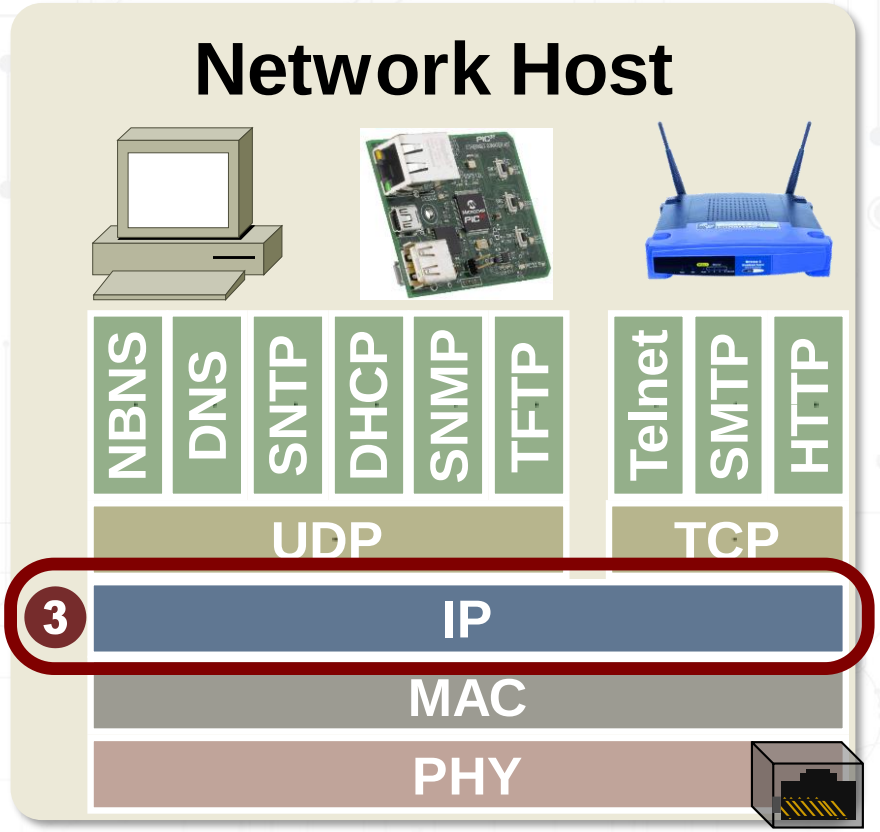
Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Sequence Number			
64	Acknowledgment Number			
96	Data Offset	Res	Flags	Window Size
128	Header and Data Checksum		Urgent Pointer	
160...	Options			

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

Network Layer
(Layer 3)

An IP address is 32 bits and looks like this: 192.168.1.101



- **Also known as the Internet layer**
- **Adds a header to the data received from the transport layer**
  - contains the source and destination IP addresses
- **Creates IP Packets**

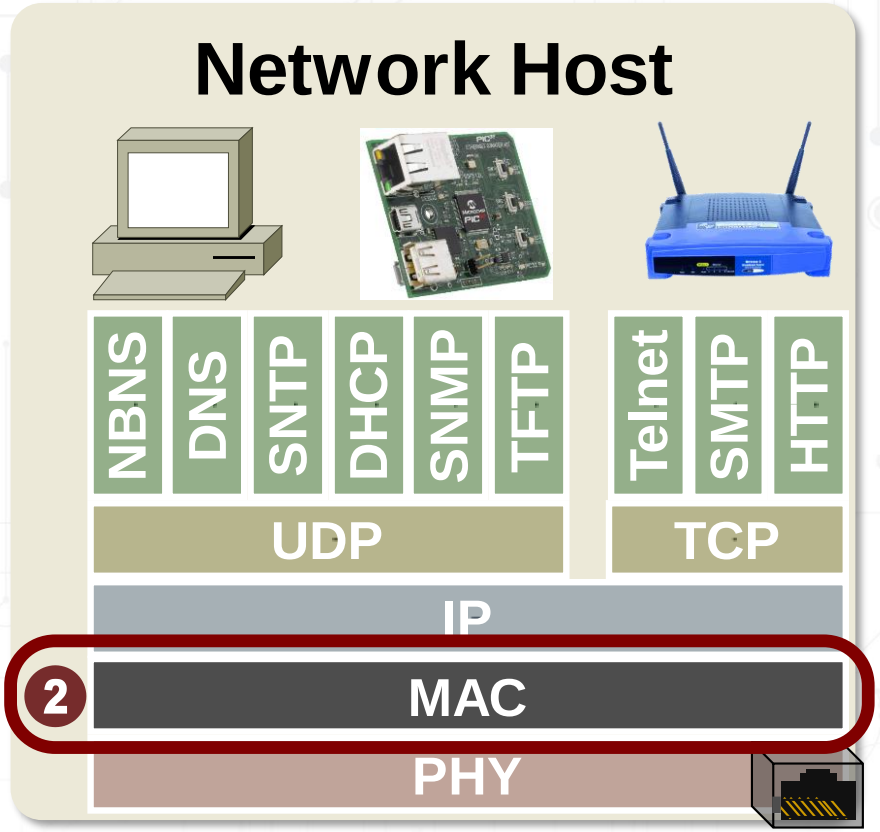
IPv4 Packet Header Format
(Reference Slide)

IPv4 Packet Header Format

Bit #	0 – 7		8 – 15		16 – 23	24 – 31
0	Version	IHL	DSCP	ECN	Total Length	
32	Identification				Flags	Fragment Offset
64	Time to Live		Protocol		Header Checksum	
96	Source IP Address					
128	Destination IP Address					
160	Options (if IHL > 5)					

Data Link Layer
(Layer 2)

A MAC address is 48 bits and looks like this: F0:DE:F1:1E:E8:93



- **Uses a Media Access Controller to generate frames**
- **Adds a header to the packet**
  - Source and destination MAC addresses
- **Every host has at least one MAC address**

Ethernet and Wi-Fi® Frame Formats
(Reference Slide)

Ethernet (802.3) Frame Format

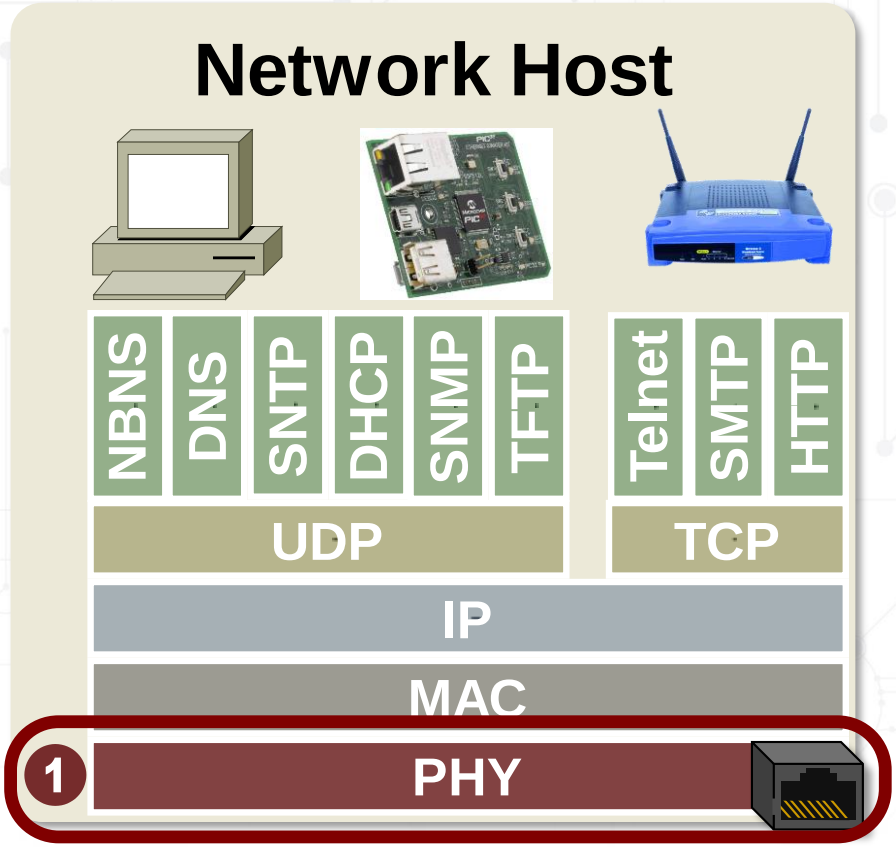
7 bytes	1 byte	6 bytes	6 bytes	2 bytes	42 to 1500 bytes	4 bytes	12 bytes
Preamble	Start of Frame Delimiter	**Destination MAC Address**	**Source MAC Address**	Type	Data (payload)	CRC	Inter-frame gap

For TCP/IP communications, the payload for a frame is a packet

Wi-Fi (802.11) Frame Format

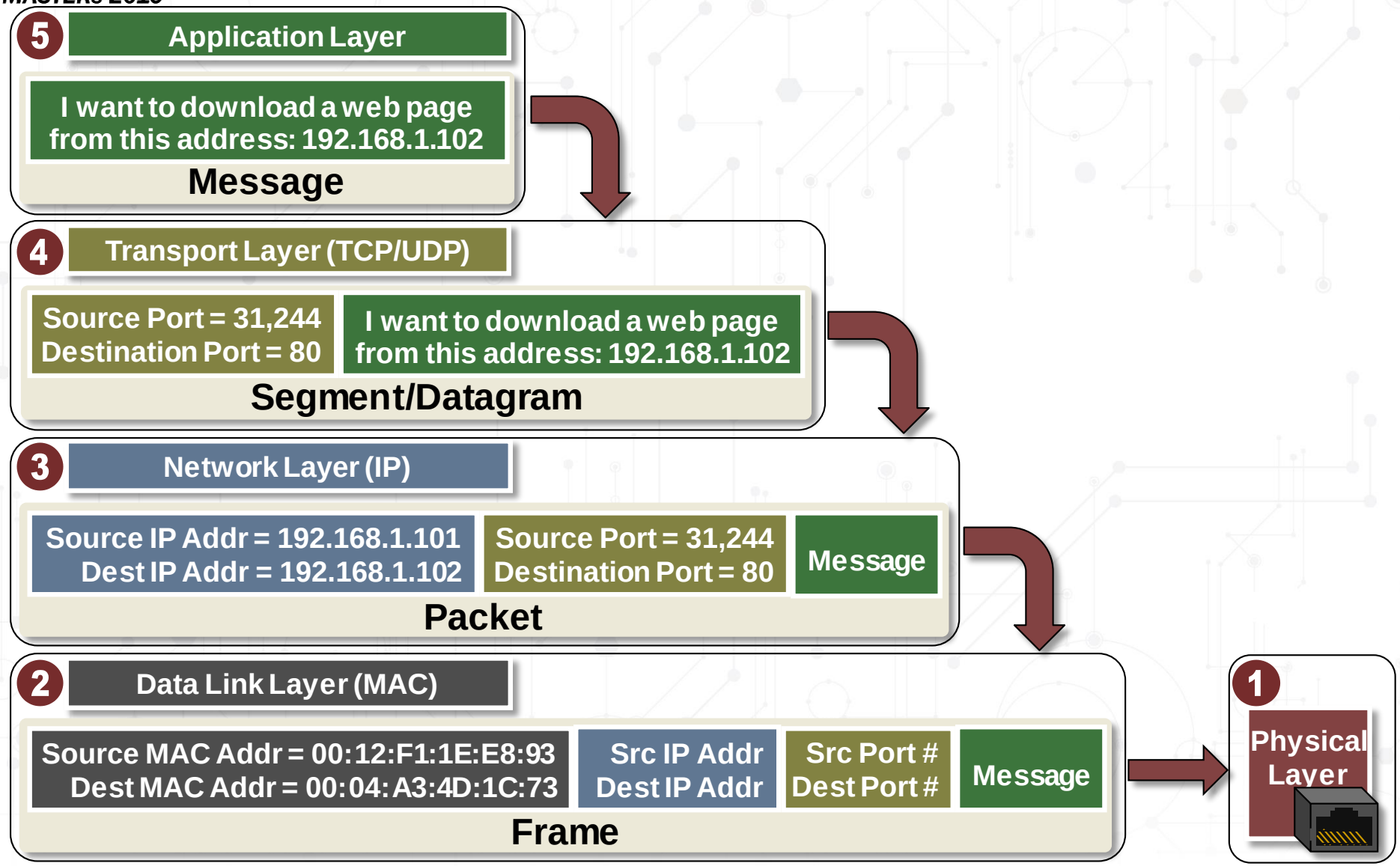
2 bytes	2 bytes	6 bytes	6 bytes	6 bytes	2 bytes	6 bytes	0 to 2312 bytes	4 bytes
Frame Control	Duration	MAC Address 1 (Destination)	MAC Address 2 (Source)	MAC Address 3 (Router)	Seq Control	MAC Address 4 (AP)	Data (payload)	CRC

Physical Layer
(Layer 1)

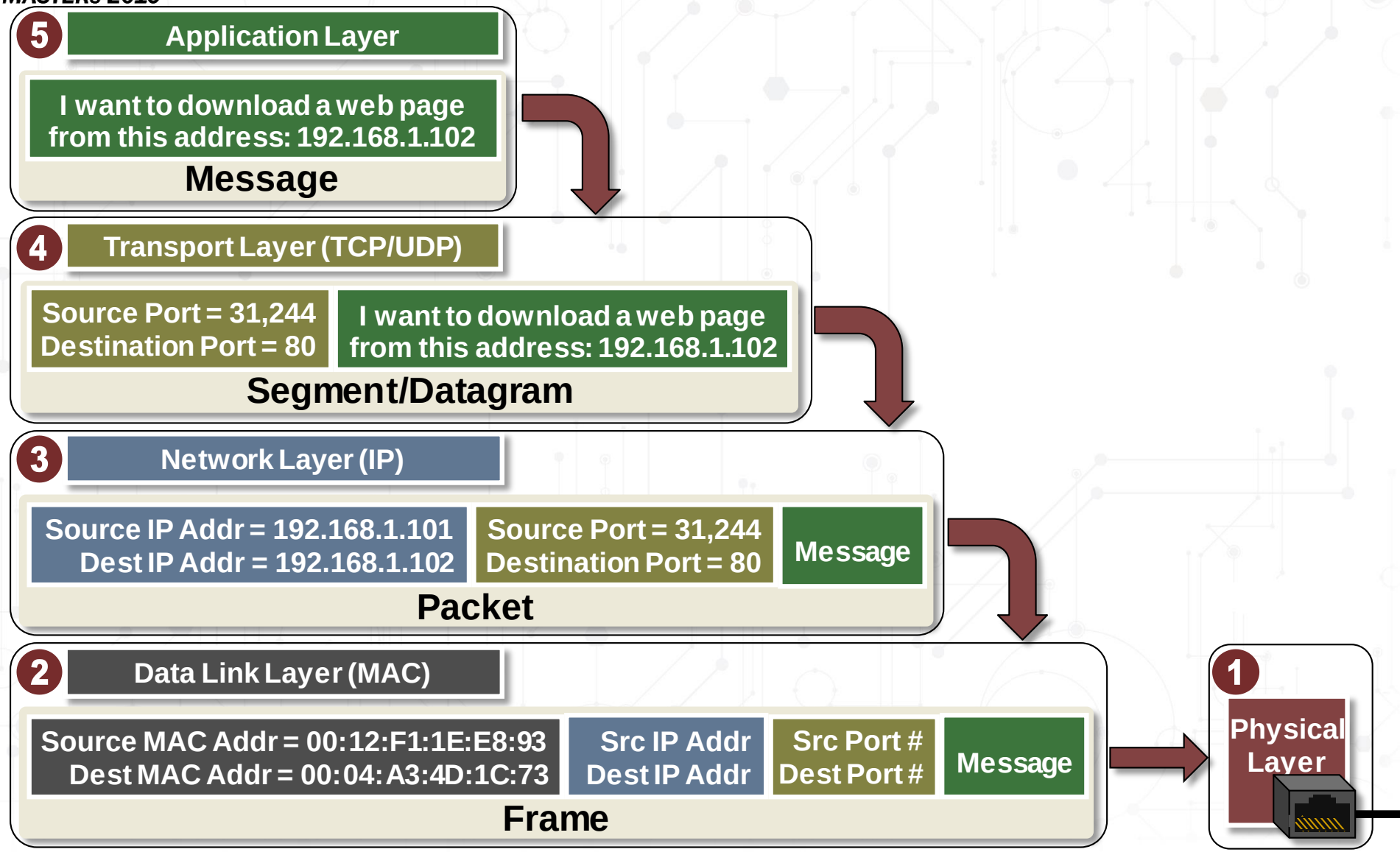


- Sends and receives signals on the physical wire or antenna
- Responsible for moving bits
- Found at the end of every network interface

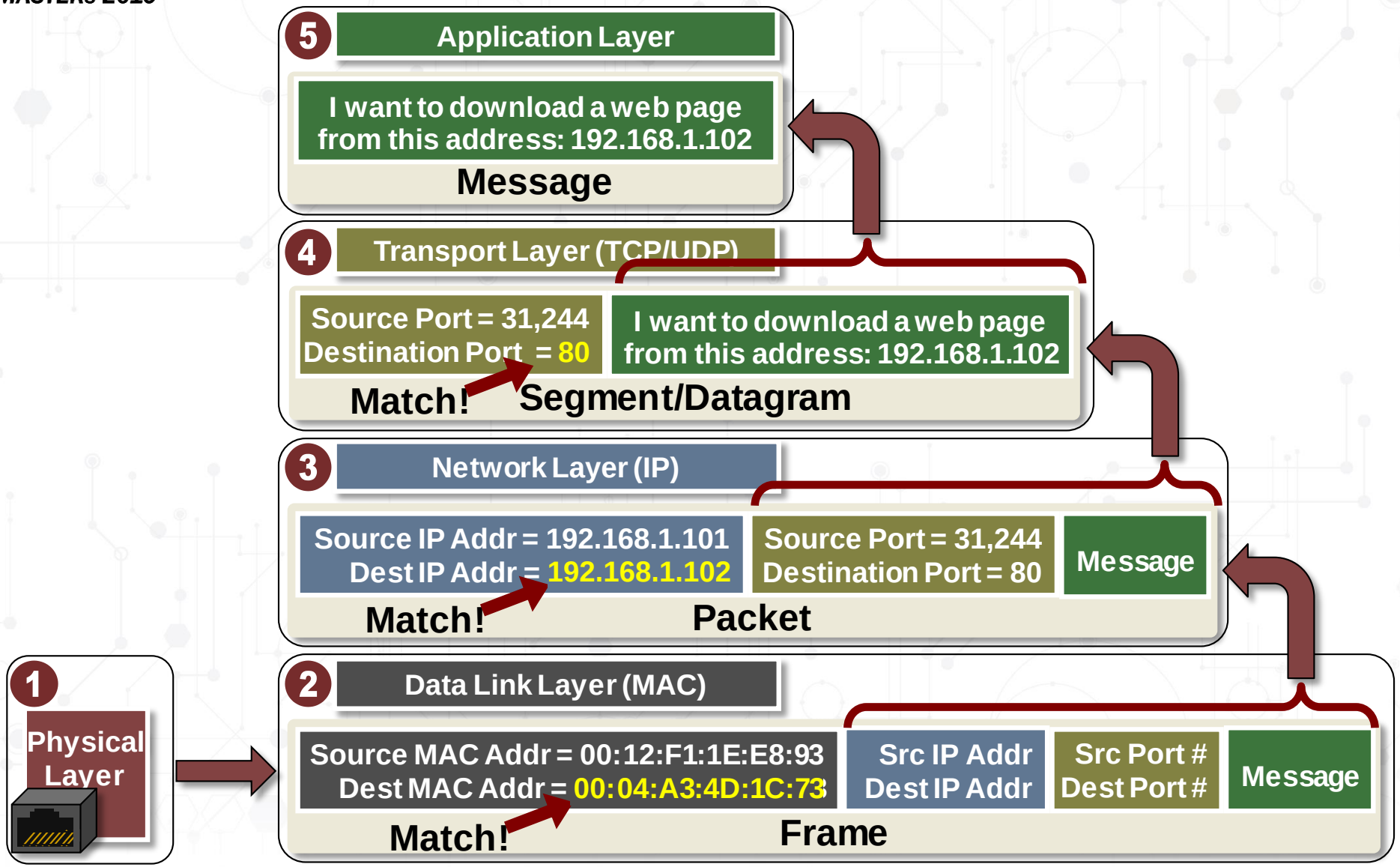
Transmit Data Using Network Layers



Receive Data Using Network Layers



Receive Data Using Network Layers



TCP/IP Protocol Stack
(Terminology Reference)

Layer #	Layer Name	Protocol	Protocol Data Unit	Addressing
5	Application	HTTP, SMTP, etc…	Messages	n/a
4	Transport	TCP/UDP	Segments/ Datagrams	Port #s
3	Network or Internet	IP	Packets	IP Address
2	Data Link	Ethernet, Wi-Fi	Frames	MAC Address
1	Physical	10 Base T, 802.11	Bits	n/a

Common TCP/IP Applications

Application	Description
DHCP	Dynamic Host Configuration Protocol assigns IP addresses
DNS	Domain Name System translates website names to IP addresses
HTTP	Hypertext Transfer Protocol used to transfer web pages
NBNS	NetBIOS Name Service translates local host names to IP addresses
SMTP	Simple Mail Transfer Protocol sends email messages
SNMP	Simple Network Management Protocol manages network devices
SNTP	Simple Network Time Protocol provides time of day
Telnet	Bi-directional text communication via a terminal application
TFTP	Trivial File Transfer Protocol used to transfer small amounts of data

Knowledge Check

?

When the Transport layer receives a message from a remote host, how does it know which application process to deliver it to?

Port Numbers

Knowledge Check

?

True or false:
The Simple Mail Transport Protocol (SMTP) application uses the UDP transport protocol.

False

SMTP uses TCP to guarantee error free delivery.

