MICROCHIP

TLS4102

# Getting started with HARMONY framework

*Alain SORIN*
*Principal Technical Training Eng..*

30/6/2015 V0.99c
**Harmony v1.05**

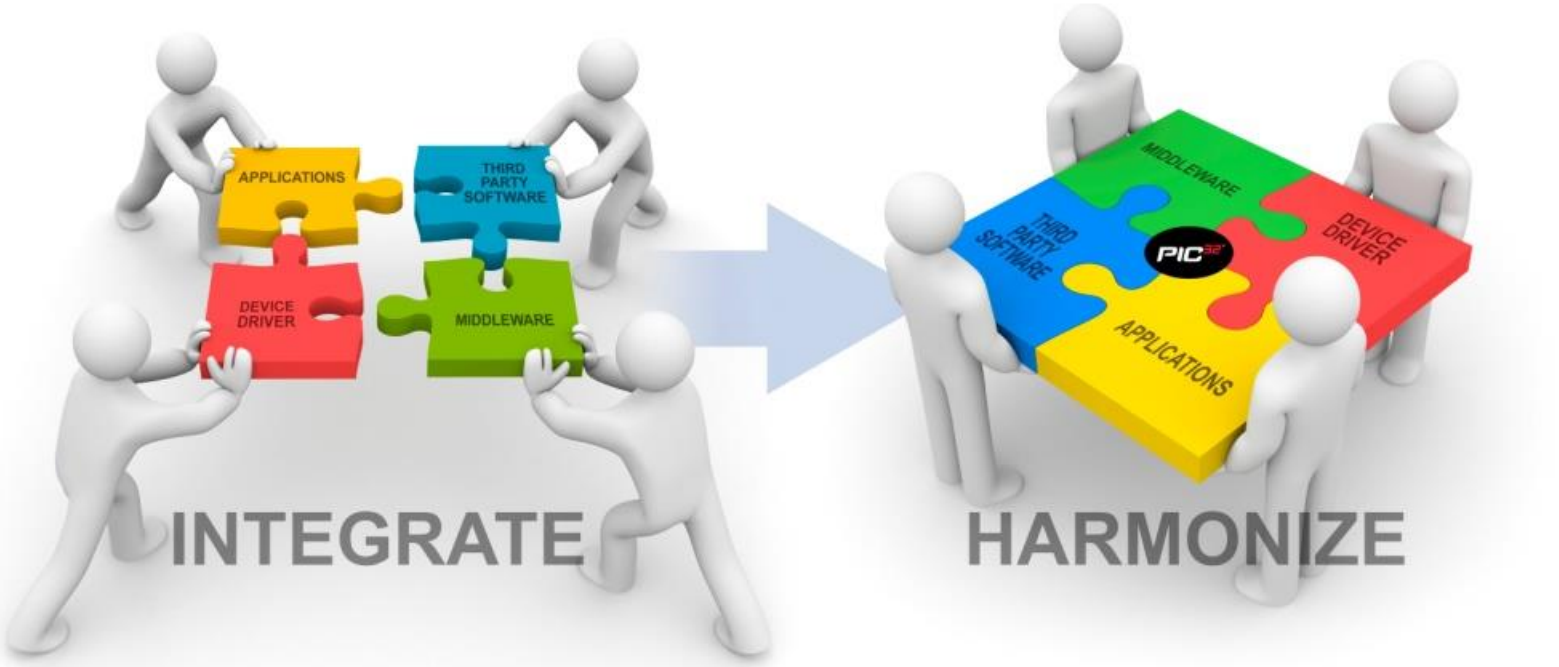
# Session Objectives

**In this session we will….**

- **Get a basic understanding of the MPLAB® Harmony PIC32 Firmware Development Framework and Benefits**
- **Create hands-on labs within MPLAB® X IDE using Harmony Framework and MPLAB® Harmony Configurator (MHC) tools**
- **Demonstration of a Harmony based RTOS application**
- **Learn about 3rd Party Partner MPLAB® Harmony Ecosystem solutions**

# Agenda


- Overview
- Getting Started + Hands-on Labs
- Adding Features/Functionality + Labs
- Application Migration + Lab
- Harmony RTOS Application + Demo
- Third Party Ecosystem Solutions
- Wrap-up, next steps




# MPLAB® Harmony Overview

# Key Benefits

- **Cross-Micro Compatibility**
  - Common, consistent APIs
  - Easier to grow into larger parts
  - Easier to shrink into smaller parts
- **Code Interoperability**
  - Drivers and libraries work together with minimal effort
  - Applications port to different boards with minimal effort
- **Faster Time to Market**
  - Able to develop applications more quickly
  - Able to easily add features
- **Microchip More Responsive to Customers**
  - Field & Apps teams able to develop applications more quickly
  - Reduced support burden frees resources for new development
- **Improved Satisfaction**
  - Eliminates conflicts
  - Improved code quality

# MPLAB® Harmony Middleware Components

MICROCHIP

MPLAB HARMONY
Integrated Software Framework

**Middleware**

- **HS**/FS/LS
- Multi-Instance
- Device
- Host
- **HUB**

*USB*

**Middleware**

- IPv4 / **IPv6**
- Transport/App Layers
- WiFi g

*TCP/IP*

**Plug-in Middleware**

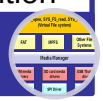
- Graphics Library
- MHGC : MPLAB Harmony Graphics Composer GUI
- Graphics Resource Converter (GRC)

*Graphics*

**Middleware**

- FAT32
- MPFS
- Multi-FileSystem
- Multi-Drive
- Multi-Partition

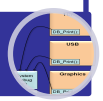
*File System*



**Common System Services**

- Timers
- Interrupts
- PORTS
- Clocks
- Misc
- …more

*System Services*



**Driver PLIB**

- Abstracted Interface
- Multi-Client
- Dynamic/Static
- Peripheral Access

*Drivers/ PLIB*

**Middleware**

- Cryptographic Library
- Compression and Hash Libraries
- Embedded SSL Library (CyaSSL)

*Embedded Security*

**Middleware**

- Real Time Engineers Ltd.
- Wittenstein
- InterNiche Tech.
- wolfSSL
- Segger.

*3rd Party Partners*

**RTOS OSAL**

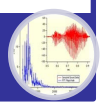
- FreeRTOS
- OPENRTOS
- uC/OS-II
- uC/OS-III
- **ThreadX**
- **embOS**

*OSAL/ RTOS*

**Middleware**

- 100+ functions DSP Library
- 29 functions Math Library

*DSP/ Math*



**Middleware**

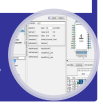
- Bluetooth Data Stack (certified)
- **Bluetooth Audio**

*Bluetooth*

**System Configuration**