MICROCHIP MASTERs 2019

Part 2: MPLAB® Harmony TCP/IP Stack Overview

- TCP/IP Layers and Features
- Network Interface Options
- Processor Requirements

SMART | CONNECTED | SECURE

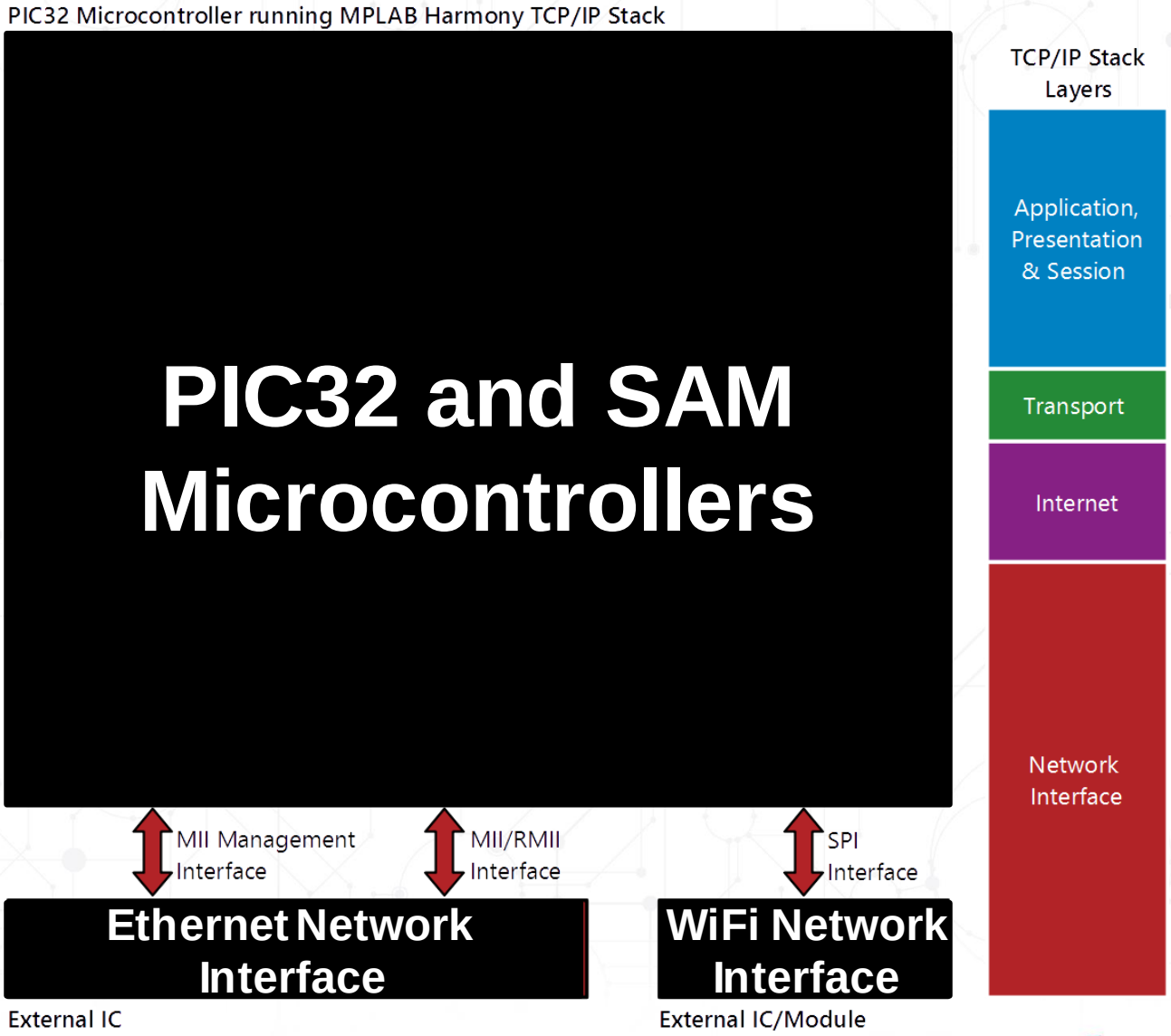
MICROCHIP

MPLAB® Harmony TCP/IP Stack

- Provides a foundation for embedded network applications by handling most of the interaction required between the physical network interface and your application.
- Fully Implemented and supported by Microchip Engineers.
- Source code is included with the MPLAB® Harmony Distribution.
- Cost: Free*

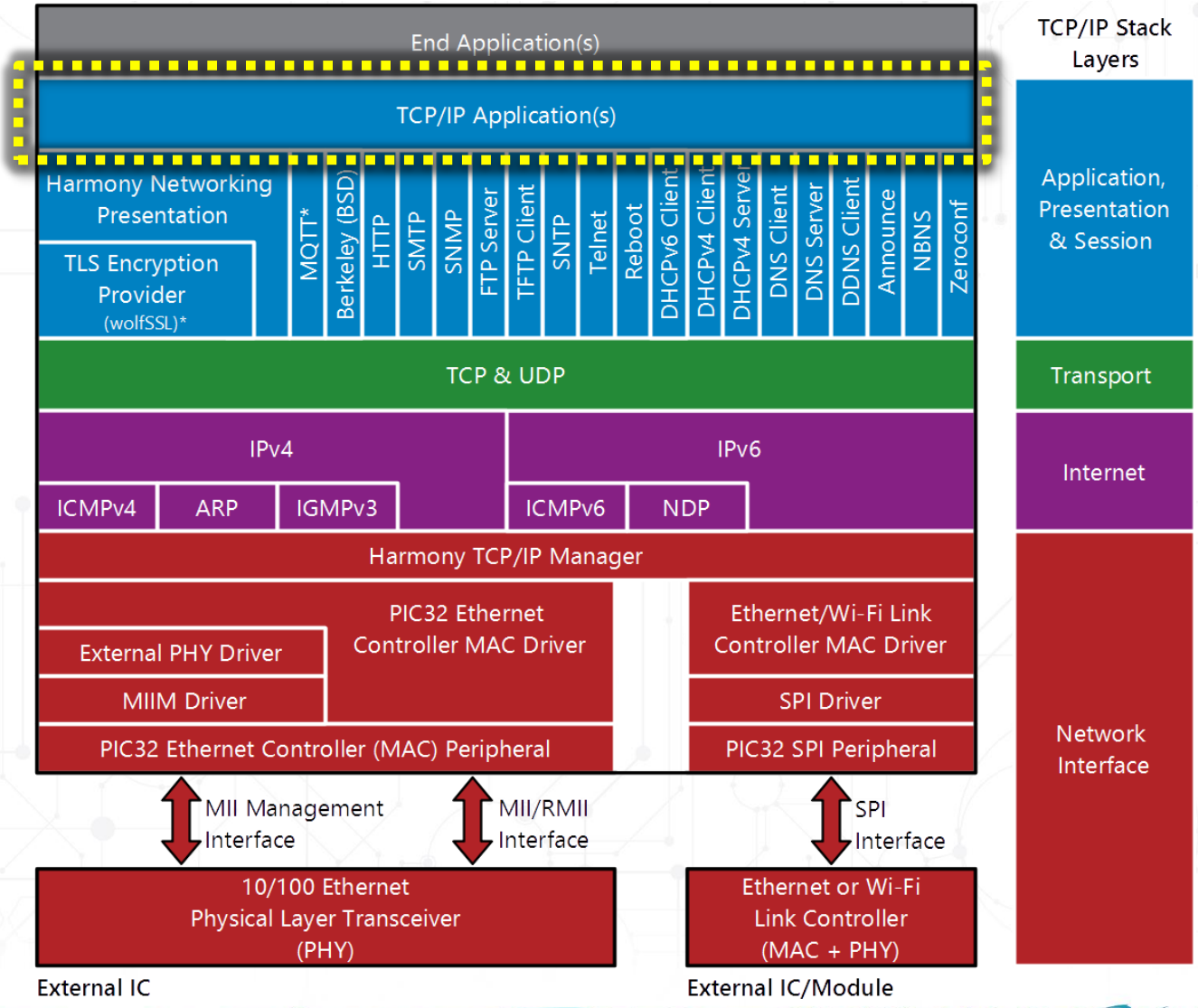
*MPLAB Harmony is only licensed for use on Microchip Microcontrollers

Architecture



Application Layer

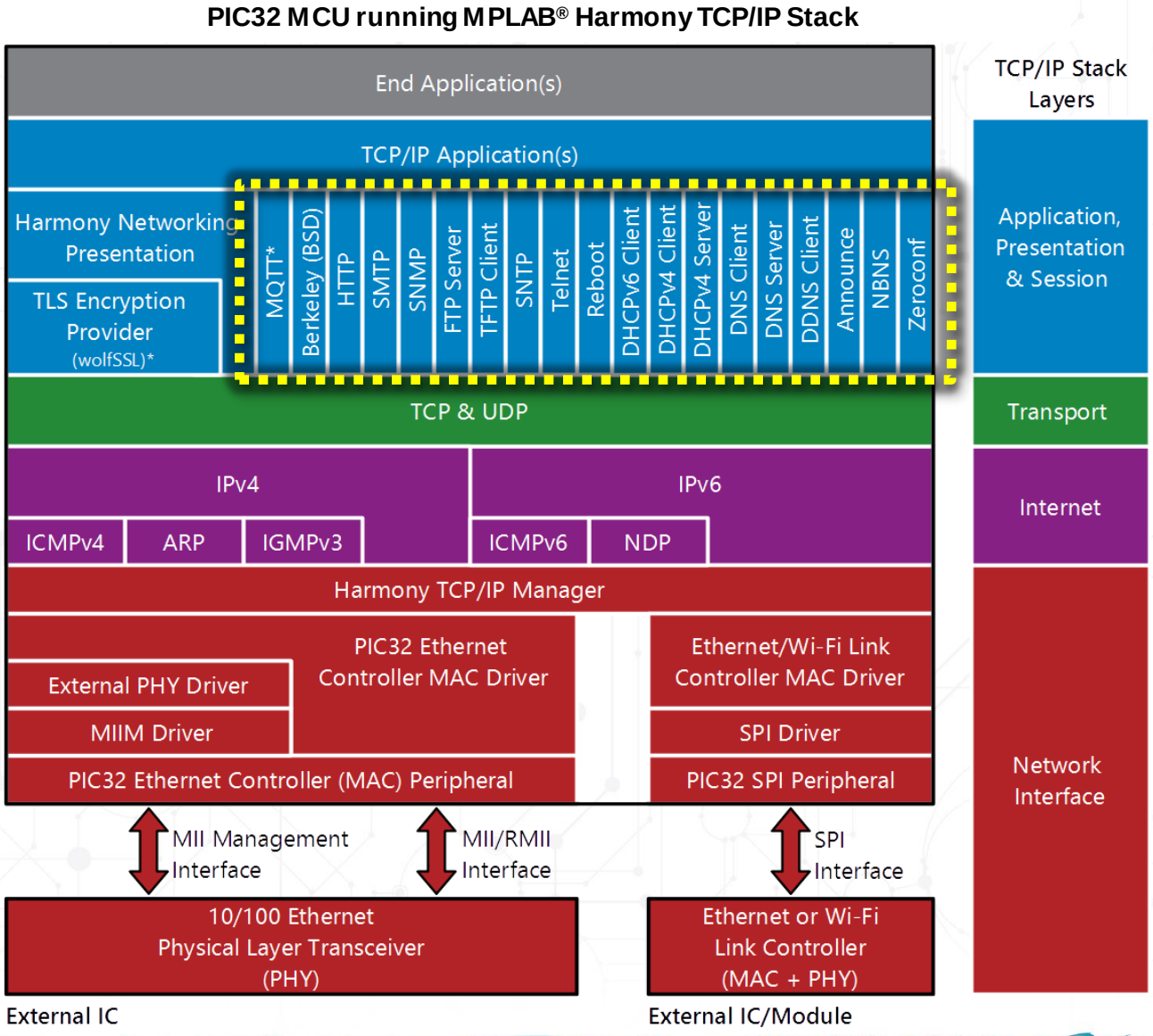
PIC32 MCU running MPLAB® Harmony TCP/IP Stack



Application Layer

- Networking Applications can consist of:
  - User implemented TCP/UDP Socket Applications.
  - Prewritten MPLAB® Harmony TCP/IP Application Modules (e.g. HTTP) .
- Multiple Networking Applications can run simultaneously.
- TCP/IP Stack can coincide with other MPLAB® Harmony Middleware (e.g. Graphics).

Services and Applications



Supported Services

- **DHCP:** *Dynamic Host Configuration Protocol*
  - Used for IP Address Assignment
  - Server and Client Support
- **DNS:** *Domain Name System*
  - Translates website name to IP Address
  - Server and Client Support
- **DDNS Client**
  - Discovery and registration of public IP address with a DDNS Provider on the internet.

Supported Services

- **Announce**
  - Microchip TCP/IP Discovery Protocol used for identifying devices on a local network.
- **NBNS:** *NetBIOS Name Service*
  - Translates local host names to IP Addresses
- **Zeroconf** (Bonjour/Avahi)
  - A set of technologies to automatically create a usable network including IP Address assignment, distribution and resolution of hostnames, location of network services.

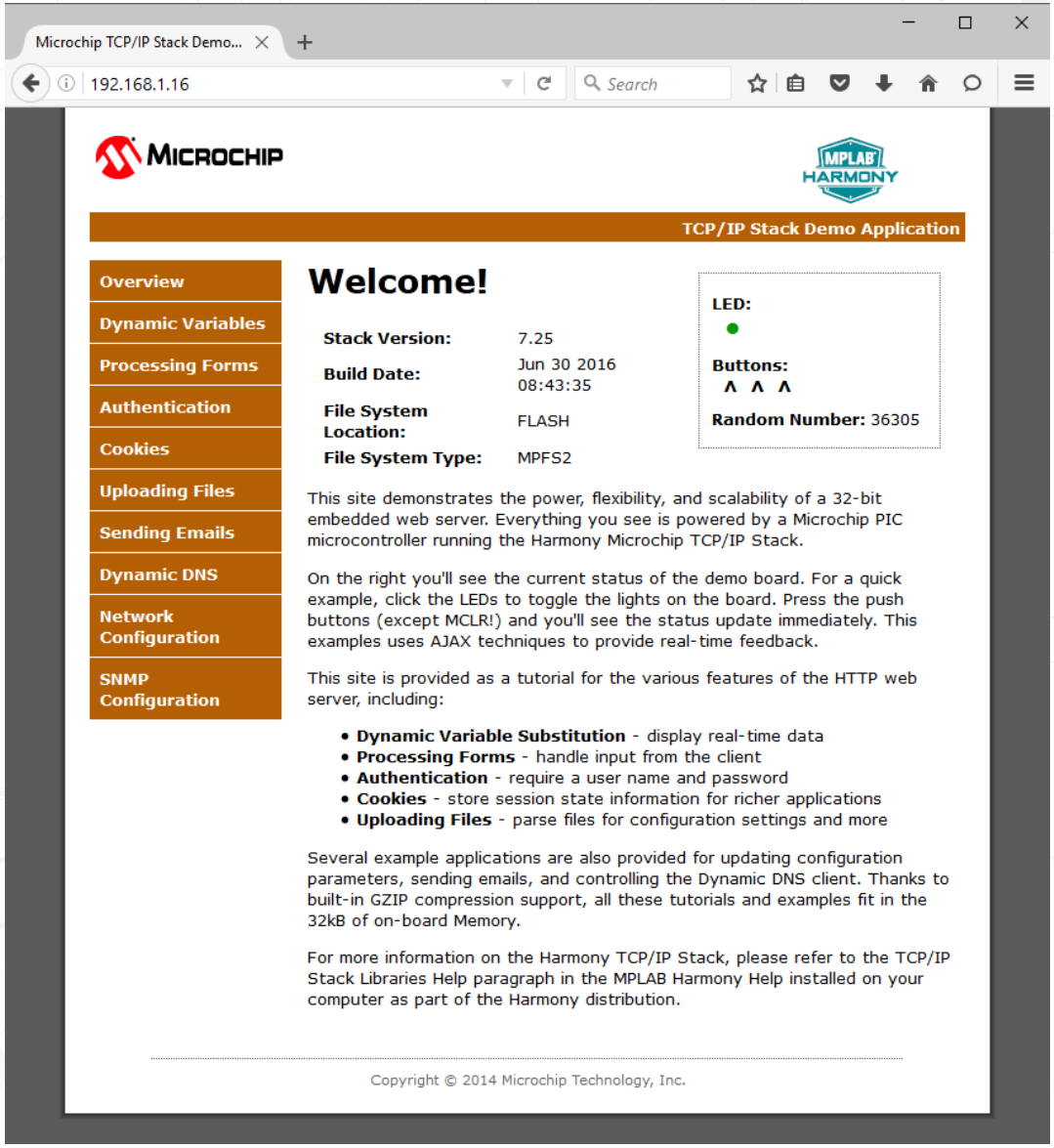
Supported Services

- **MQTT:** *Message Queue Telemetry Transport*
  - Machine-to-machine (M2M) or "Internet of Things" connectivity protocol
  - Lightweight publish/subscribe messaging transport
  - Supported with third party ***wolfMQTT*** which is bundled with MPLAB® Harmony.
    - https://www.wolfssl.com/wolfSSL/Products-wolfmqtt.html

Supported Applications

- **HTTP:** *Hypertext Transfer Protocol (Web Server)*
  - Allows for serving of Webpages
  - Webpages can be stored in Program Memory, SD Card, or External Flash
  - Supports Forms and Dynamic Variables

HTTP Web Server

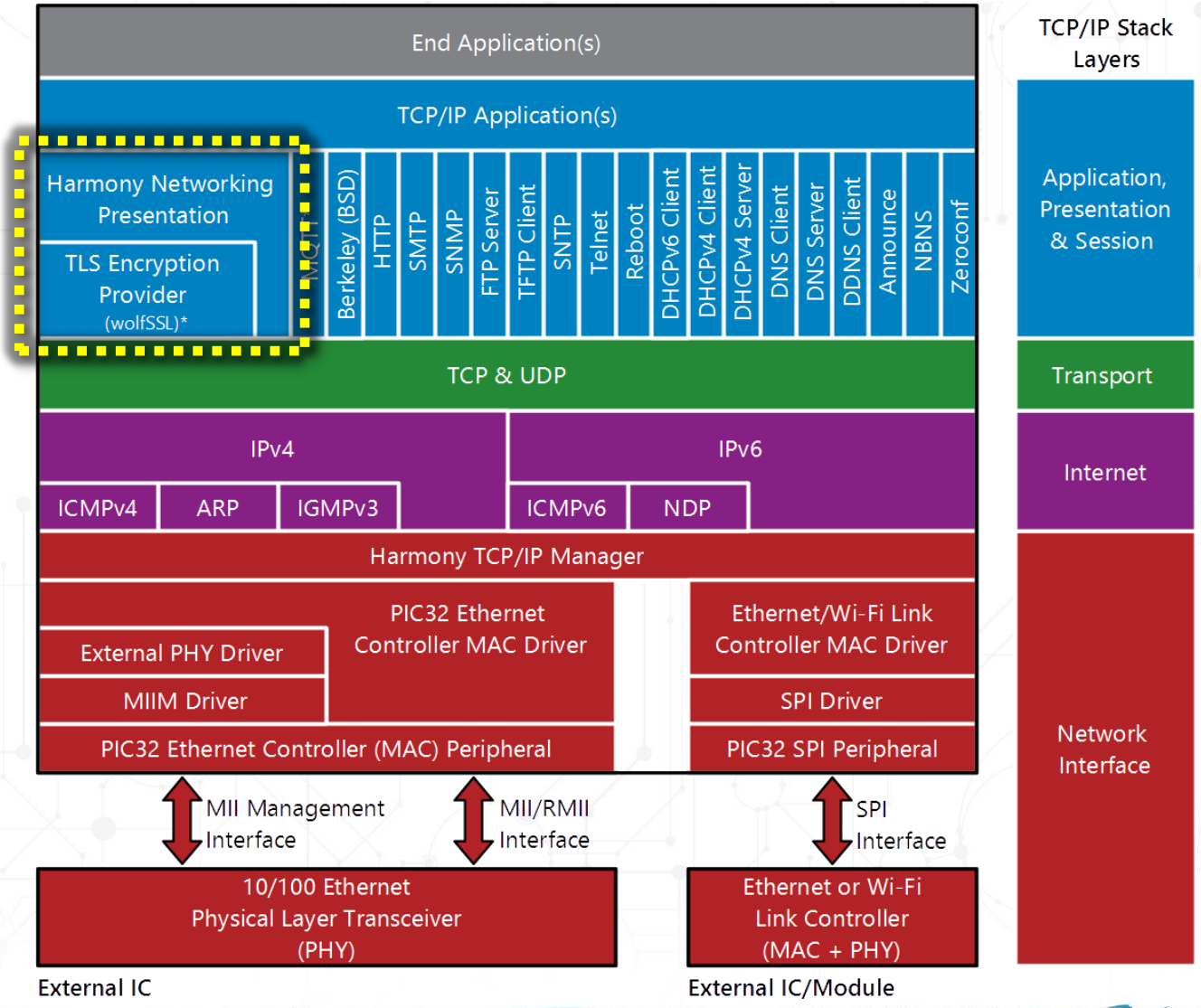


Supported Applications

- **SMTP** *Simple Mail Transfer Protocol*
  - Allows E-Mails to be sent.
  - Client Support
- **SNMP** *Simple Network Management Protocol*
  - Allows for management of devices on a network.
- **FTP** *File Transfer Protocol*
  - Allows for transfer of files.
- **NTP** *Network Time Protocol*
  - Allows the time of day to be obtained from a precise timer server on the internet.

Encryption Provider

PIC32 MCU running MPLAB® Harmony TCP/IP Stack



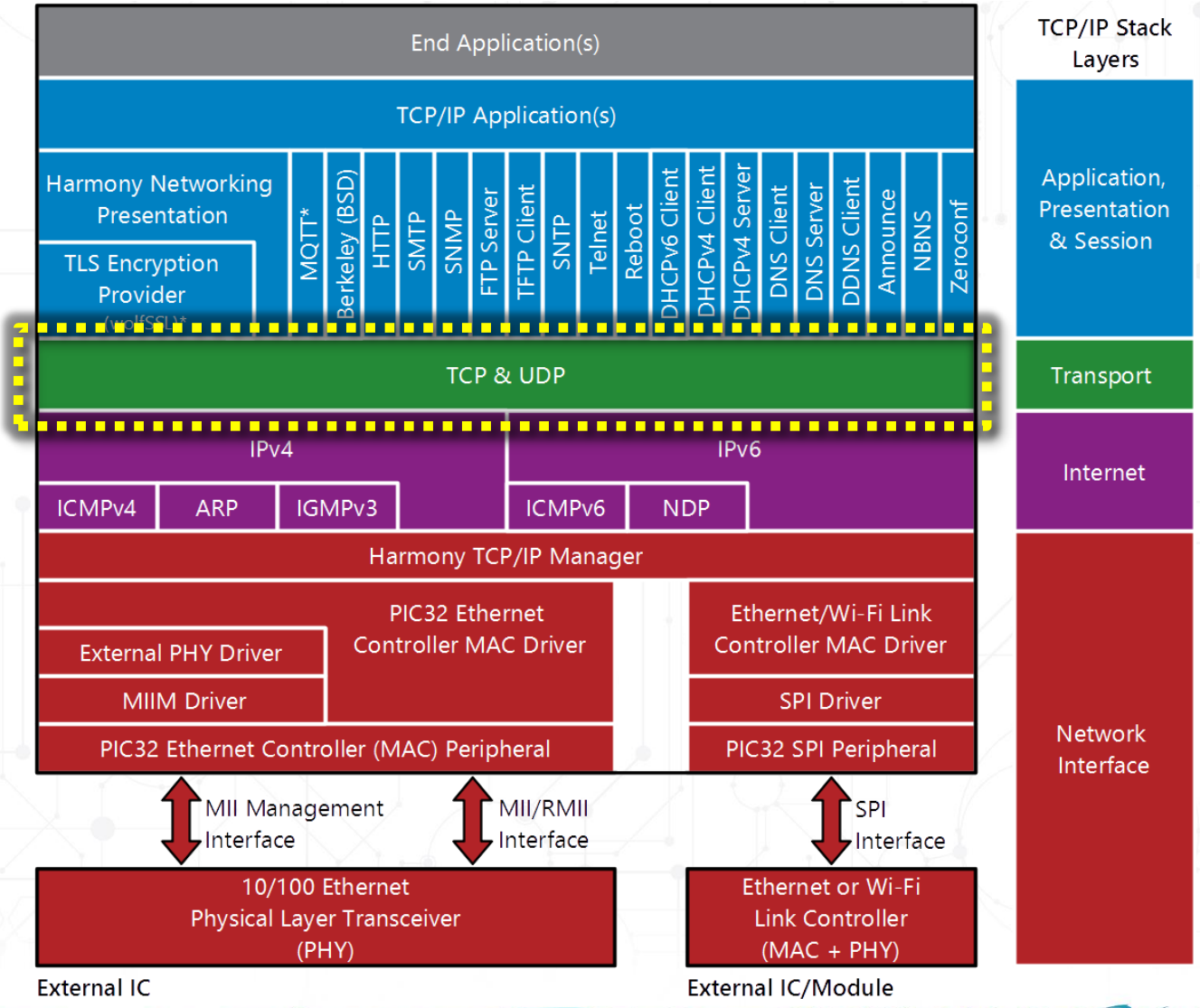
Encryption Provider

- **TLS from wolfSSL (3rd party provider)**
  - Written in ANSI C
  - Works with or without RTOS
  - TLS 1.2 and DTLS 1.2
  - Integrates directly into MPLAB® Harmony TCP/IP Stack
  - Distributed with MPLAB Harmony
  - Can be evaluated free of charge
  - Requires a commercial license purchase for production

SSL= Secure Sockets Layer, TLS = Transport Layer

Transport Layer

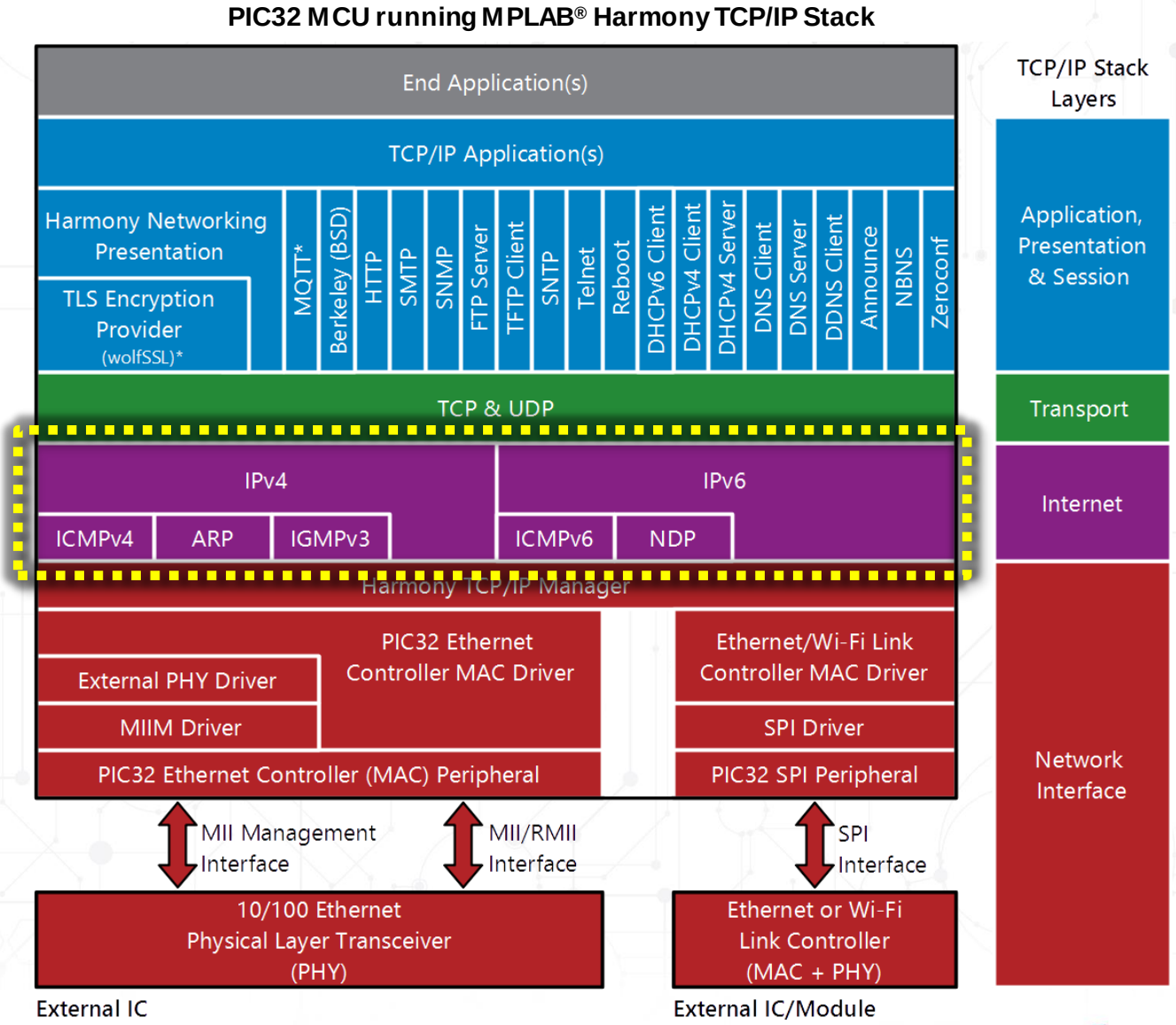
PIC32 MCU running MPLAB® Harmony TCP/IP Stack



Transport Layer

- Light-weight and high-performance implementations of the TCP and UDP transport layers.
- Various TCP and UDP configuration options are available at both project generation and run time (eg TX/RX buffer sizes).
- Simultaneous sockets are supported (max number is configurable).
- Simple APIs are used to create socket, check connection status, check TX/RX buffer status, read/write data, and close the socket.

Internet Layer



Internet Layer

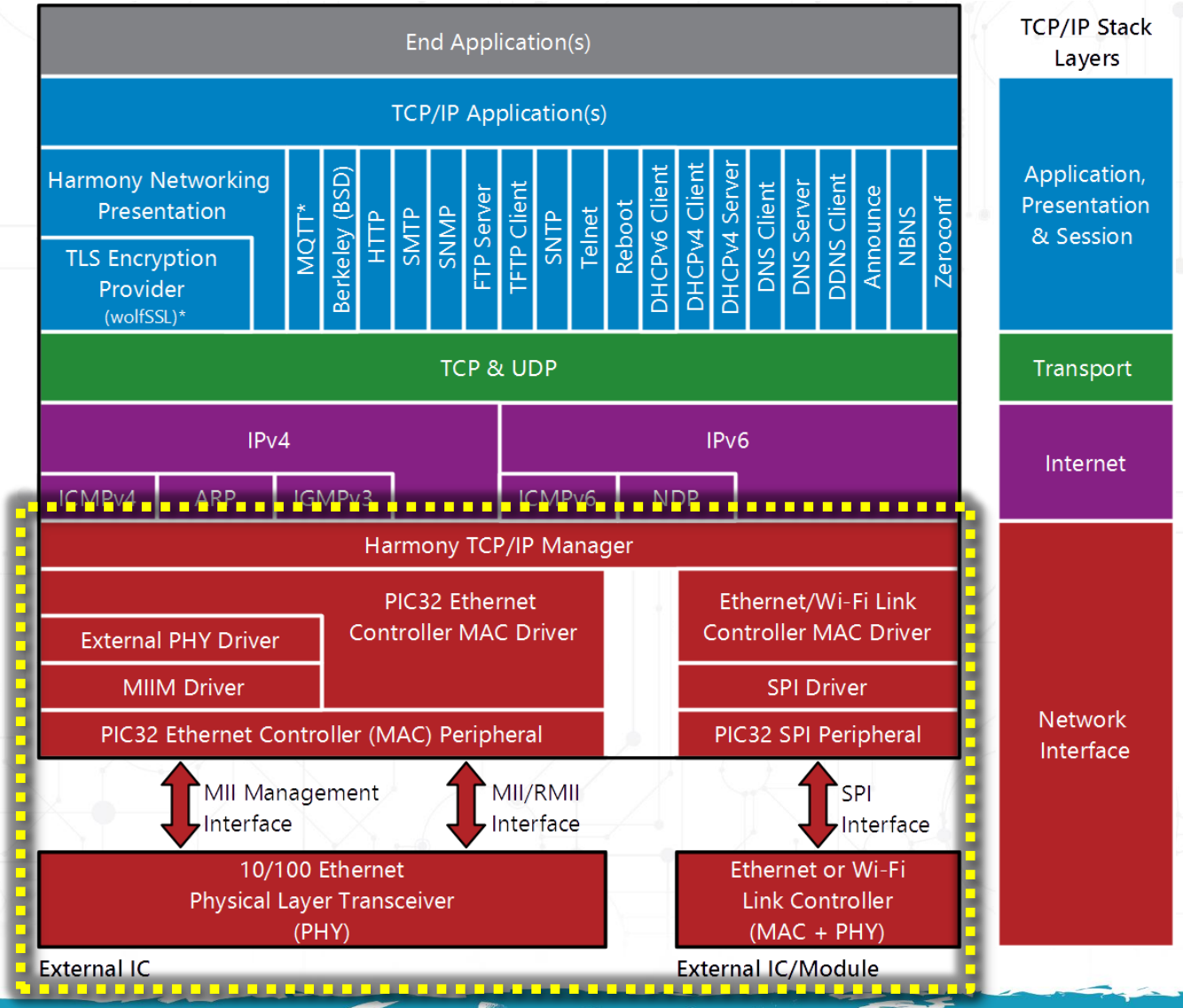
- IPv4 and IPv6 Support
- Both IPv4 and IPv6 can run simultaneously
- The IP Layer is transparent to all applications that use the built in Transport Layer functions.
- IP Datagrams can be sent directly using a special API bypassing higher layer protocols.

IPv4 vs IPv6

- **IPv4** 192.168.1.1
  - Four 8-bit segments
  - Decimal numbers 0 to 255
  - Number of addresses: 32 bits = 4.3 billion
- **IPv6** FE80:0000:0000:0000:75EA:CEFF:FE44:5101
  - Eight 16-bit segments
  - Hex numbers 0 to FFFF
  - Number of addresses: 128 bits = 3.4×10^{38}
    =340,282,366,920,938,463,463,374,607,431,768,211,456

Network Interface Layer

PIC32 MCU running MPLAB® Harmony TCP/IP Stack

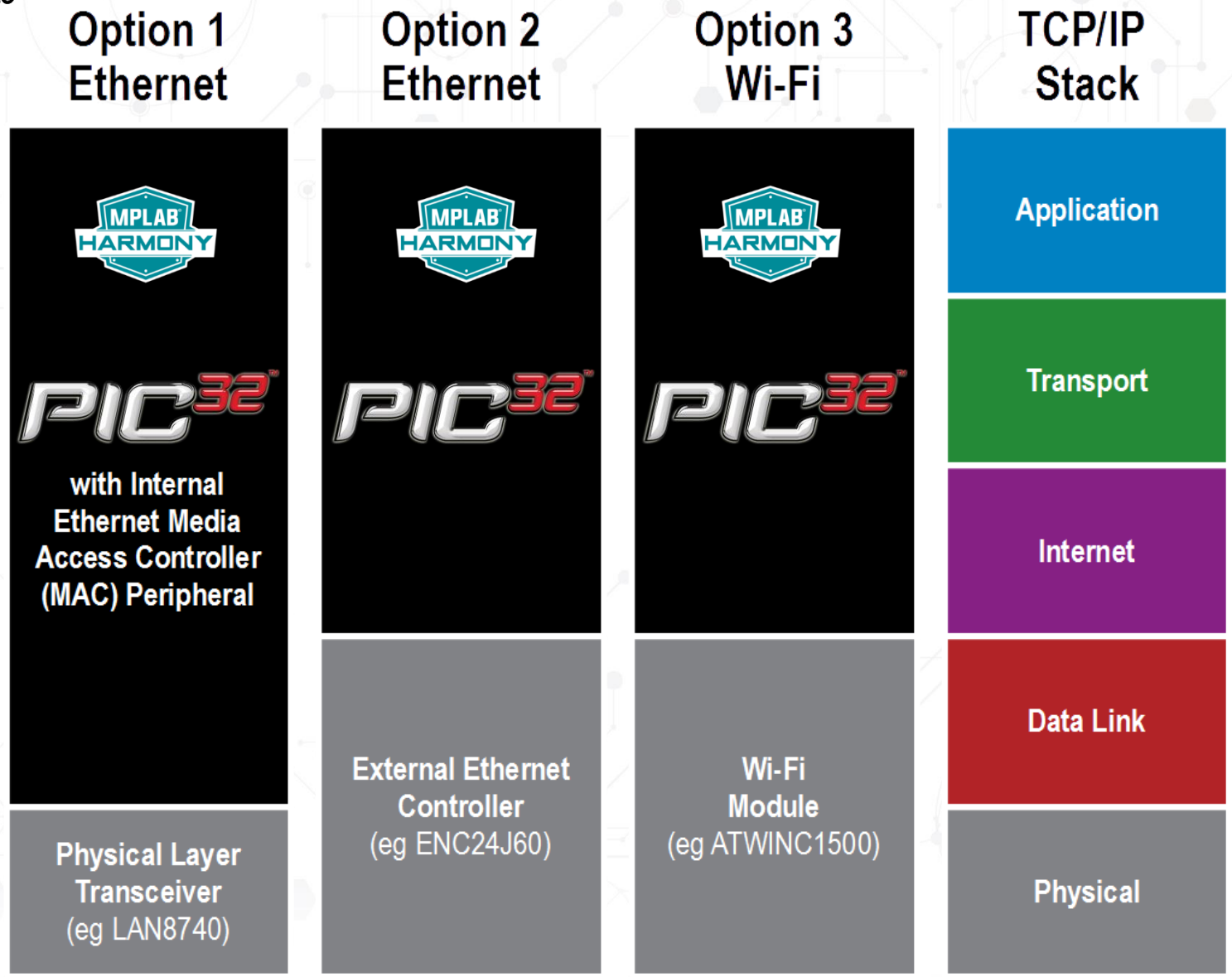


Network Interface Features

- Multiple Concurrent Interfaces on one MCU e.g.
  - Ethernet + Wi-Fi®*
  - Ethernet + Ethernet
- Drivers for supported interface devices are supplied in the TCP/IP stack.
- Interface options are selected and configured using MPLAB® Harmony Configuration (MHC) Tool.
- The TCP/IP Stack automatically initialises and manages the configured network interface(s).

***In Harmony 2.06, the Wi-Fi drivers can only support a single Wi-Fi interface.**

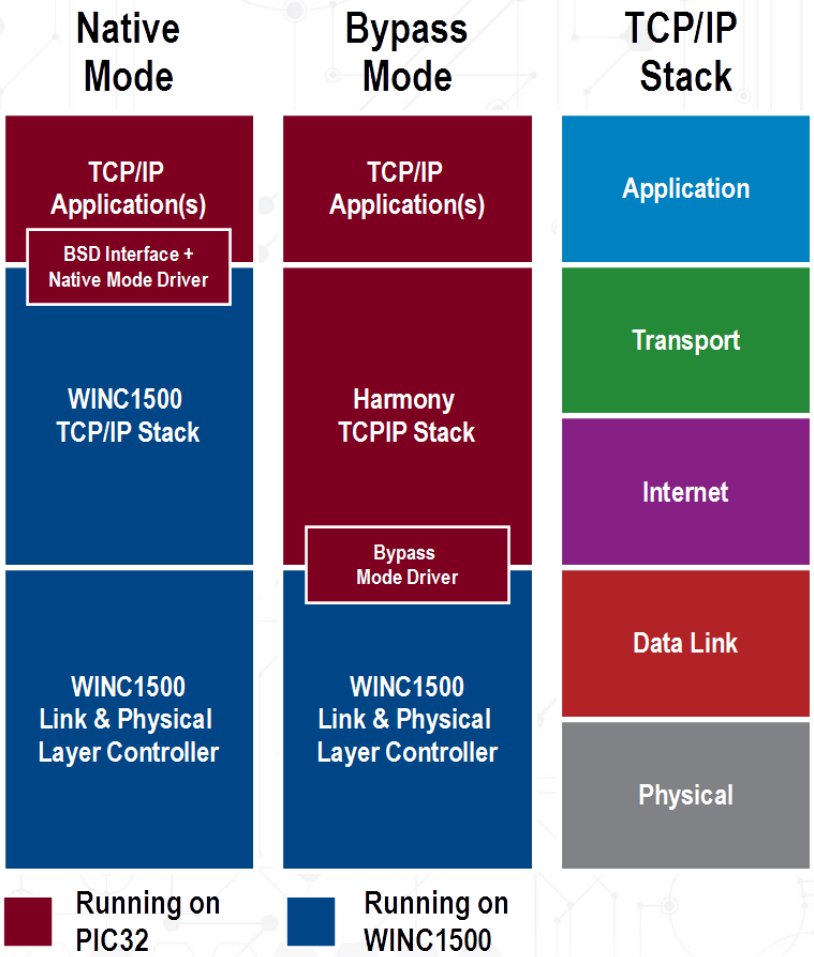
Network Interface Options



Supported Interface Devices

- **Ethernet PHYs**
  - **Only used with PIC32 devices that have internal Ethernet MAC Peripheral**
  - **PHY Only**: LAN8700, LAN8720, LAN8740, KSZ8041 & KSZ8061
  - **PHY + Switch**: LAN9303 & KSZ8863
- **Standalone Ethernet Link Controllers**
  - ENC28J60 (SPI)
  - ENCx24J600 (SPI)
- **Wi-Fi®**
  - ATWINC1500 (SPI)
  - ATWILC1000 (SPI)

WINC1500 Wi-Fi® Module
Operation Modes with Harmony



Two modes of operation:

- **Native Mode**: TCP/IP Stack residing on WINC1500 device is utilised
- **Bypass Mode**: Harmony TCP/IP Stack is utilised

Harmony TCP/IP Stack features are only available when WINC1500 is operated in Bypass mode.

MAC Address

- All network interface devices must have a globally unique MAC address.
- Some devices are already supplied with a MAC Address, including:
  - PIC32 devices with Internal Ethernet MAC Peripheral
  - All Microchip Wi-Fi® modules
  - Microchip ENCx24J600 Ethernet Controller

PIC32 MAC Address

- A globally unique MAC Address is programmed into a read only section of Flash Program Memory.
- This feature is only available on PIC32 Devices with an Internal Ethernet MAC Peripheral.
- The address is automatically loaded into the EMAC1Sx registers after reset.
- The address can be manually read using a Harmony API: **`PLIB_ETH_MACGetAddress`**.

TCP/IP Stack Features

- **Fully dynamic:**
  - Stack initialization/de-initialization
  - Interface up/down
  - Resource management
  - Module configuration
- **Run-time configuration via APIs or Console**
- **Interrupt driven operation**
- **RTOS compatible**

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
  - Socket Debugging

MICROCHIP MASTERs 2019

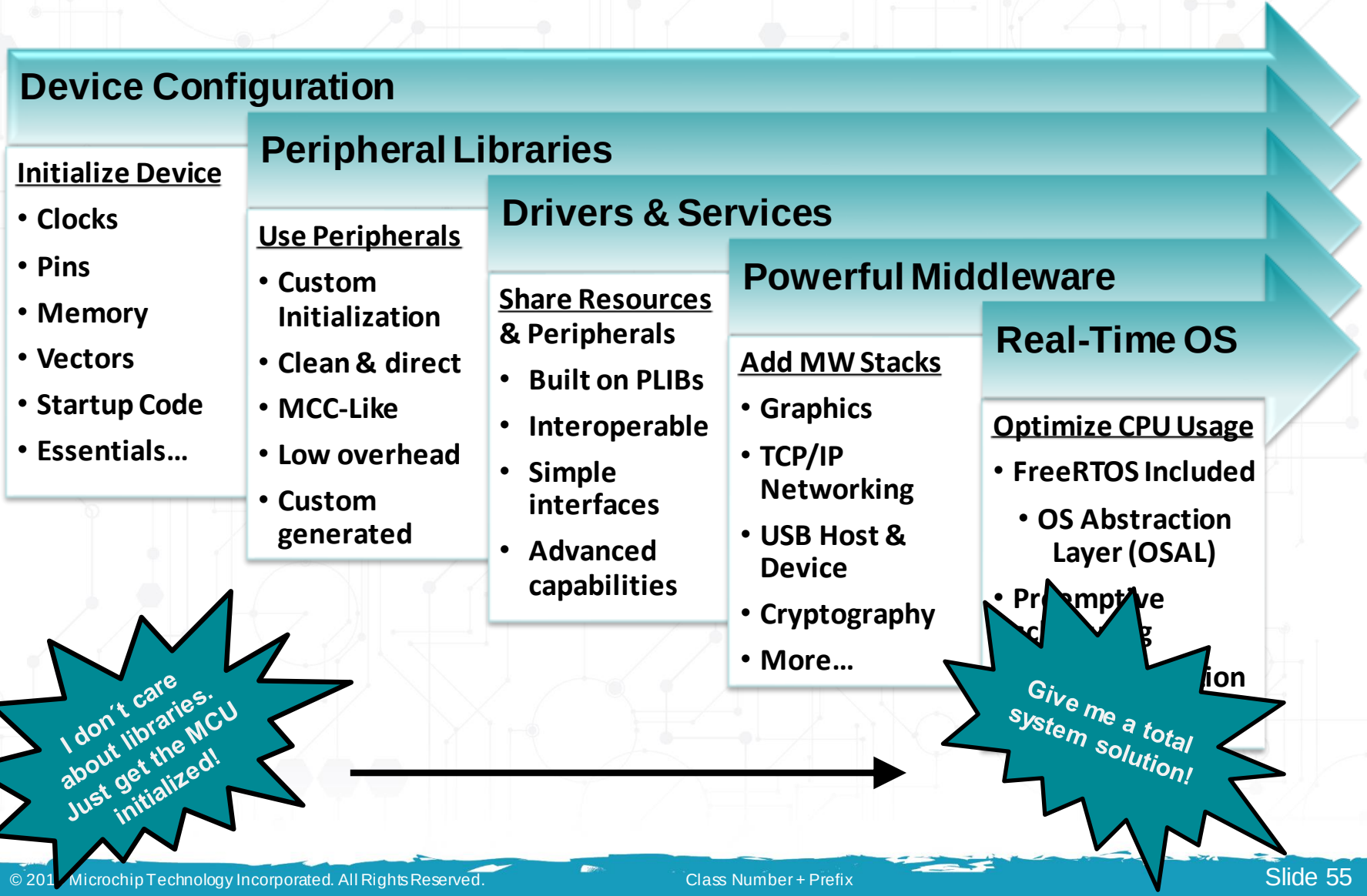
Part 3: Using the MPLAB® Harmony TCP/IP Stack

- Open an example project
- Stack (Re)Configuration
- Connectivity Check
- MPLAB® Harmony TCP/IP Stack APIs
- Tools

SMART | CONNECTED | SECURE

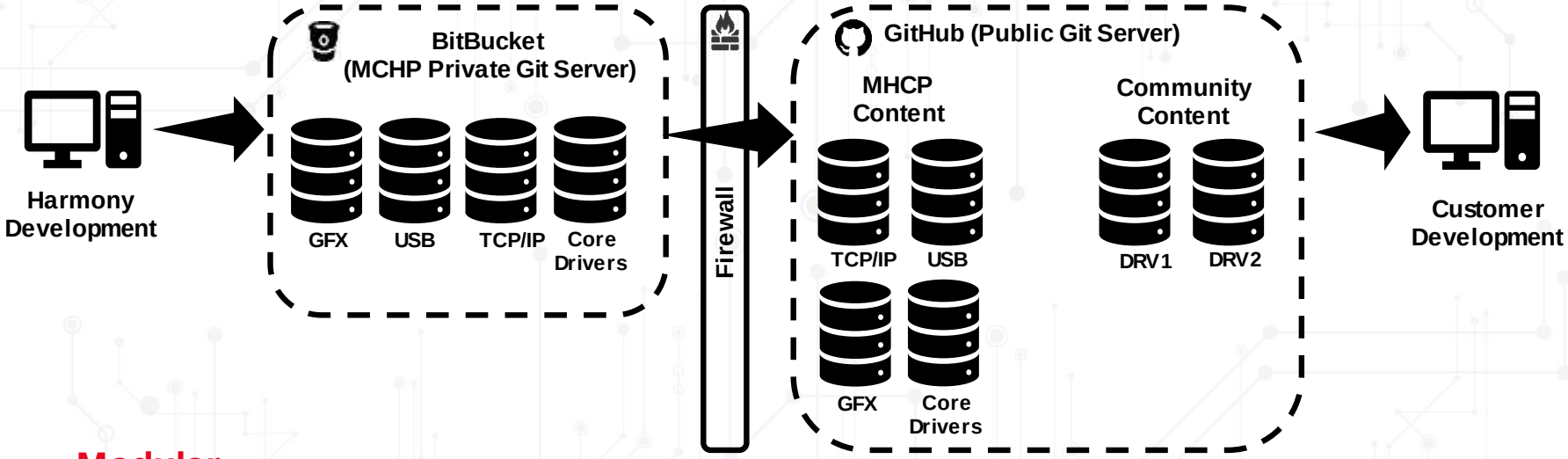
MICROCHIP

Harmony V3



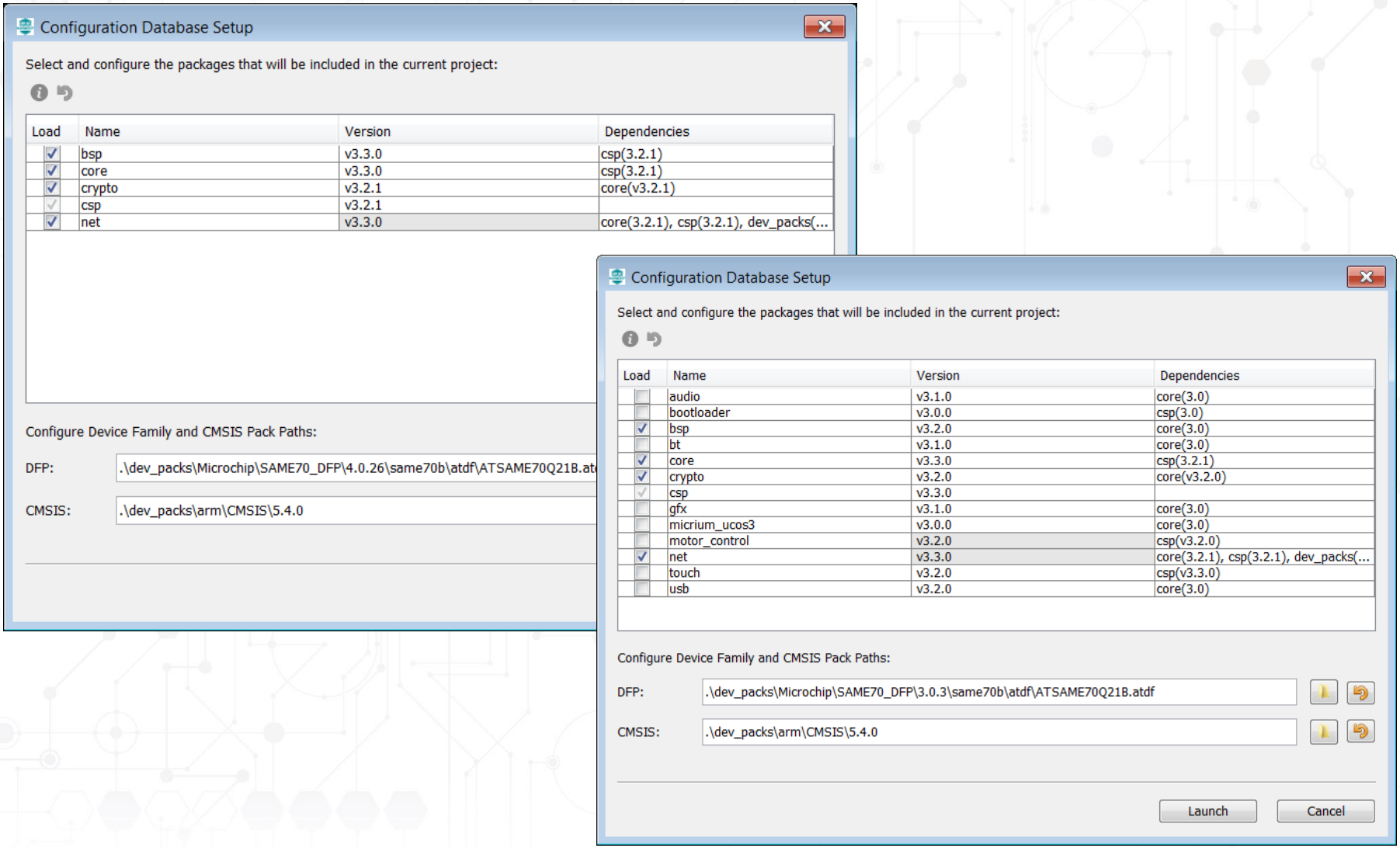
Harmony V3

MPLAB Harmony V3 on GitHub



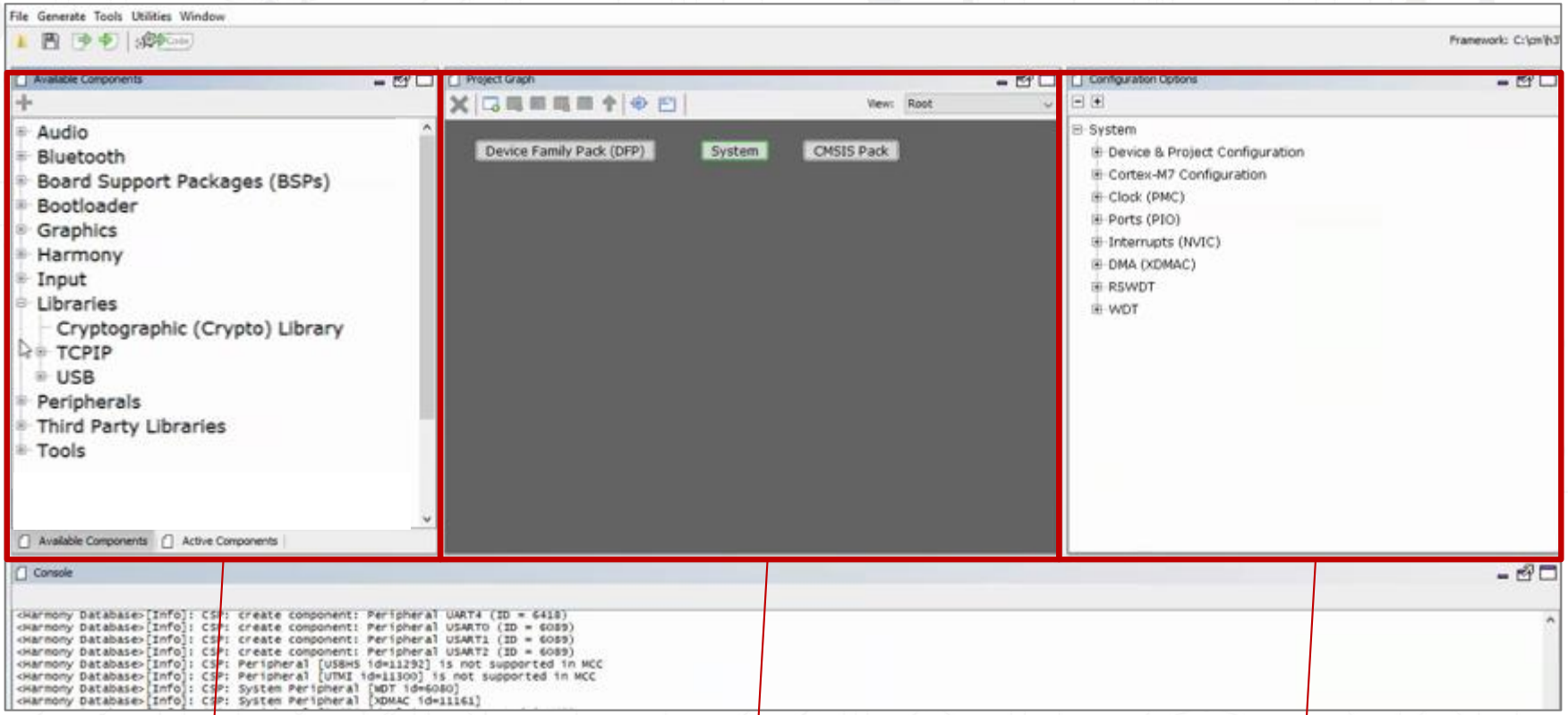
- **Modular:**
  - Individual libraries released when ready. No need to wait for complete Harmony release
  - Customer selects desired modules. No need to download everything.
- **Flexible:**
  - Ability to quickly add/modify project configuration to meet evolving customer needs
- **Open:**
  - Facilitates community content. Customer can select desired content with package manager.

MHC Git



Harmony V3

MPLAB Harmony Configurator



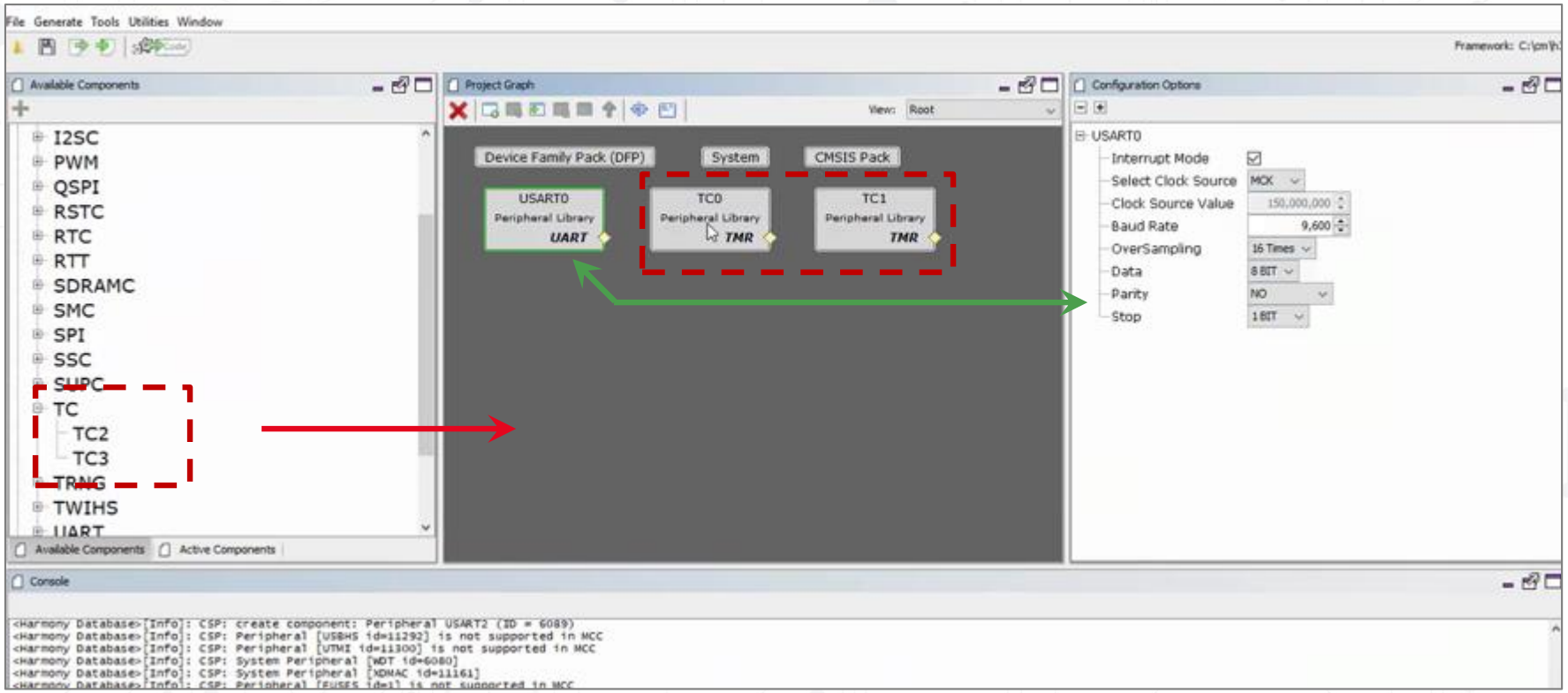
Available Components (Peripherals, Libraries,..)

Project Graph

Configuration Options

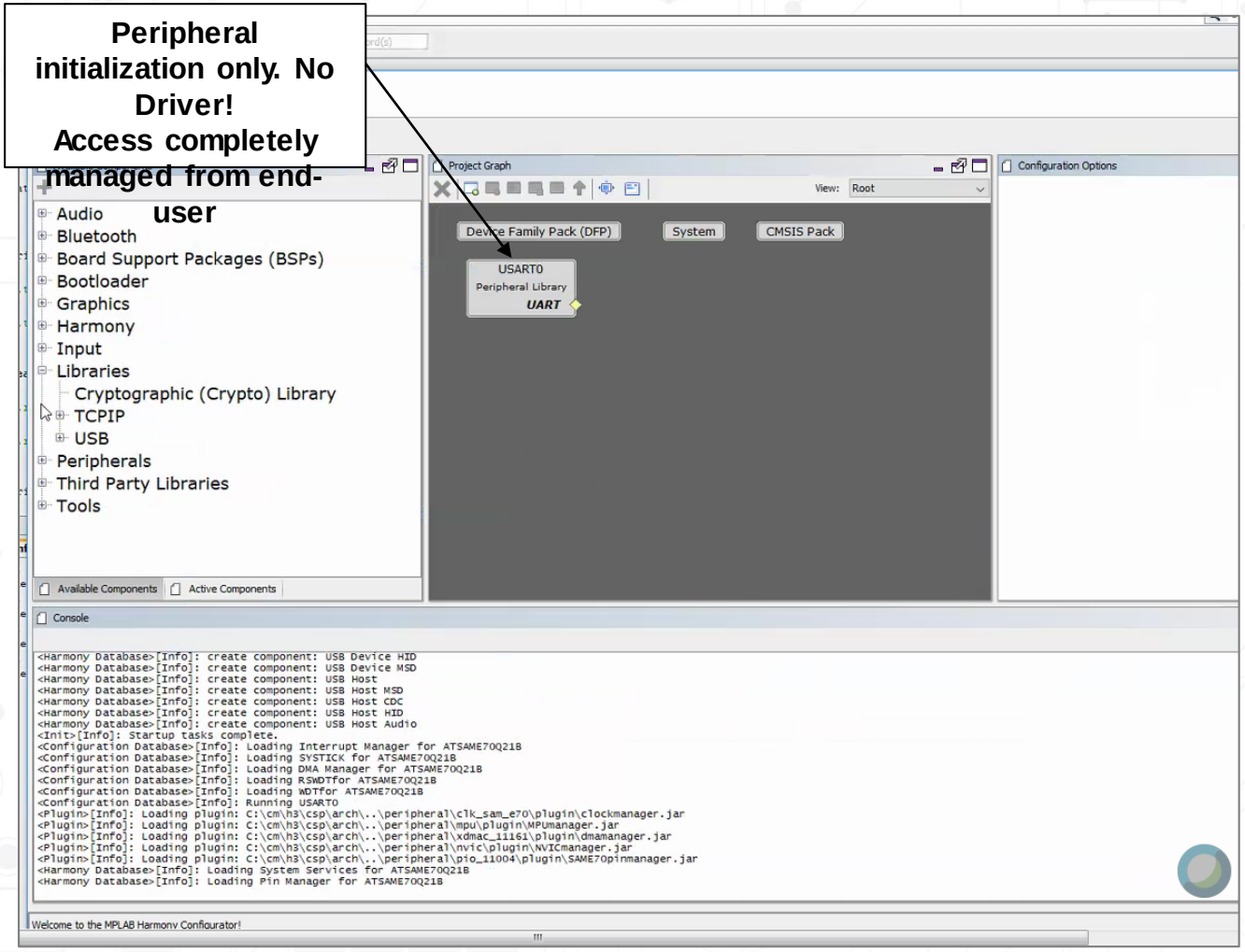
Harmony V3

MPLAB Harmony Configurator



Harmony V3

Peripheral initialization only. No Driver!
Access completely managed from end-user

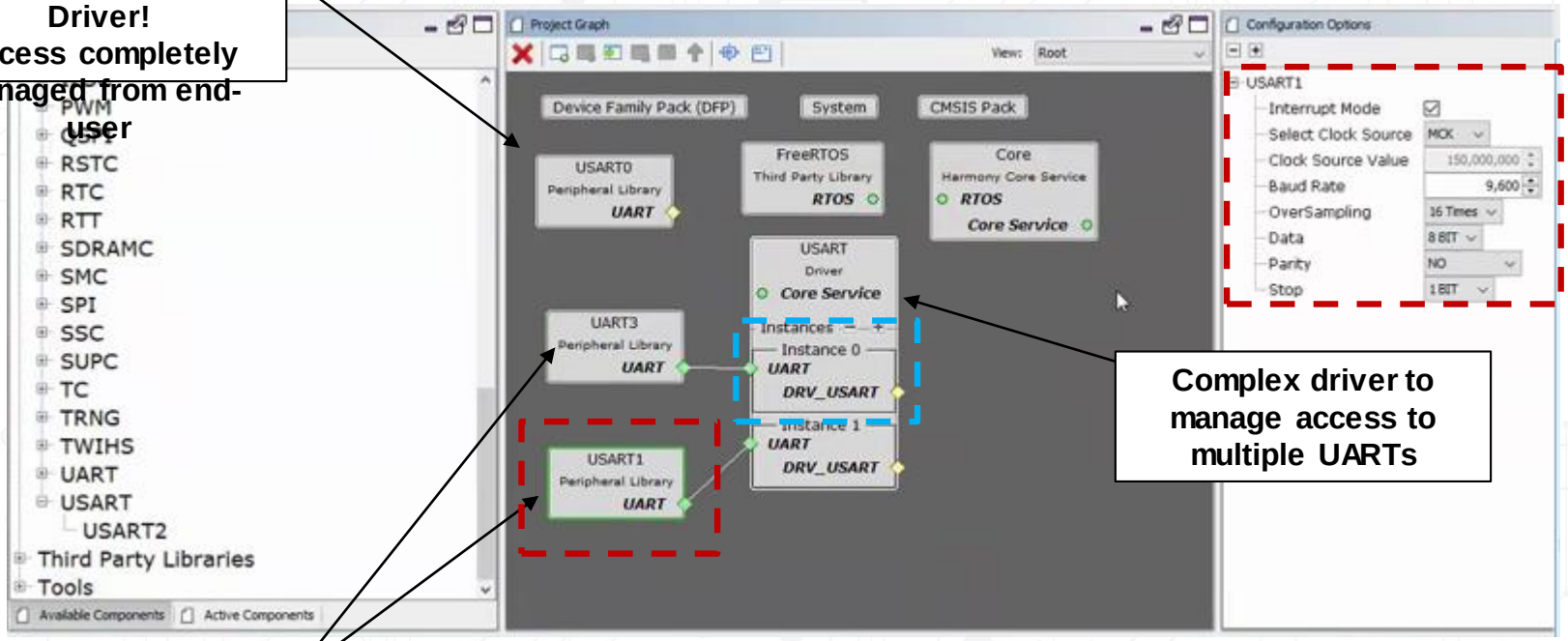


MPLAB Harmony Configurator

Harmony V3

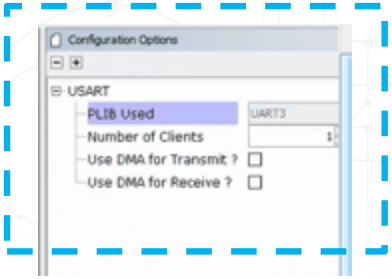
Peripheral initialization only. No Driver!
Access completely managed from end-user

MPLAB Harmony Configurator



Complex driver to manage access to multiple UARTs

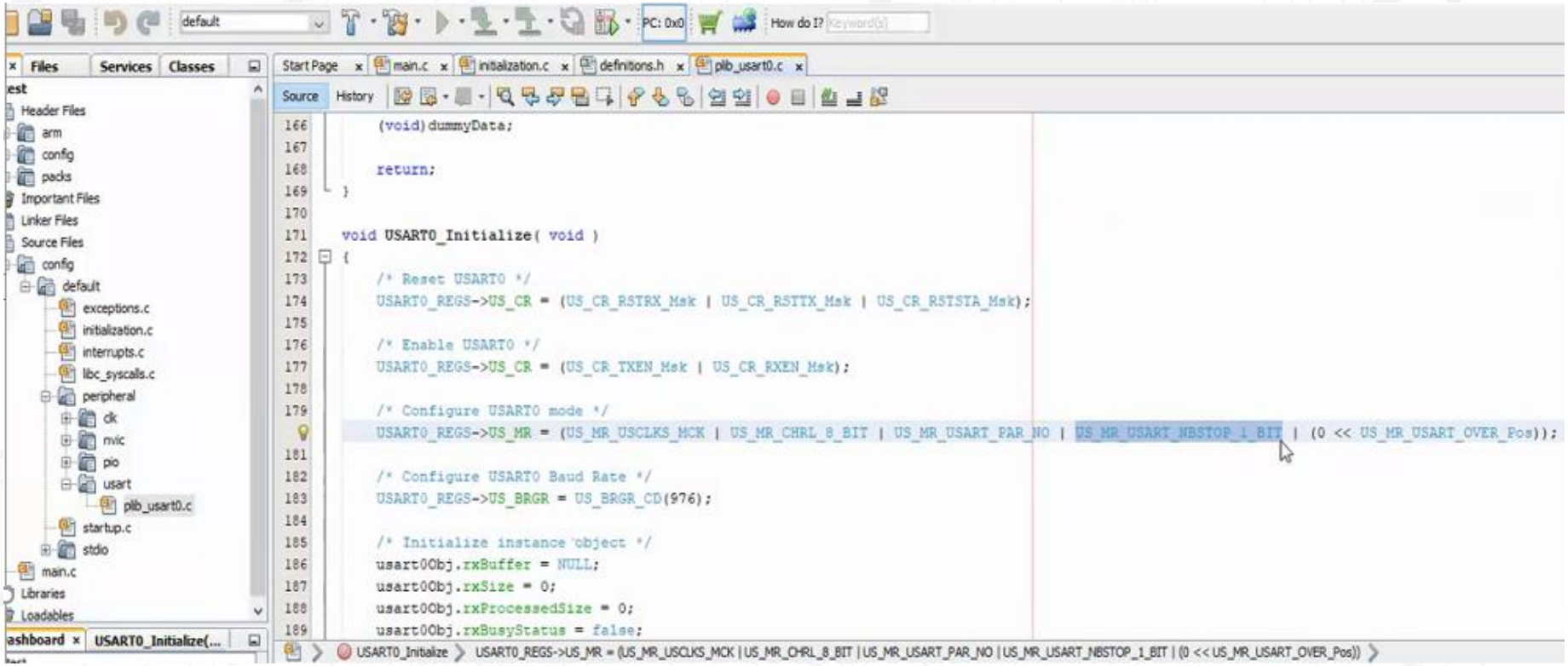
Peripheral initialization only.



MICROCHIP MASTERs 2019

Harmony V3

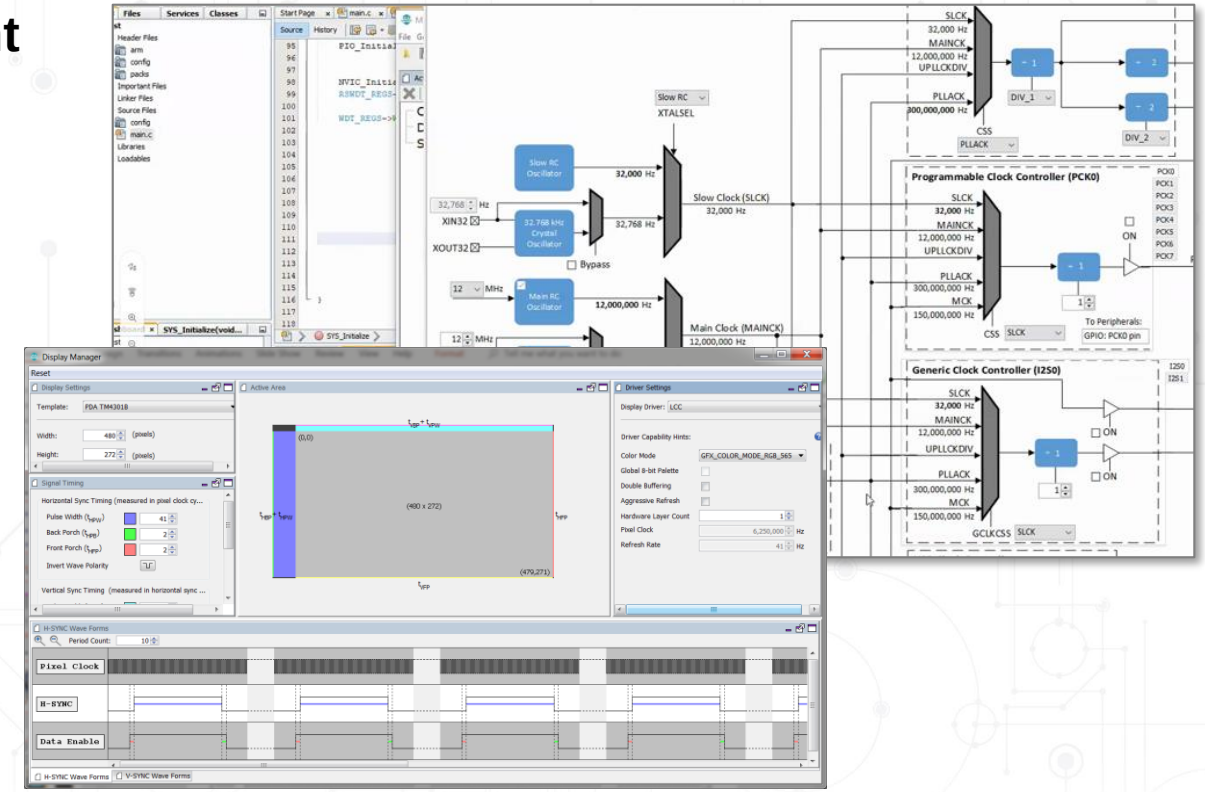
MPLAB HARMONY



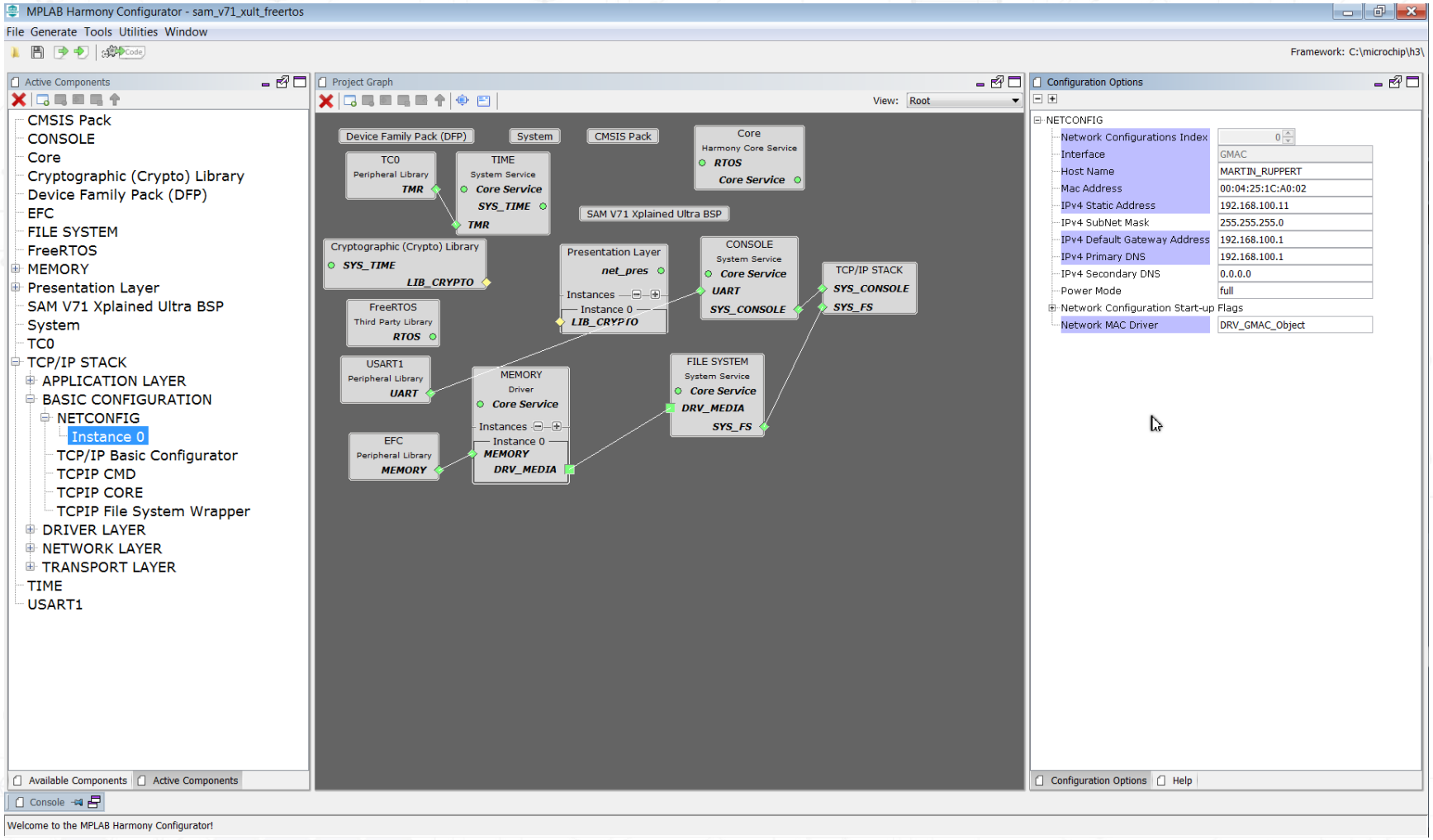
Harmony V3

Keep + improve the good things from Harmony V2

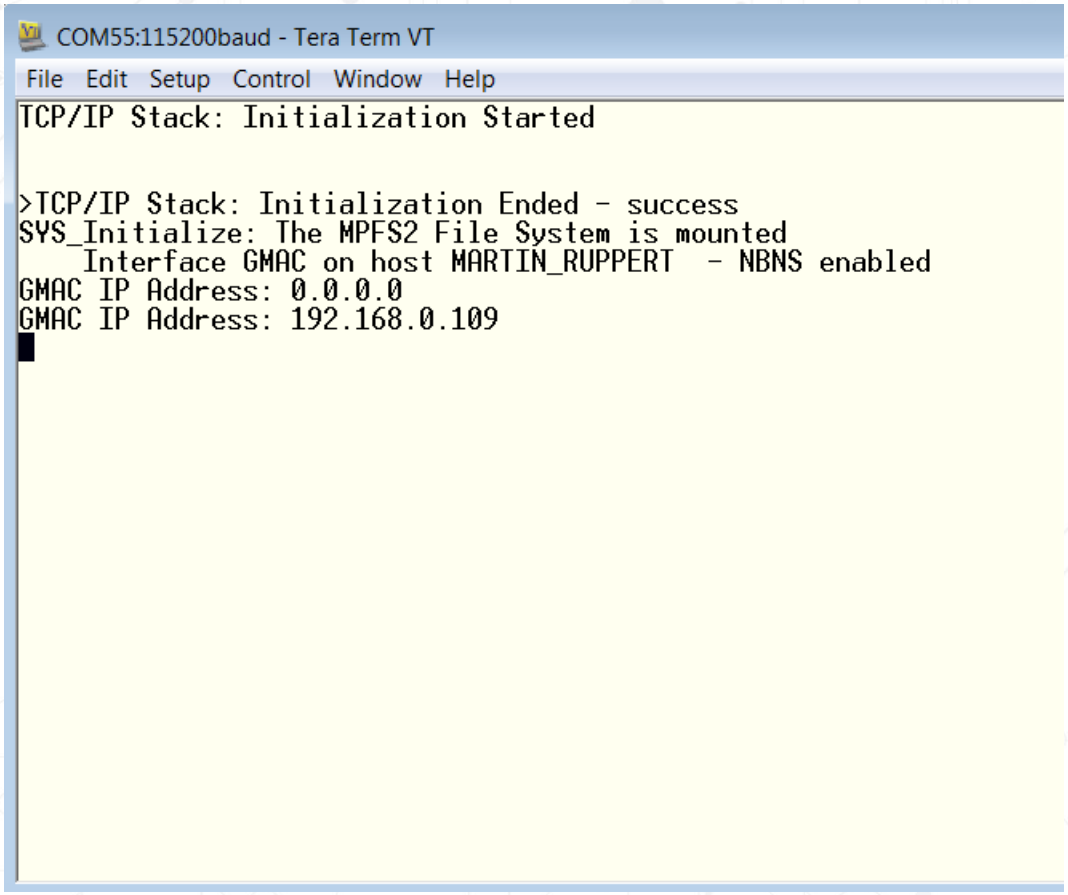
- Clock Initialization
- Pinout/I/O Management
  - Graphic
- TCP/IP and SSL1.3
  - Security
    - USB
    - Audio
    - …



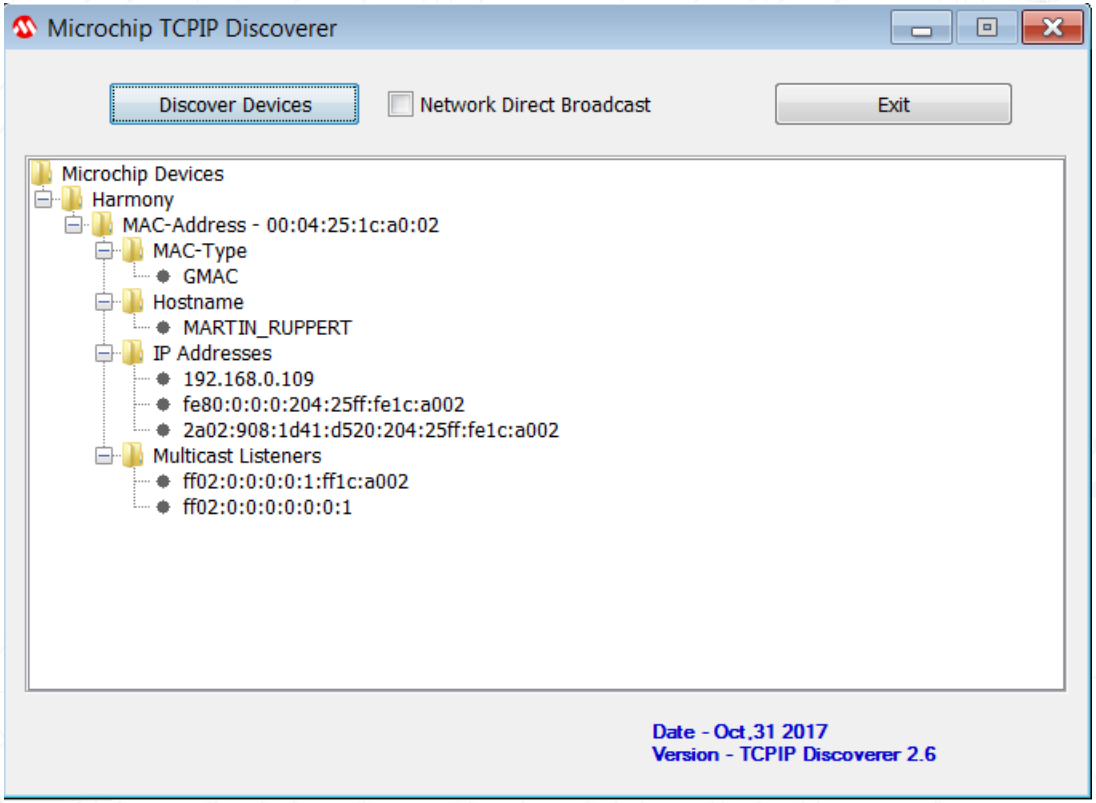
MHC



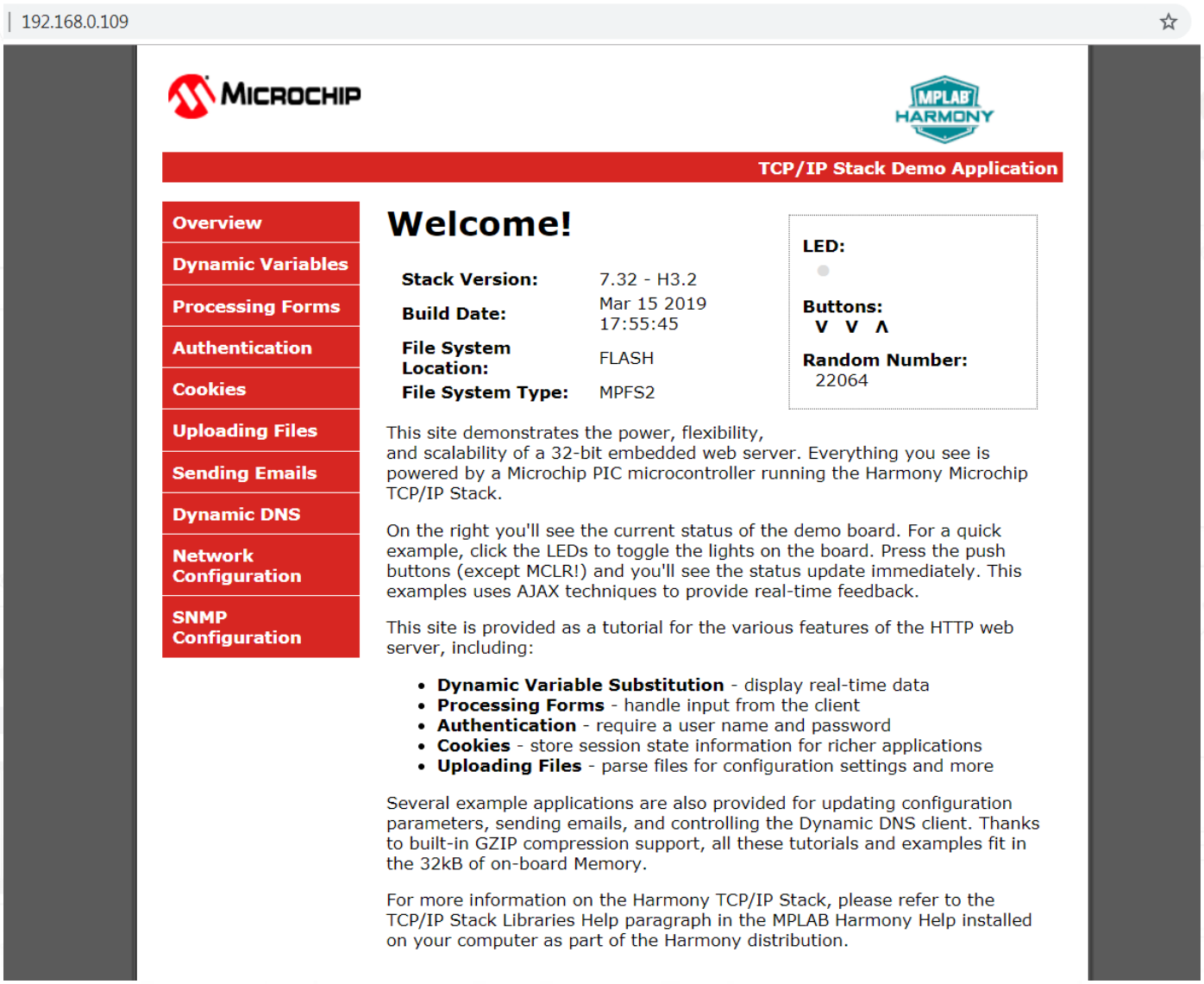
Terminal



TCPIP Discoverer



HTTP2 Webserver



Using RTOS with MPLAB Harmony

Why use RTOS?

- **Priority based task scheduling**
  - Ensures better responsiveness to events
  - A pre-emptive scheduler always runs the highest priority task that is ready to run, allowing hard real time tasks to meet its deadlines
  - Removes dependency of tasks on the execution time of other functions running in the “super loop”

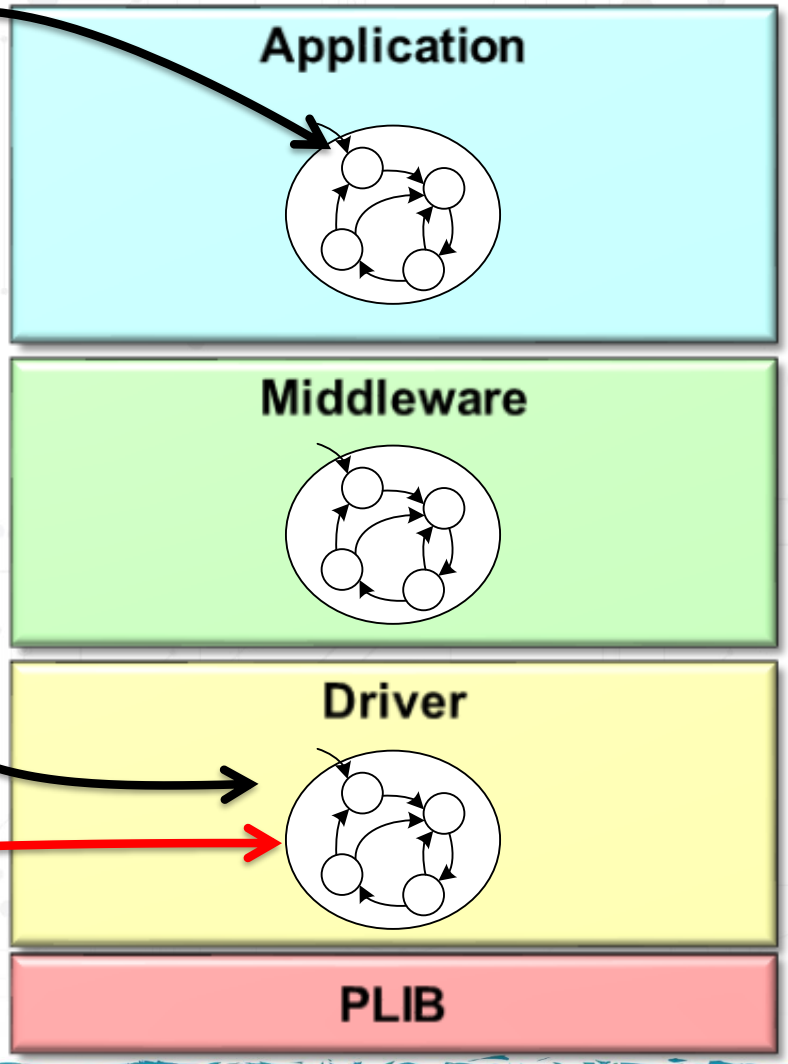
- **Efficient CPU usage**
  - Does not waste CPU cycles by running tasks waiting on events/ resources
  - Idle task is run when there are no application tasks ready to run. This allows putting CPU in low power modes

RTOS Driven

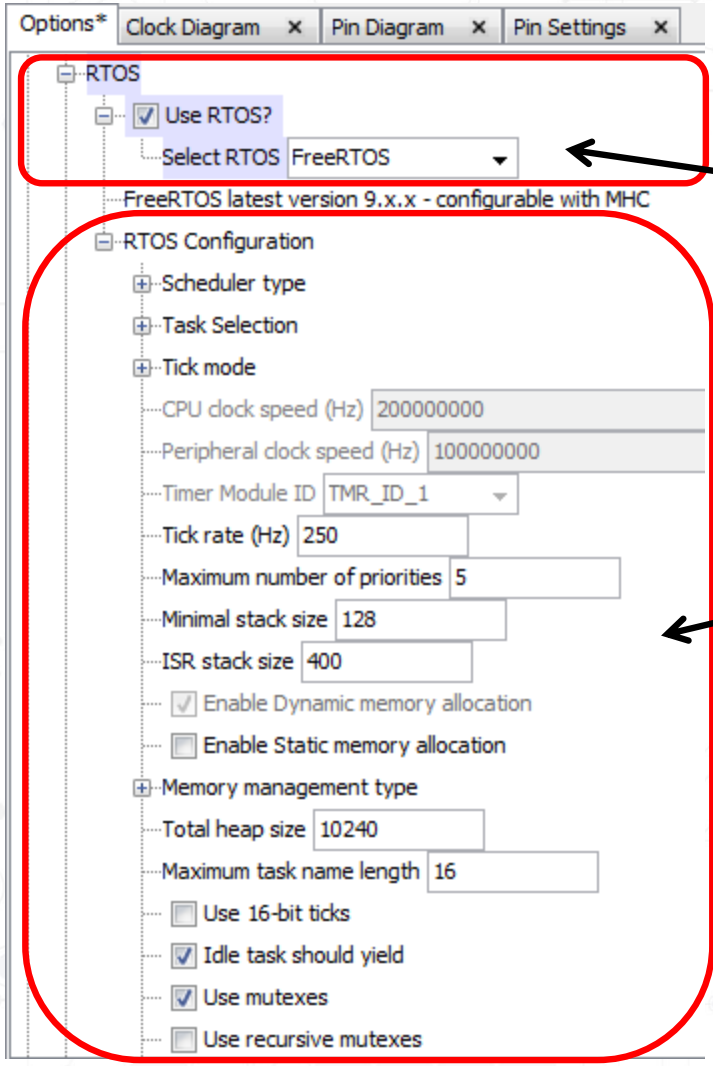
```
static void _SENSORTASK_Tasks(void)
{
    while(1)
    {
        SENSORTASK_Tasks();
        vTaskDelay(10); /*Give time back*/
    }
}
```

```
void _SYS_Tasks (void *param)
{
  while(1)
  {
    DRV_SPI_Tasks(sysObj.spiObjectIdx0);
    vTaskDelay(10);/*Give time back*/
  }
}
```

```
void SPI_ISR(void)
{
    DRV_SPI_Tasks(sysObj.spiObjectIdx0);
}
```



RTOS – MHC Configuration

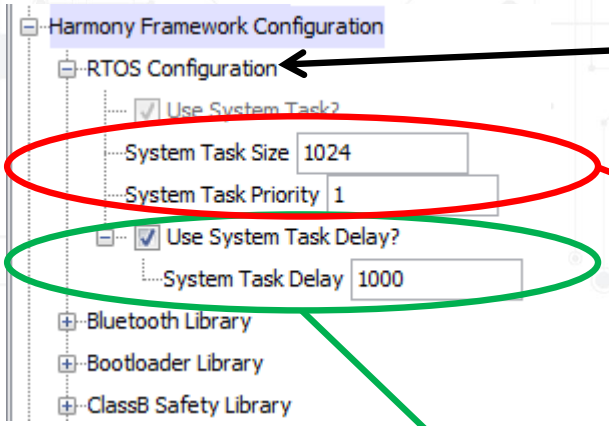


Enable and Select desired RTOS

- ❑ freeRTOS
- ❑ openRTOS
- ❑ uc/os-II
- ❑ uc/os-III
- ❑ Thread X
- ❑ embOS

Enable and Configure RTOS specific options

RTOS – MHC Configuration



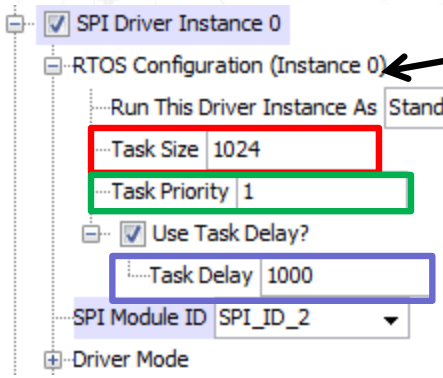
RTOS configuration for the SYS_Tasks thread.

```
void SYS_Tasks ( void )
{
   /* Create OS Thread for Sys Tasks. */
   xTaskCreate((TaskFunction_t) _SYS_Tasks,
               "Sys Tasks",
               1024, NULL, 1, NULL);
     …..
}
```

Block the task for 1000 milliseconds. Allows other tasks to run.

```
static void _SYS_Tasks ( void)
{
   while(1)
   {
      /* Maintain system services */
      SYS_CONSOLE_Tasks(sysObj.sysConsole0);

      /* Maintain Device Drivers */

      /* Maintain Middleware */


      /* Task Delay */
      vTaskDelay(1000 / portTICK_PERIOD_MS);
   }
}
```

RTOS – MHC Configuration

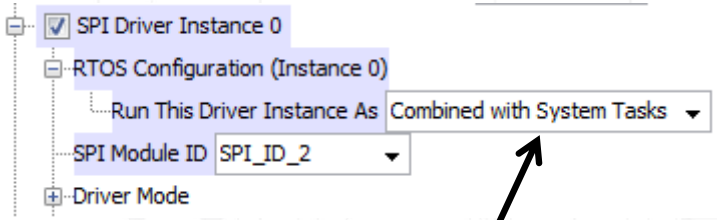


RTOS configuration for a driver instance which is configured for polled mode operation

Standalone –Run the driver task as a separate RTOS thread

```
void SYS_Tasks ( void )
{
   /* Create OS Thread for DRV_SPI  Instance 0 Tasks.*/

   xTaskCreate((TaskFunction_t) _DRV_SPI_IDX0_Tasks,
            "DRV_SPI  Instance 0 Tasks",
            1024, NULL, 1, NULL);
   ....
}
```

```
void _DRV_SPI_IDX0_Tasks(void)
{
   while(1)
   {
      DRV_SPI_Tasks(sysObj.spiObjectIdx0);

      vTaskDelay(1000 / portTICK_PERIOD_MS);
   }
}
```



Combined with System Tasks – Run the driver task as part of the _SYS_Tasks thread

```
static void _SYS_Tasks ( void)
{
   while(1)
   {
      /* Maintain system services */
      SYS_CONSOLE_Tasks(sysObj.sysConsole0);

      /* Maintain Device Drivers */
      DRV_SPI_Tasks(sysObj.spiObjectIdx0);

      /* Maintain Middleware */

      /* Task Delay */
      vTaskDelay(1000 / portTICK_PERIOD_MS);
   }
}
```

System Initialization & Tasks

```
int main(void)
{
    SYS_Initialize(NULL);

    while(true)
    {
        SYS_Tasks();
    }

    return(EXIT_VALUE);
}
```

```
void SYS_Initialize( void* data )
{
    SYS_CLK_Initialize( &clkInit );
    BSP_Initialize();
    sysObj.spiObjectIdx0 = DRV_SPI_Initialize (DRV_SPI_INDEX_0,
&drvSpi0InitData);
    sysObj.drvTmr0     = DRV_TMR_Initialize(DRV_TMR_INDEX_0,
&drvTmr0InitData);
     ……

   /*create Tasks*/
    APP_Initialize();
```

```
void SYS_Tasks ( void )
{
    /* Create OS Thread for Sys Tasks. */
    xTaskCreate((TaskFunction_t) _SYS_Tasks, "Sys Tasks",1024, NULL, 3, NULL);
    xTaskCreate((TaskFunction_t) _USB_Tasks, "USB Tasks",1024, NULL, 3, NULL);

    /* Create OS Thread for SENSORTASK Tasks. */
    xTaskCreate((TaskFunction_t) _SENSORTASK_Tasks, "SENSORTASK Tasks", 1024, NULL, 2,
NULL);

    /* Create OS Thread for EEPROMTASK Tasks. */
    xTaskCreate((TaskFunction_t) _EEPROM_Tasks, "EEPROM Tasks", 1024, NULL, 1, NULL);
    …….

     /**************
     * Start RTOS *
     **************/
    vTaskStartScheduler(); /* This function never returns. */
```

MICROCHIP MASTERs 2019

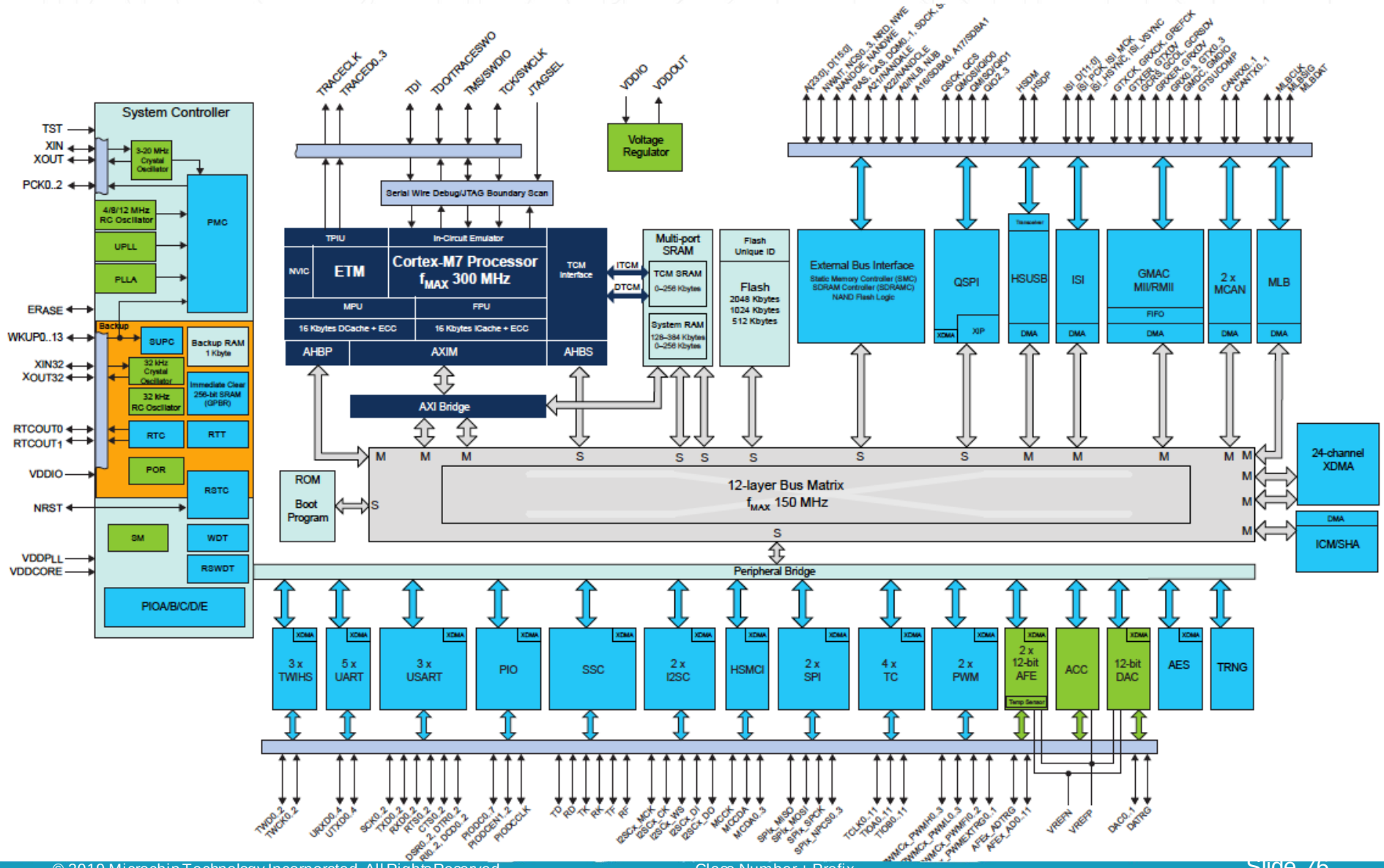
The Hardware for this Class

SMART | CONNECTED | SECURE

MICROCHIP

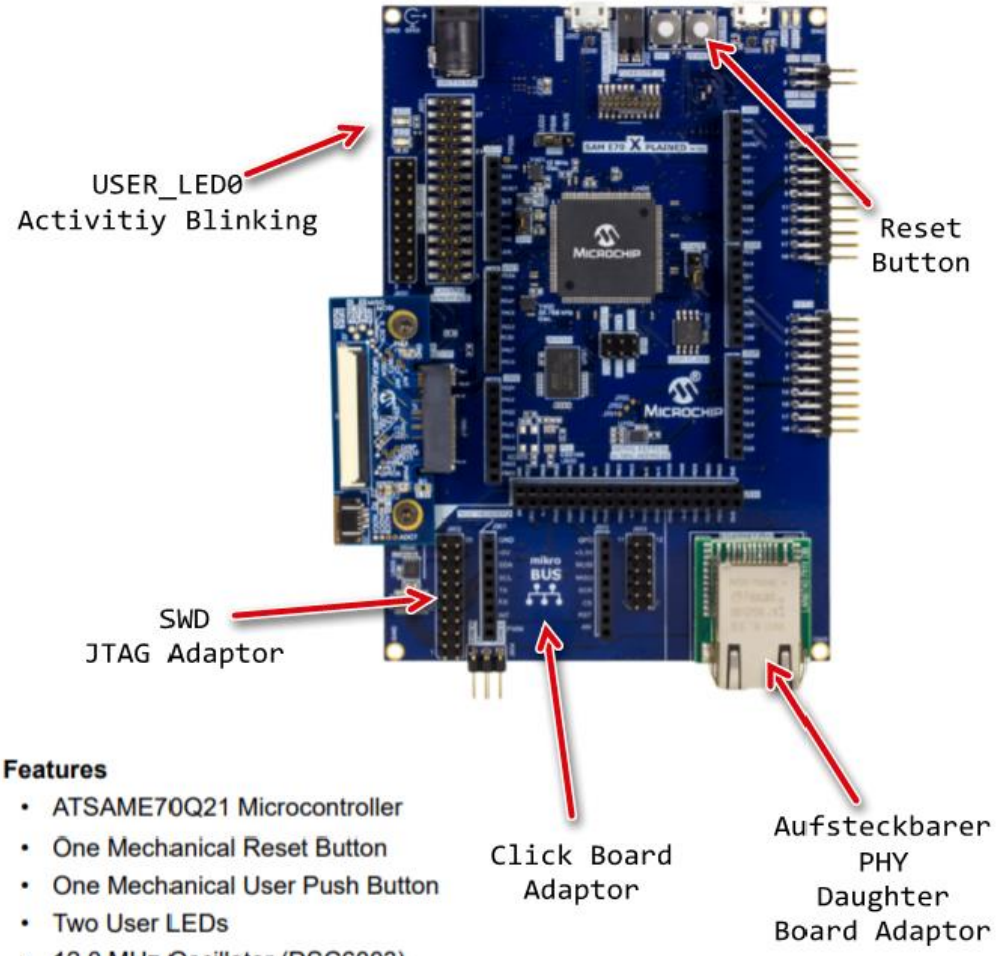
MICROCHIP
MASTERs 2019

SAME70



SAME70 Xplained Ultra

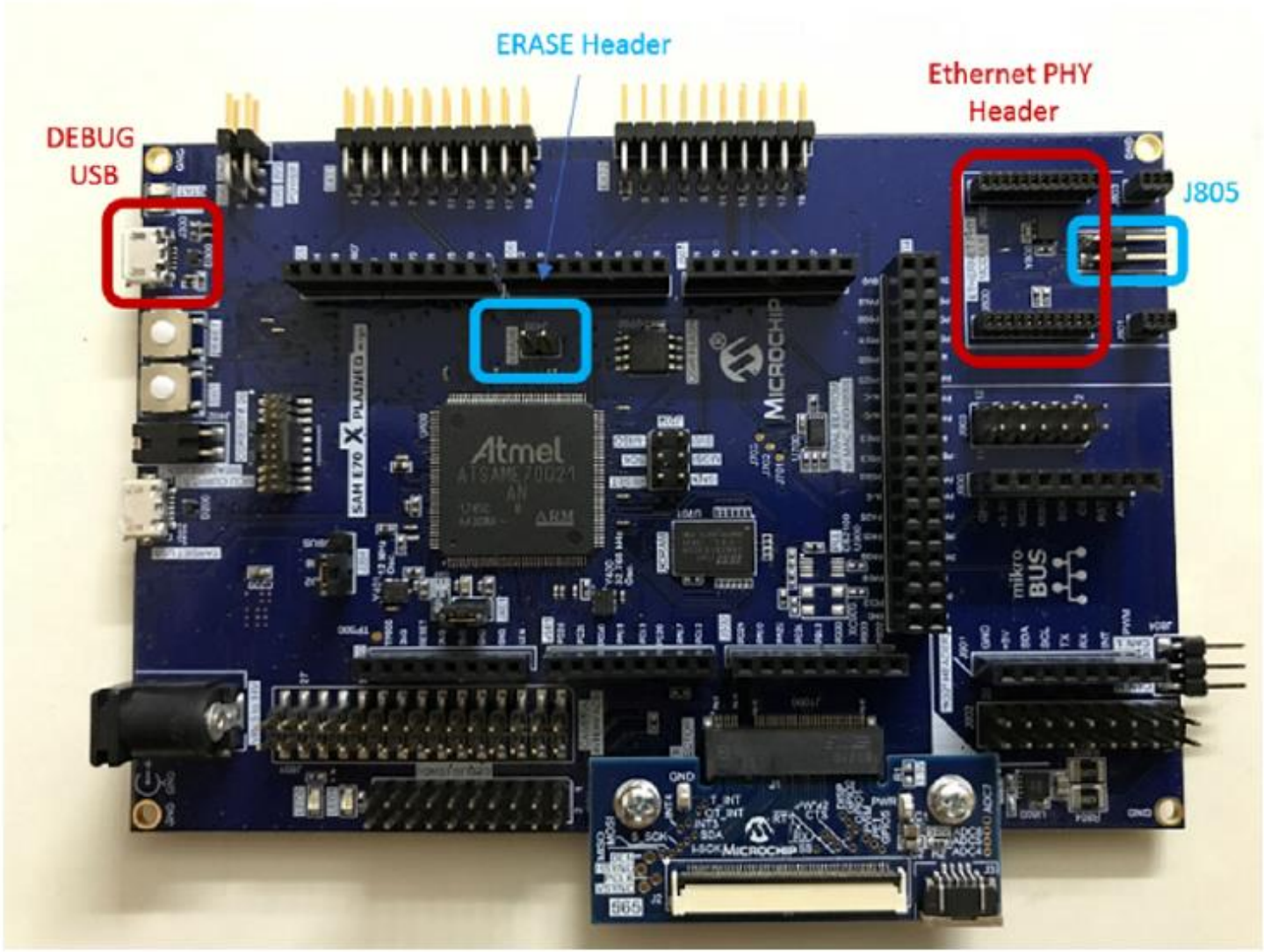
SAME70 Xplained Ultra



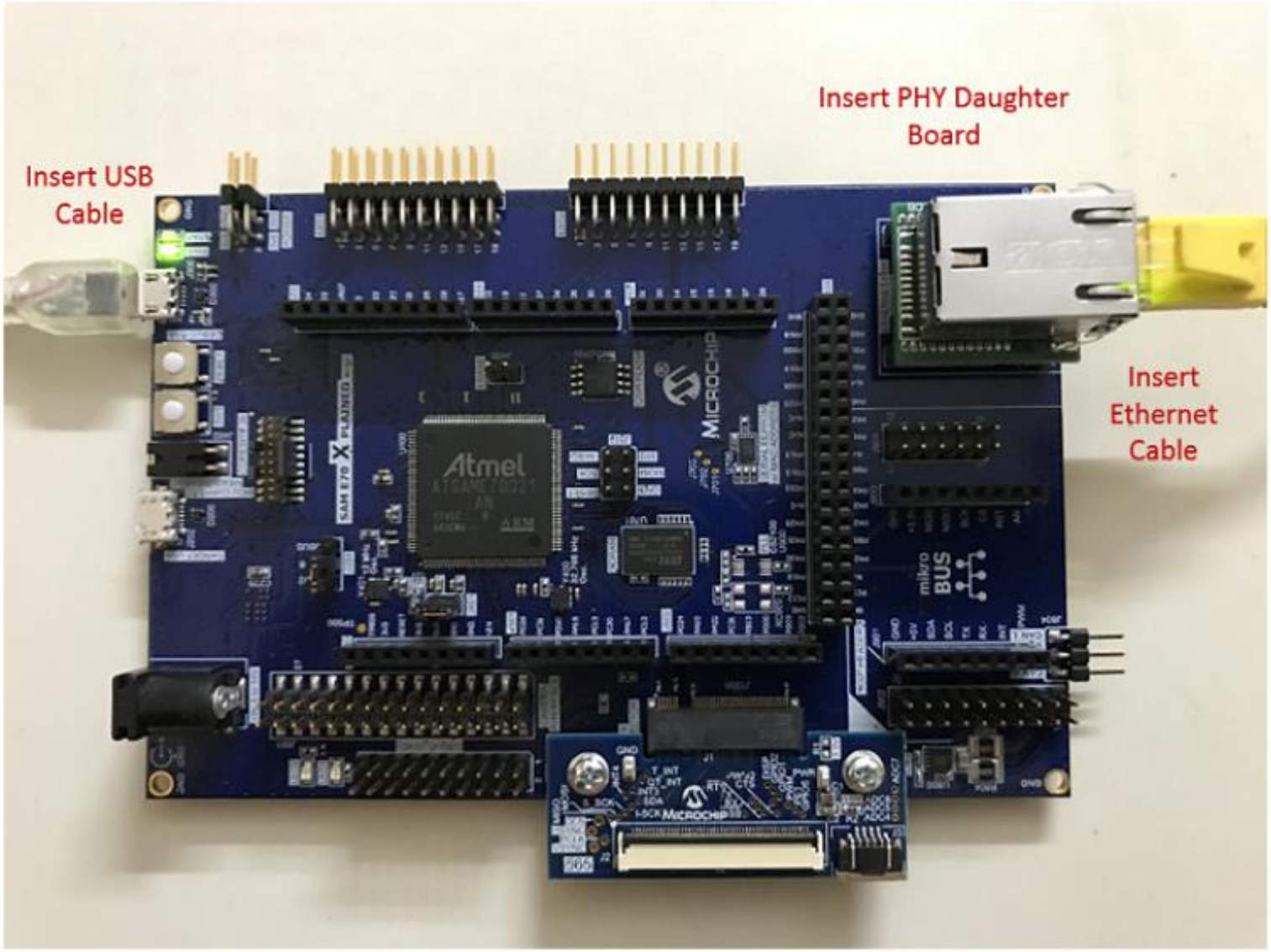
Features

- ATSAME70Q21 Microcontroller
- One Mechanical Reset Button
- One Mechanical User Push Button
- Two User LEDs
- 12.0 MHz Oscillator (DSC6003)
- 32.768 kHz Oscillator (DSC6083)
- 2-MB SDRAM
- 4-MB QSPI Flash (SST26VF032BA)

SAME70 Xplained Ultra



SAME70 Xplained Ultra



Now it is time for Lab1

A walk through an existing TCP Application, adding some changes with Microchip Harmony Configurator And playing with the tools

Using the MPLAB® Harmony TCP/IP Stack

BERKELEY MODULE

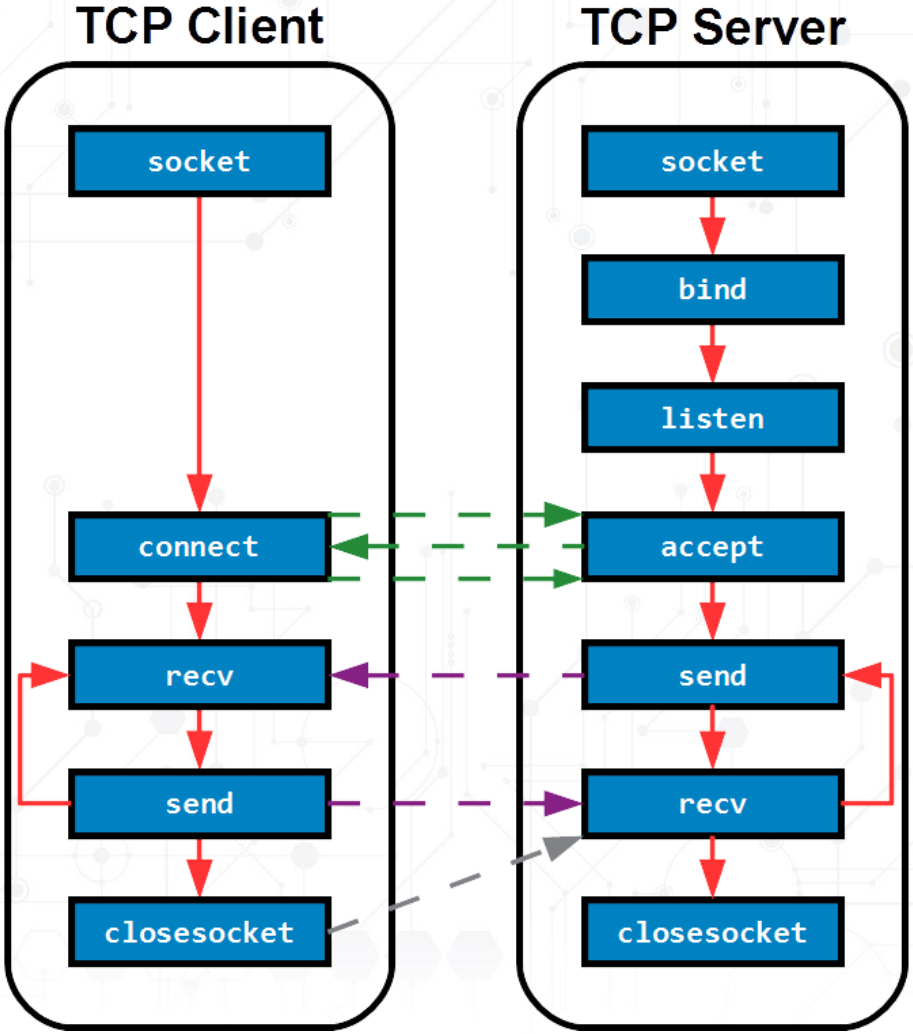
Berkeley Socket

- Berkeley sockets is an application programming interface (API) for Internet sockets and Unix domain sockets, used for inter-process communication (IPC). It is commonly implemented as a library of linkable modules. It originated with the 4.2BSD Unix operating system, released in 1983.
- A socket is an abstract representation (handle) for the local endpoint of a network communication path. The Berkeley sockets API represents it as a file descriptor (file handle) in the Unix philosophy that provides a common interface for input and output to streams of data.

Berkeley APIs

API Name	Host	Description
socket	Server/Client	Creates a new BSD socket.
bind	Server	Assigns the local address of the communication endpoint.
connect	Client	Assigns the address of the peer communications endpoint, and establishes a connection between the end points.
listen	Server	Sets the specified socket in a listen mode and ready to accept connection requests.
accept	Server	Accept connection requests queued for a listening socket.
send	Server/Client	Send outgoing data on an already connected socket.
recv	Server/Client	Receive incoming data that has been queued for a socket.
closesocket	Server/Client	Closes an existing socket.

BSD Socket Application Flow



Server and Client both create a socket that uses the TCP Protocol.

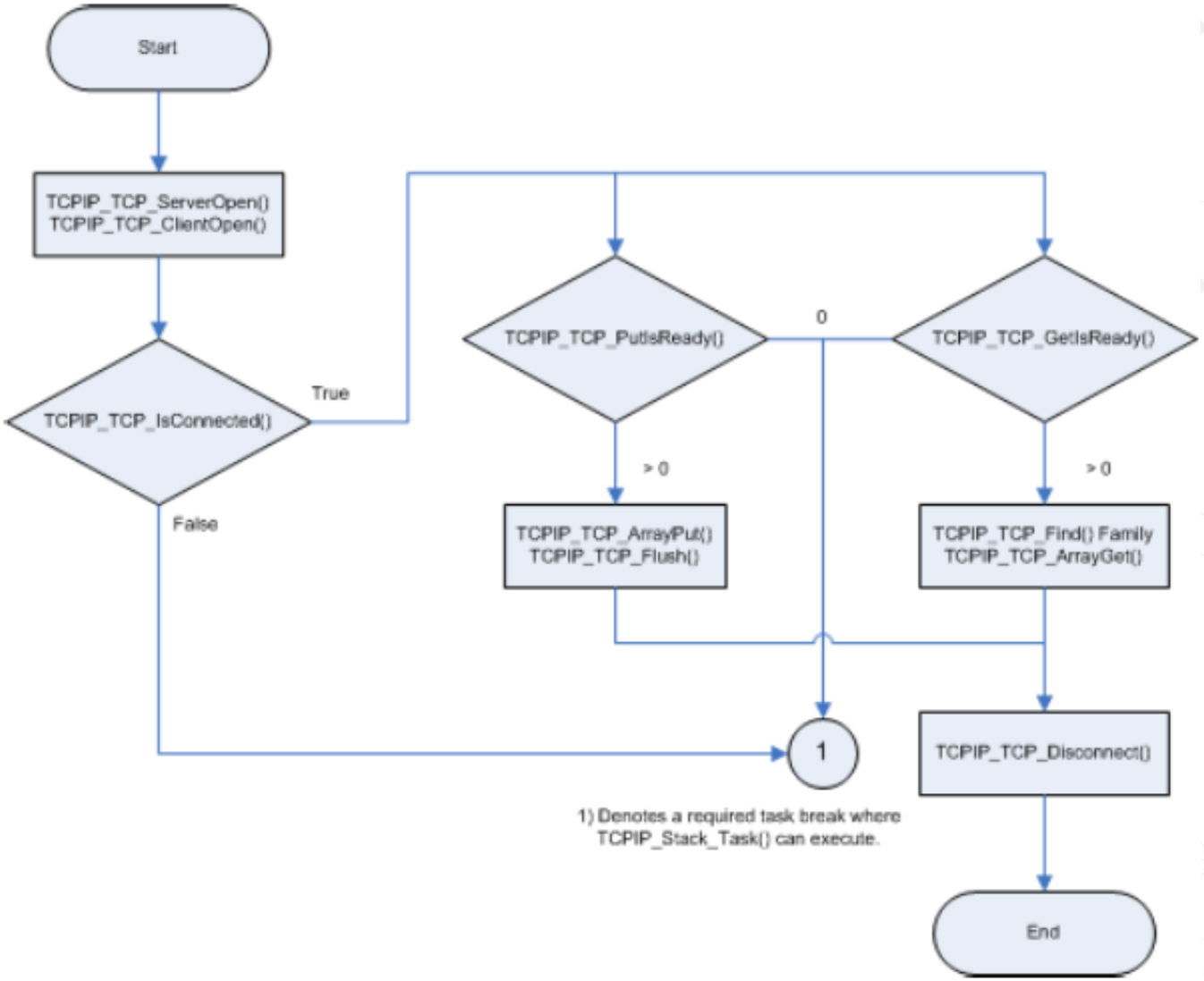
Client attempts to establish a TCP connection with the Server using a 3-Way Handshake.

Server Sends data to Client, recv returns a value > 0.

Client Sends data to Server, recv returns a value > 0.

Client Closes Socket, recv returns a value of 0. Server closes Socket.

H3 Socket Application Flow



Tips

- The Berkeley APIs available in the Harmony Framework can only operate in non-blocking mode.
- TCP/IP Applications using Berkeley Module APIs should be implemented using a non-blocking state machine.
- Example application source code can be found in the app.c file in the following Harmony TCP/IP Demonstration Projects:
  - *berkeley_tcp_client*
  - *berkeley_tcp_server*
  - *berkeley_udp_client*
  - *berkeley_udp_server*

Now it is time for Lab2

A complete Vending Machine Service Application

LAB 2

- Vending Machine
- Libraries to establish communication between a TCP Client and Server
- Dynamic variables and GET request from an HTTP client handled in Harmony HTTP server

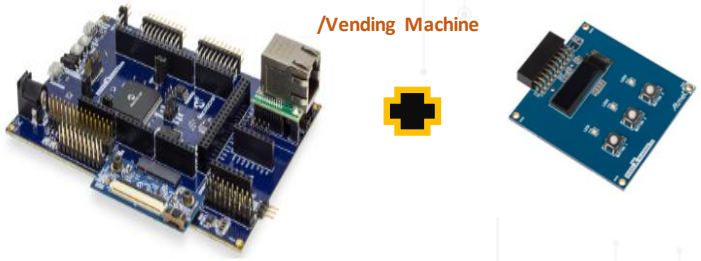
LAB 2- TASK 1

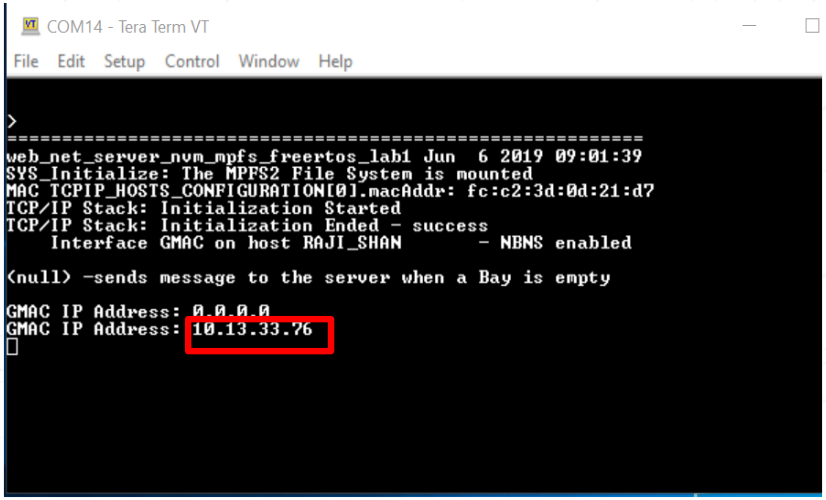
Hardware

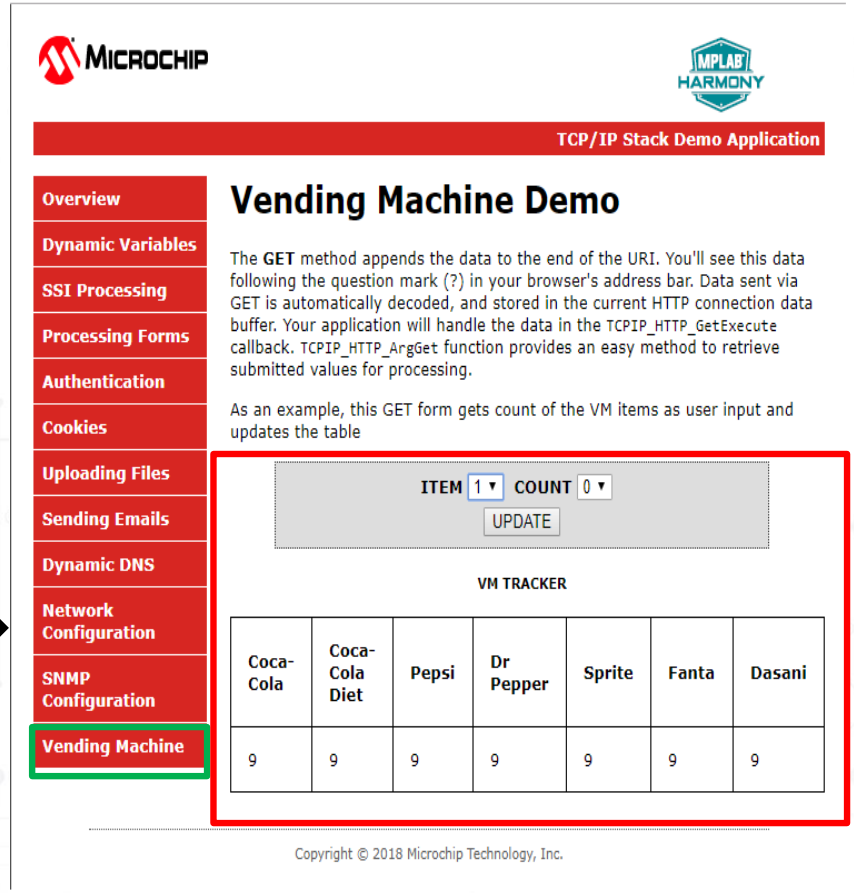
- SAM E70 Xplained Ultra Evaluation Kit
- OLED1 Xplained Pro Extension Kit

E70 WEB SERVER

/Vending Machine

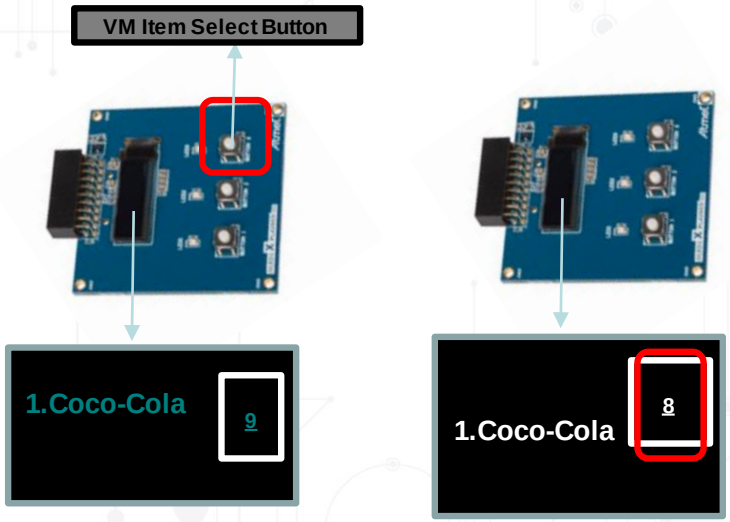






LAB2 –TASK1

Server updates the VM Client



MICROCHIP

MPLAB HARMONY

TCP/IP Stack Demo Application

- Overview
- Dynamic Variables
- SSI Processing
- Processing Forms
- Authentication
- Cookies
- Uploading Files
- Sending Emails
- Dynamic DNS
- Network Configuration
- SNMP Configuration
- Vending Machine

Vending Machine Demo

The **GET** method appends the data to the end of the URI. You'll see this data following the question mark (?) in your browser's address bar. Data sent via GET is automatically decoded, and stored in the current HTTP connection data buffer. Your application will handle the data in the `TCPIP_HTTP_GetExecute` callback. `TCPIP_HTTP_ArgGet` function provides an easy method to retrieve submitted values for processing.

As an example, this GET form gets count of the VM items as user input and updates the table

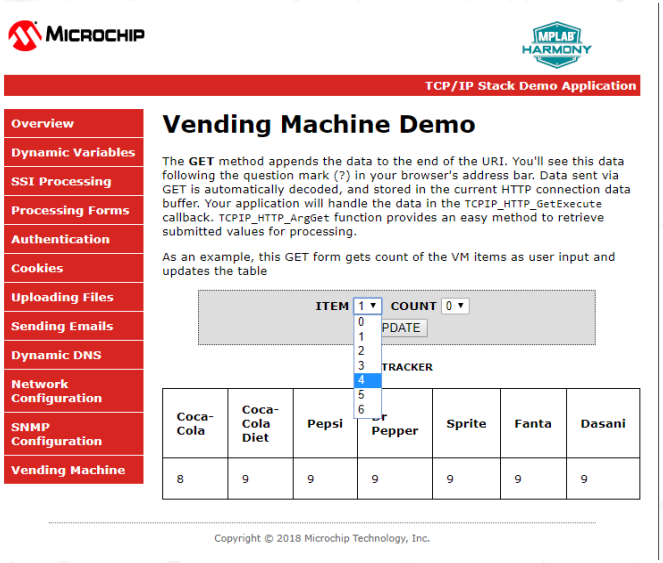
ITEM 1 ▾ COUNT 0 ▾

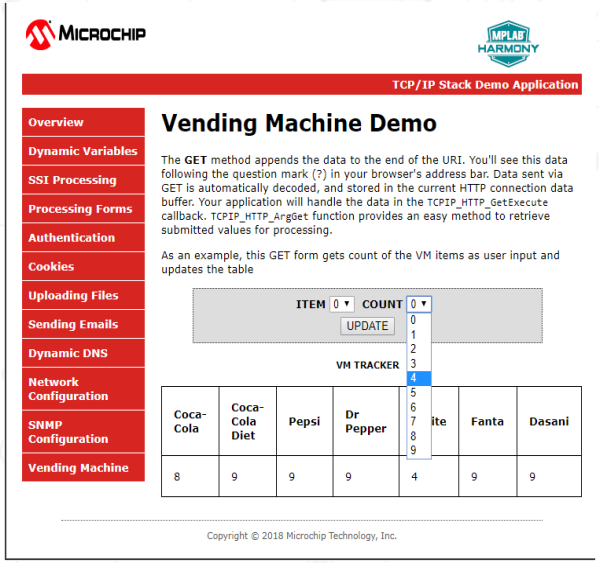
UPDATE

VM TRACKER

Coca-Cola	Coca-Cola Diet	Pepsi	Dr Pepper	Sprite	Fanta	Dasani
8	9	9	9	9	9	9

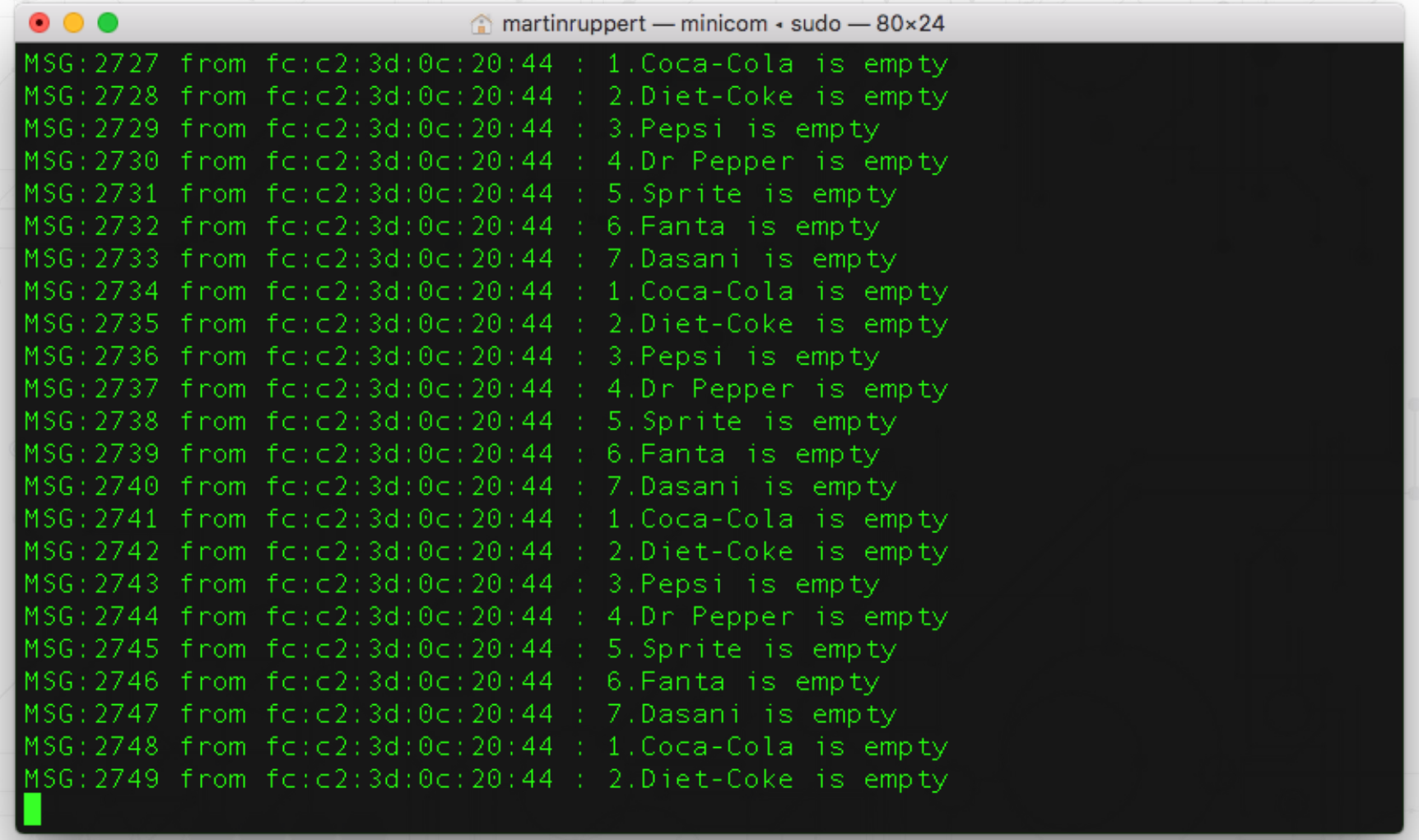
LAB 2 –Task 1



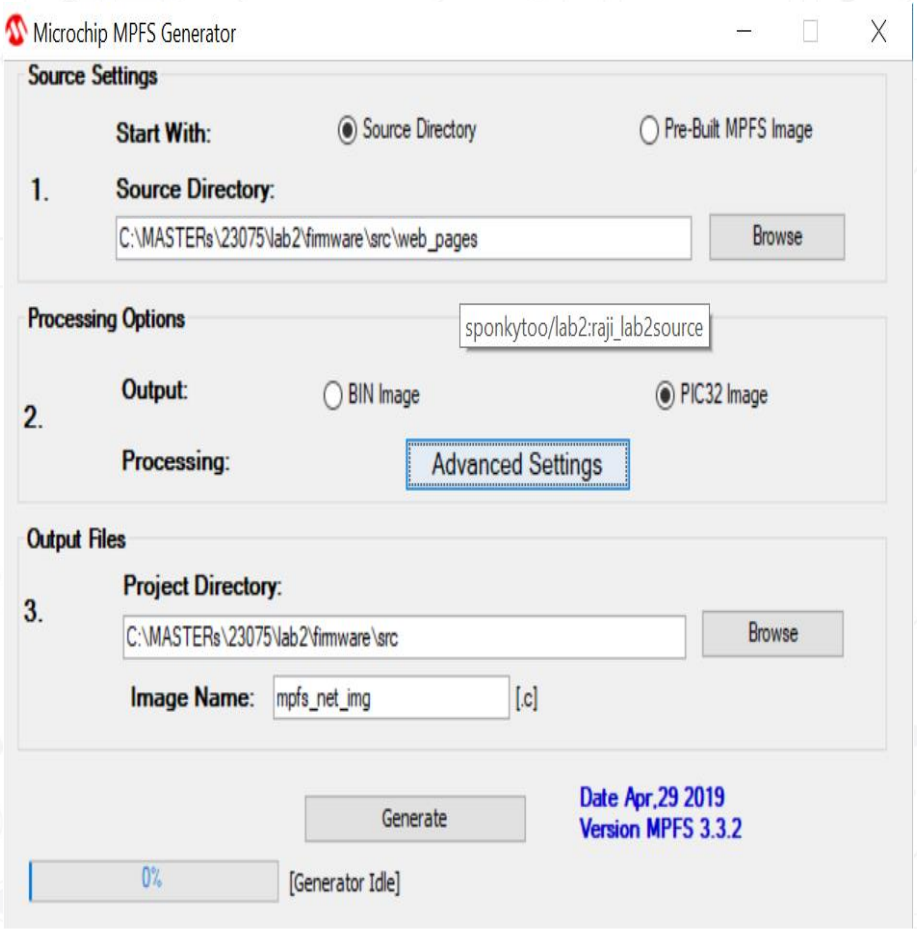


4. Dr Pepper	4

VM Server



LAB 2 – MPFS Generator



- Packages web pages into efficient storage format
- Microchip MPFS Utility
- Can include the utility on the build process

Using RTOS with MPLAB Harmony

Why use RTOS?

- **Priority based task scheduling**
  - Ensures better responsiveness to events
  - A pre-emptive scheduler always runs the highest priority task that is ready to run, allowing hard real time tasks to meet its deadlines
  - Removes dependency of tasks on the execution time of other functions running in the “super loop”

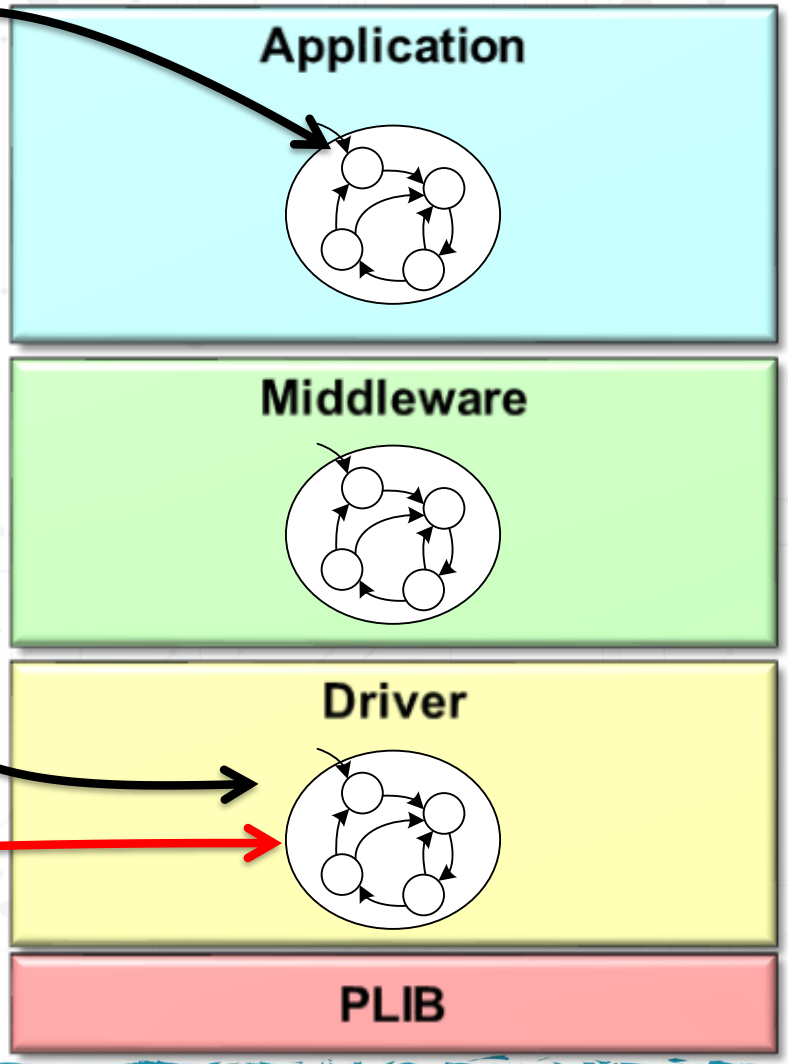
- **Efficient CPU usage**
  - Does not waste CPU cycles by running tasks waiting on events/ resources
  - Idle task is run when there are no application tasks ready to run. This allows putting CPU in low power modes

RTOS Driven

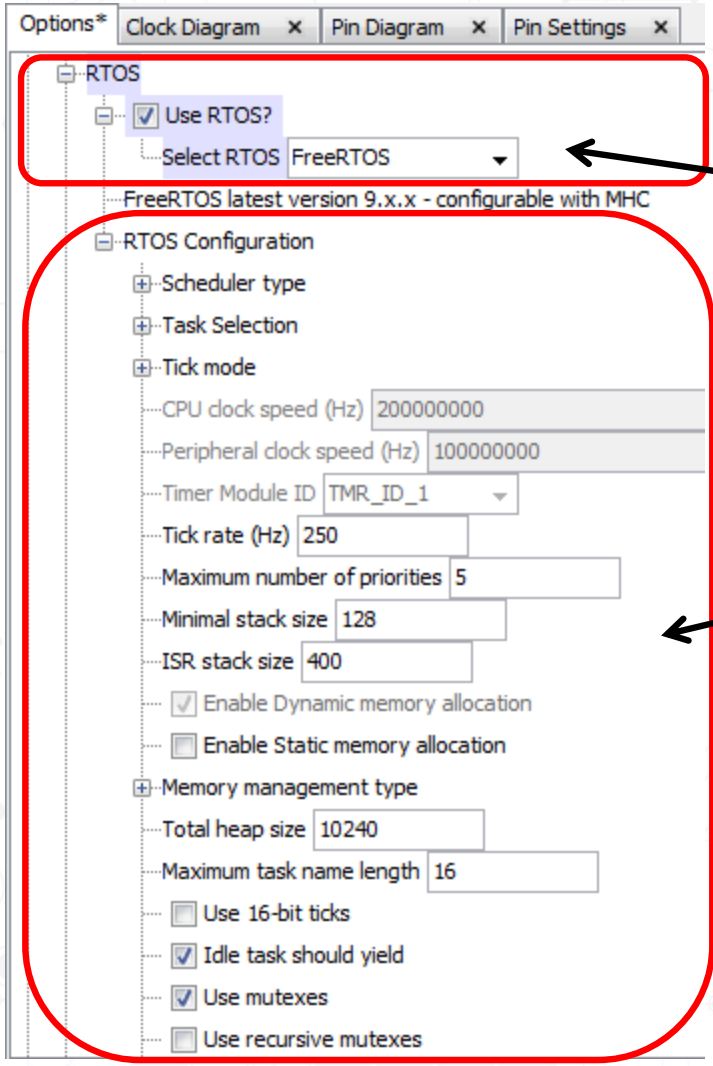
```
static void _SENSORTASK_Tasks(void)
{
    while(1)
    {
        SENSORTASK_Tasks();
        vTaskDelay(10); /*Give time back*/
    }
}
```

```
void _SYS_Tasks (void *param)
{
  while(1)
  {
    DRV_SPI_Tasks(sysObj.spiObjectIdx0);
    vTaskDelay(10);/*Give time back*/
  }
}
```

```
void SPI_ISR(void)
{
    DRV_SPI_Tasks(sysObj.spiObjectIdx0);
}
```



RTOS – MHC Configuration

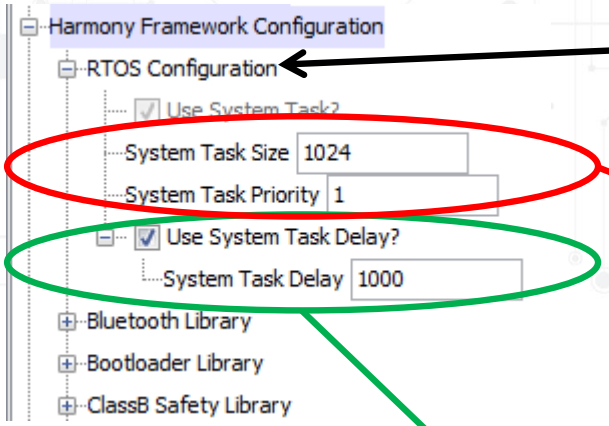


Enable and Select desired RTOS

- ❑ freeRTOS
- ❑ openRTOS
- ❑ uc/os-II
- ❑ uc/os-III
- ❑ Thread X
- ❑ embOS

Enable and Configure RTOS specific options

RTOS – MHC Configuration



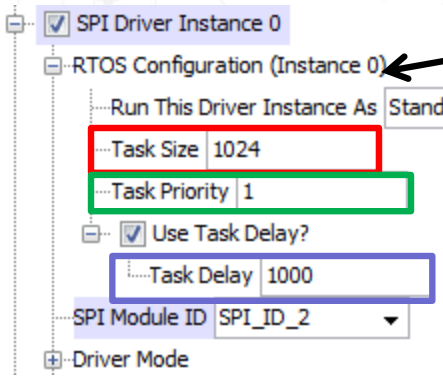
RTOS configuration for the SYS_Tasks thread.

```
void SYS_Tasks ( void )
{
   /* Create OS Thread for Sys Tasks. */
   xTaskCreate((TaskFunction_t) _SYS_Tasks,
               "Sys Tasks",
               1024, NULL, 1, NULL);
     …..
}
```

Block the task for 1000 milliseconds. Allows other tasks to run.

```
static void _SYS_Tasks ( void)
{
   while(1)
   {
      /* Maintain system services */
      SYS_CONSOLE_Tasks(sysObj.sysConsole0);

      /* Maintain Device Drivers */

      /* Maintain Middleware */


      /* Task Delay */
      vTaskDelay(1000 / portTICK_PERIOD_MS);
   }
}
```

RTOS – MHC Configuration

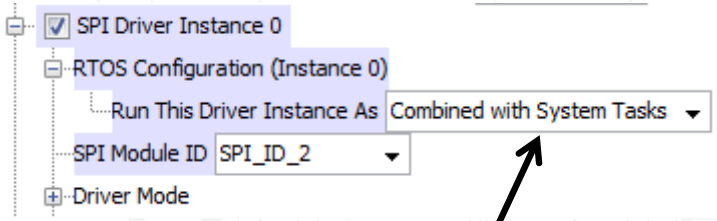


RTOS configuration for a driver instance which is configured for polled mode operation

Standalone –Run the driver task as a separate RTOS thread

```
void SYS_Tasks ( void )
{
   /* Create OS Thread for DRV_SPI  Instance 0 Tasks.*/

   xTaskCreate((TaskFunction_t) _DRV_SPI_IDX0_Tasks,
            "DRV_SPI  Instance 0 Tasks",
            1024, NULL, 1, NULL);
   ....
}
```

```
void _DRV_SPI_IDX0_Tasks(void)
{
   while(1)
   {
      DRV_SPI_Tasks(sysObj.spiObjectIdx0);

      vTaskDelay(1000 / portTICK_PERIOD_MS);
   }
}
```



Combined with System Tasks – Run the driver task as part of the _SYS_Tasks thread

```
static void _SYS_Tasks ( void)
{
   while(1)
   {
      /* Maintain system services */
      SYS_CONSOLE_Tasks(sysObj.sysConsole0);

      /* Maintain Device Drivers */
      DRV_SPI_Tasks(sysObj.spiObjectIdx0);

      /* Maintain Middleware */

      /* Task Delay */
      vTaskDelay(1000 / portTICK_PERIOD_MS);
   }
}
```

System Initialization & Tasks

```
int main(void)
{
    SYS_Initialize(NULL);

    while(true)
    {
        SYS_Tasks();
    }

    return(EXIT_VALUE);
}
```

```
void SYS_Initialize( void* data )
{
    SYS_CLK_Initialize( &clkInit );
    BSP_Initialize();
    sysObj.spiObjectIdx0 = DRV_SPI_Initialize (DRV_SPI_INDEX_0,
&drvSpi0InitData);
    sysObj.drvTmr0     = DRV_TMR_Initialize(DRV_TMR_INDEX_0,
&drvTmr0InitData);
     ……

   /*create Tasks*/
    APP_Initialize();
```

```
void SYS_Tasks ( void )
{
    /* Create OS Thread for Sys Tasks. */
    xTaskCreate((TaskFunction_t) _SYS_Tasks, "Sys Tasks",1024, NULL, 3, NULL);
    xTaskCreate((TaskFunction_t) _USB_Tasks, "USB Tasks",1024, NULL, 3, NULL);

    /* Create OS Thread for SENSORTASK Tasks. */
    xTaskCreate((TaskFunction_t) _SENSORTASK_Tasks, "SENSORTASK Tasks", 1024, NULL, 2,
NULL);

    /* Create OS Thread for EEPROMTASK Tasks. */
    xTaskCreate((TaskFunction_t) _EEPROM_Tasks, "EEPROM Tasks", 1024, NULL, 1, NULL);
    …….

     /**************
     * Start RTOS *
     **************/
    vTaskStartScheduler(); /* This function never returns. */
```

Data Protocol

- Data exchanged through a TCP/UDP Socket can use a custom protocol or a standard protocol such as HTTP or MQTT.
- Data can be formatted using a standard data-interchange format such as JSON or XML.

Data Protocol Examples

Example Sensor Data: "ID: AZ2355, Temperature: 25C, Time: 08:56, Type: K Thermocouple"

Custom Protocol

```
<AZ2355|25C|08:56|KTC>
```

Custom over HTTP

```
POST/sensorData/temperature.html HTTP/1.0User-Agent: HTTPTool/1.0Content-Type: application/x-www-form-urlencodedContent-Length: 23

<AZ2355|25C|08:56|KTC>
```

JSON

```
{
  "sensor": {
    "id": "AZ2355",
    "type": "kthermocouple"
  },
  "readings": [
    {
      "time": "08:56",
      "temperature": 25,
      "unit": "C"
    }
  ]
}
```

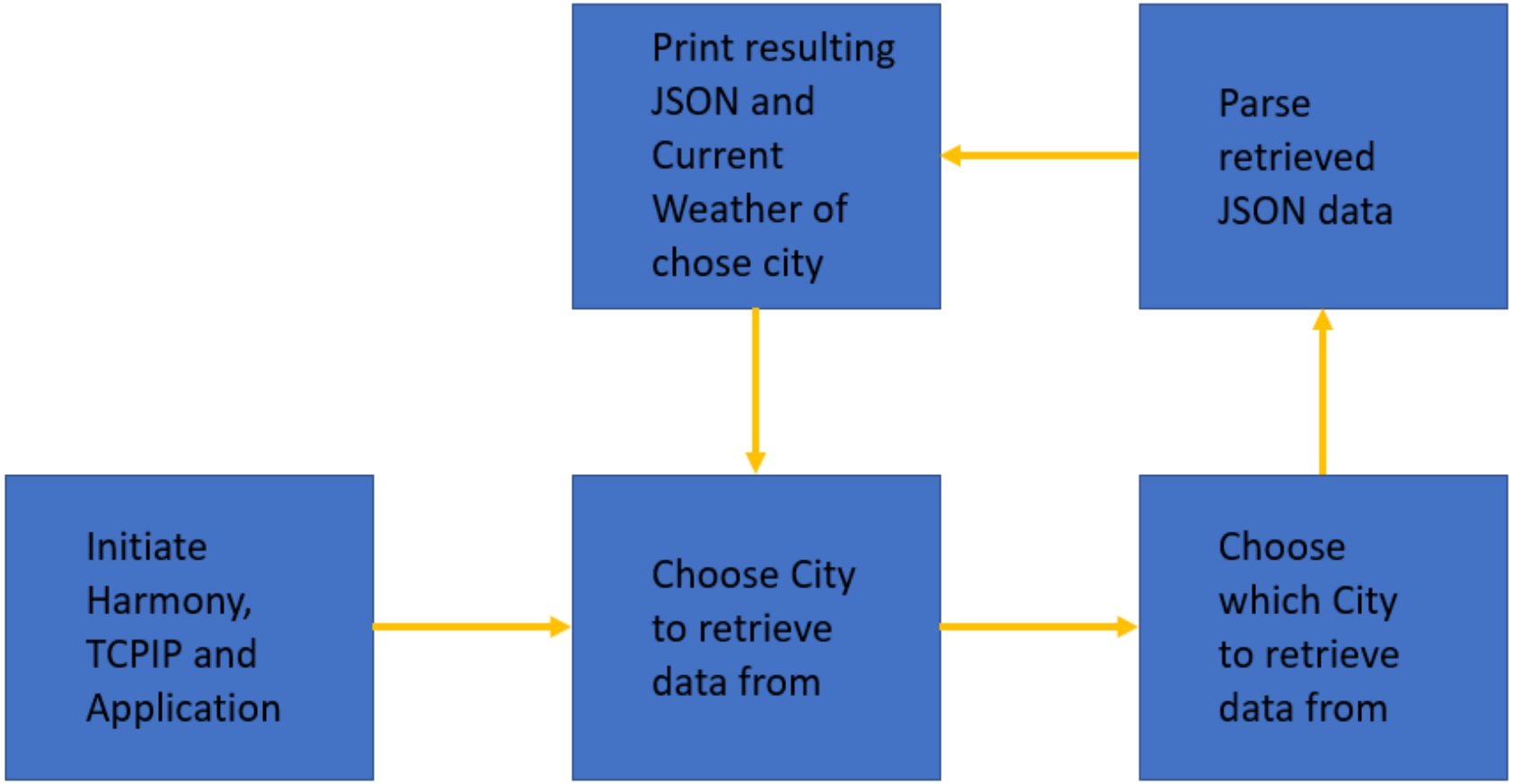
XML

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <readings>
      <element>

<temperature>25</temperature>

<time>08:56</time>
         <unit>C</unit>
      </element>
   </readings>
   <sensor>
      <id>az2355</id>

<type>kthermocouple</type>
   </sensor>
</root>
```

Now it is time for Lab3

Getting the Weather Report

Pulling the Weather Data



Pulling the Weather Data

martinruppert — minicom • sudo — 80×26

```
============================================================
request weather lab3 Jun 10 2019 12:27:31
MAC TCPIP_HOSTS_CONFIGURATION[0].macAddr: fc:c2:3d:0c:20:44
TCP/IP Stack: Initialization Started
TCP/IP Stack: Initialization Ended - success
    Interface GMAC on host MCHPBOARD_C     - NBNS disabled
GMAC IP Address: 192.168.0.19
Waiting for command type: requestWeather <city>


[>rw Phoenix                                                  ]


>cityBuffer: Phoenix
Starting connection

Connection Closed
resultingJson:
 {"coord":{"lon":-112.08,"lat":33.45},"weather":[{"id":801,"main":"Clouds","des

Current Weather in Phoenix
Humidity: 13
Pressure: 1016
Temperature: 34.22
Main Weather: Clouds
```

MICROCHIP MASTERs 2019

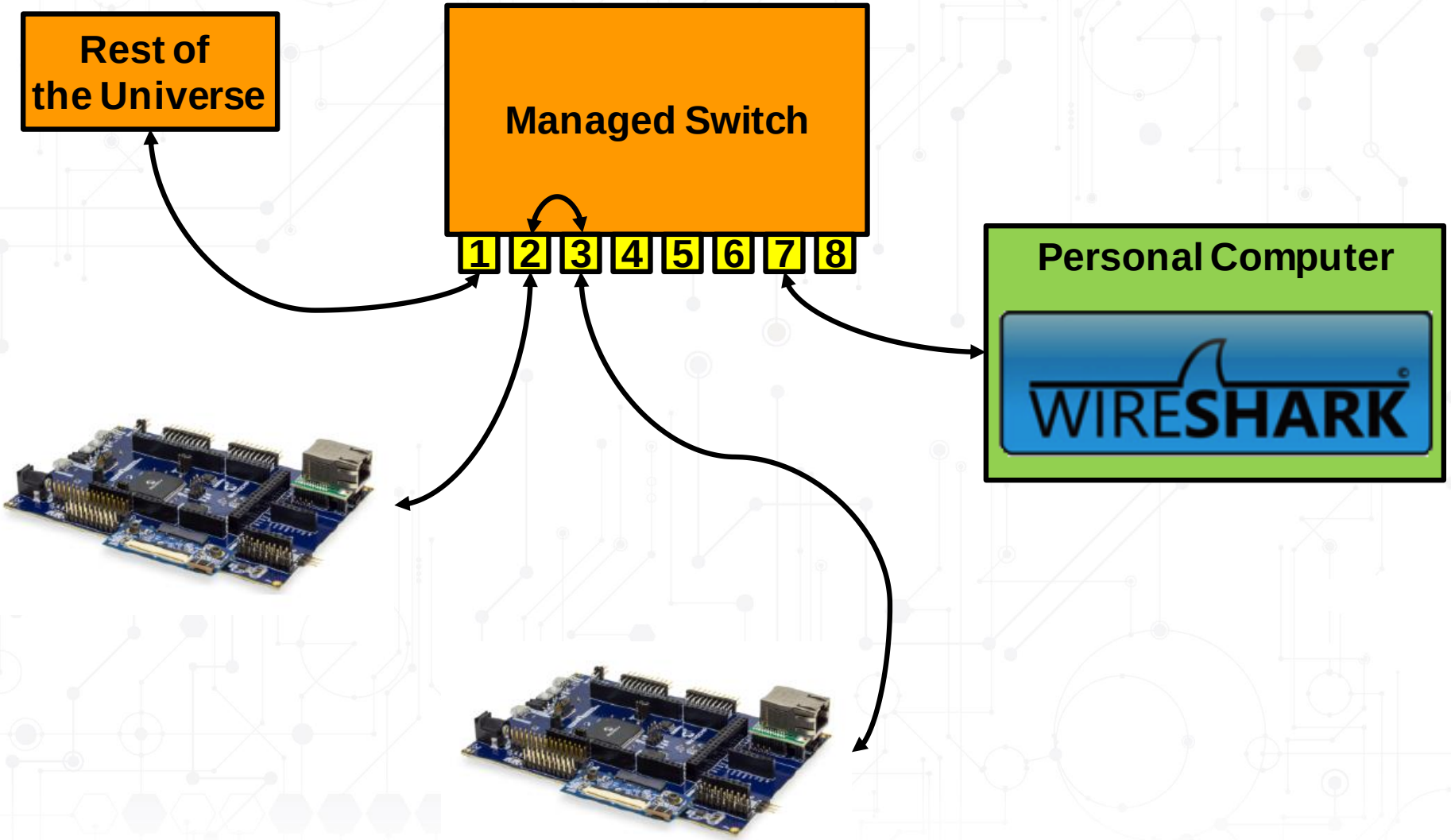
A TCP programmer without a Traffic Analyzer Tool is like a Electronic Developer without an Oscilloscope:

He or she is then Blind

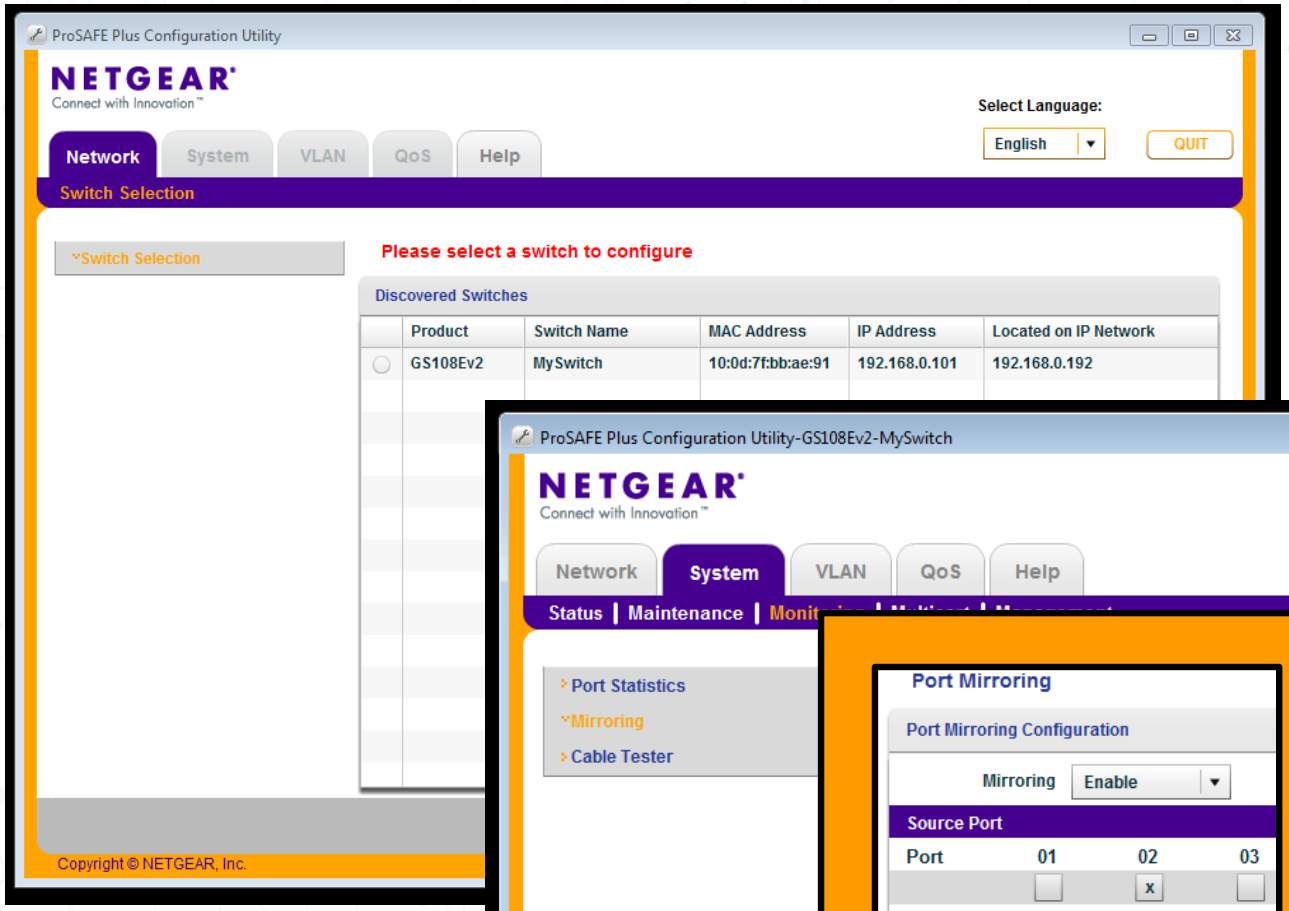
SMART | CONNECTED | SECURE

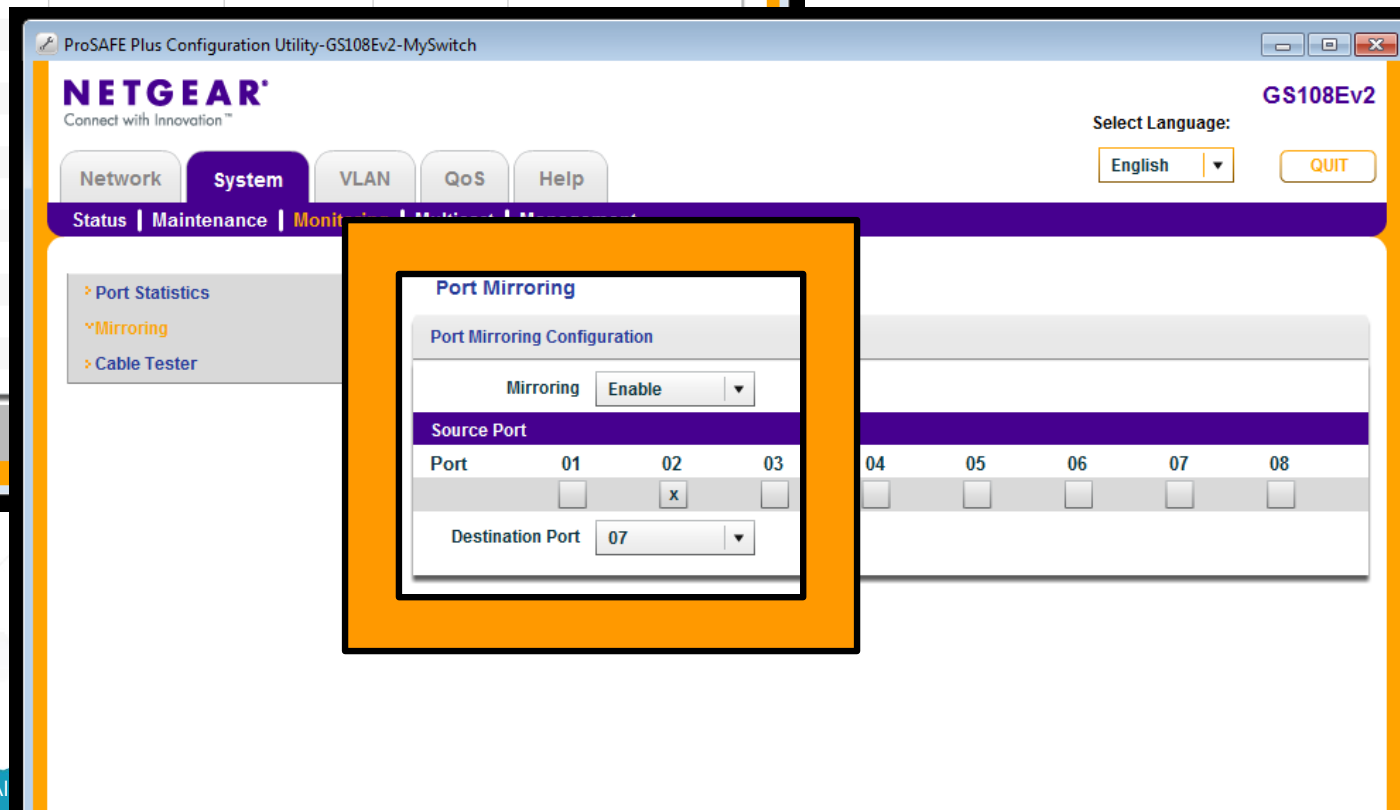
MICROCHIP

How to Analyze the Traffic

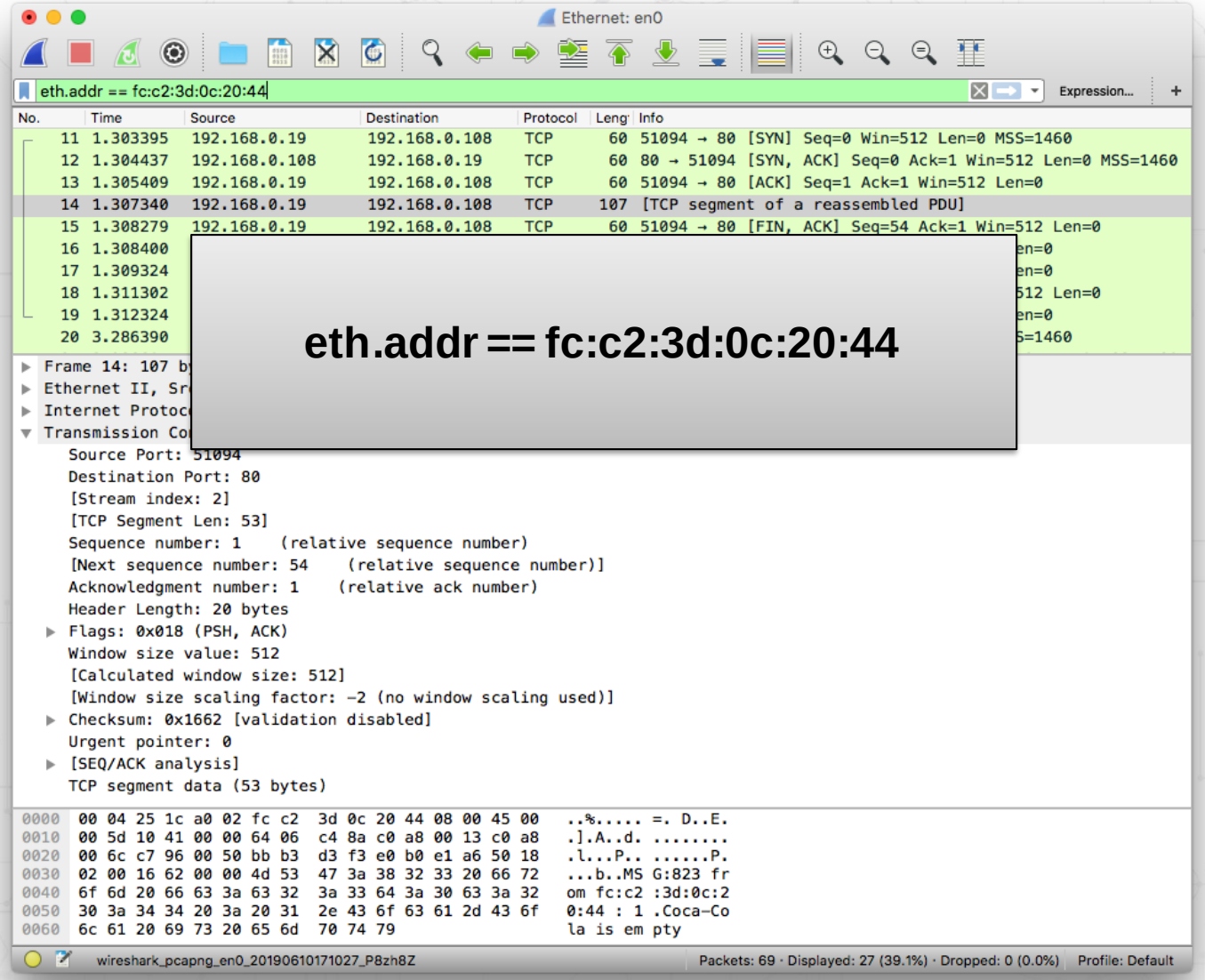


Configuration Software for Managed Switch

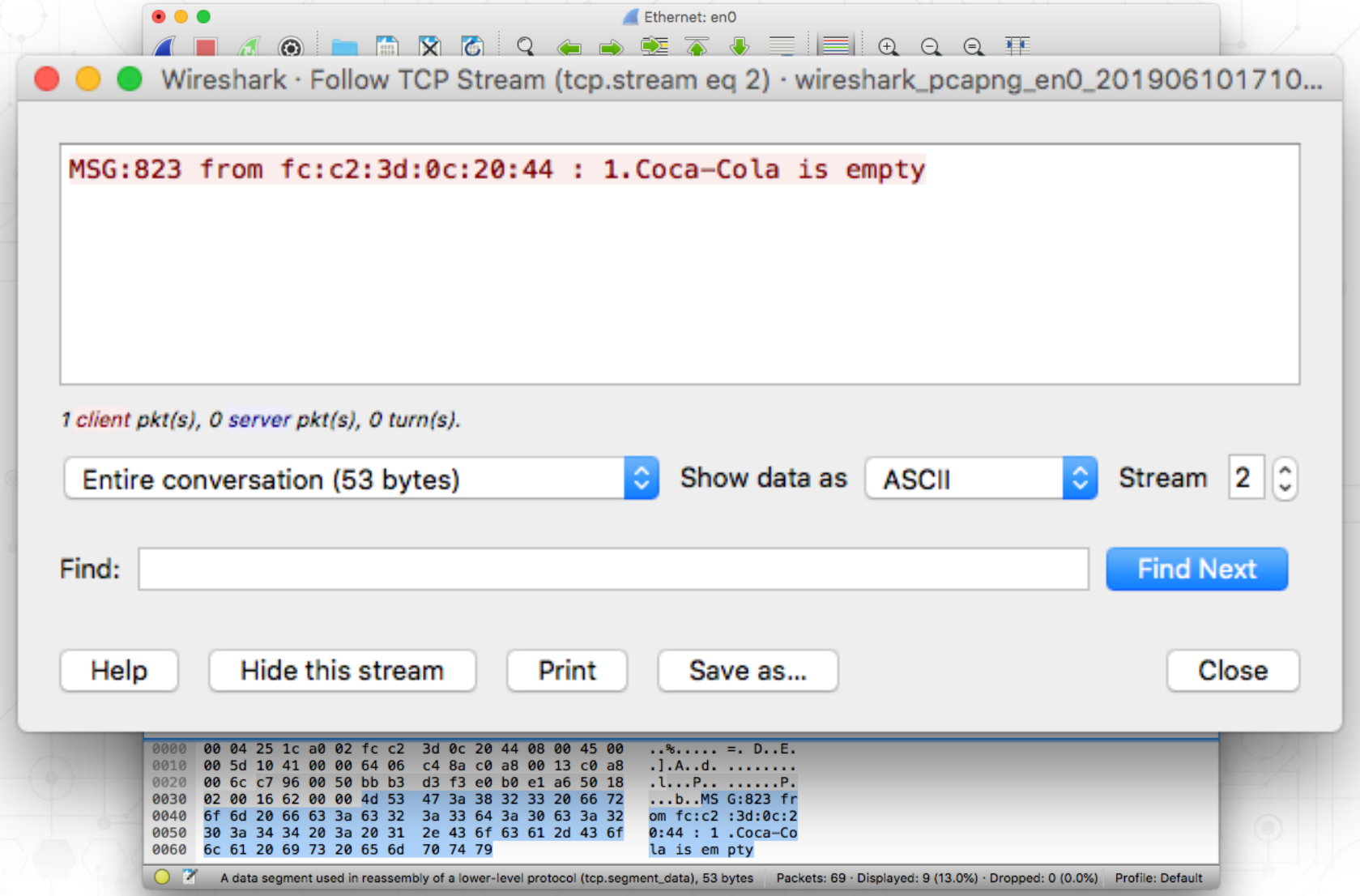




Wireshark Filter



Payload Data



Many thanks for attending this class

And...

...let us make this world a better place with TCPIP

MICROCHIP
MASTERs 2019

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, Adaptec, AnyRate, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, chipKIT, chipKIT logo, CryptoMemory, CryptoRF, dsPIC, FlashFlex, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PackeTime, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TempTrackr, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

APT, ClockWorks, The Embedded Control Solutions Company, EtherSynch, FlashTec, Hyper Speed Control, HyperLight Load, IntelliMOS, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, Temux, TimeCesium, TimeHub, TimePictra, TimeProvider, Vite, WinPath, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, EtherGREEN, In-Circuit Serial Programming, ICSP, INICnet, Inter-Chip Connectivity, JitterBlocker, KleerNet, KleerNet logo, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, SAM-ICE, Serial Quad I/O, SMART-I.S., SQI, SuperSwitcher, SuperSwitcher II, 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.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark 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.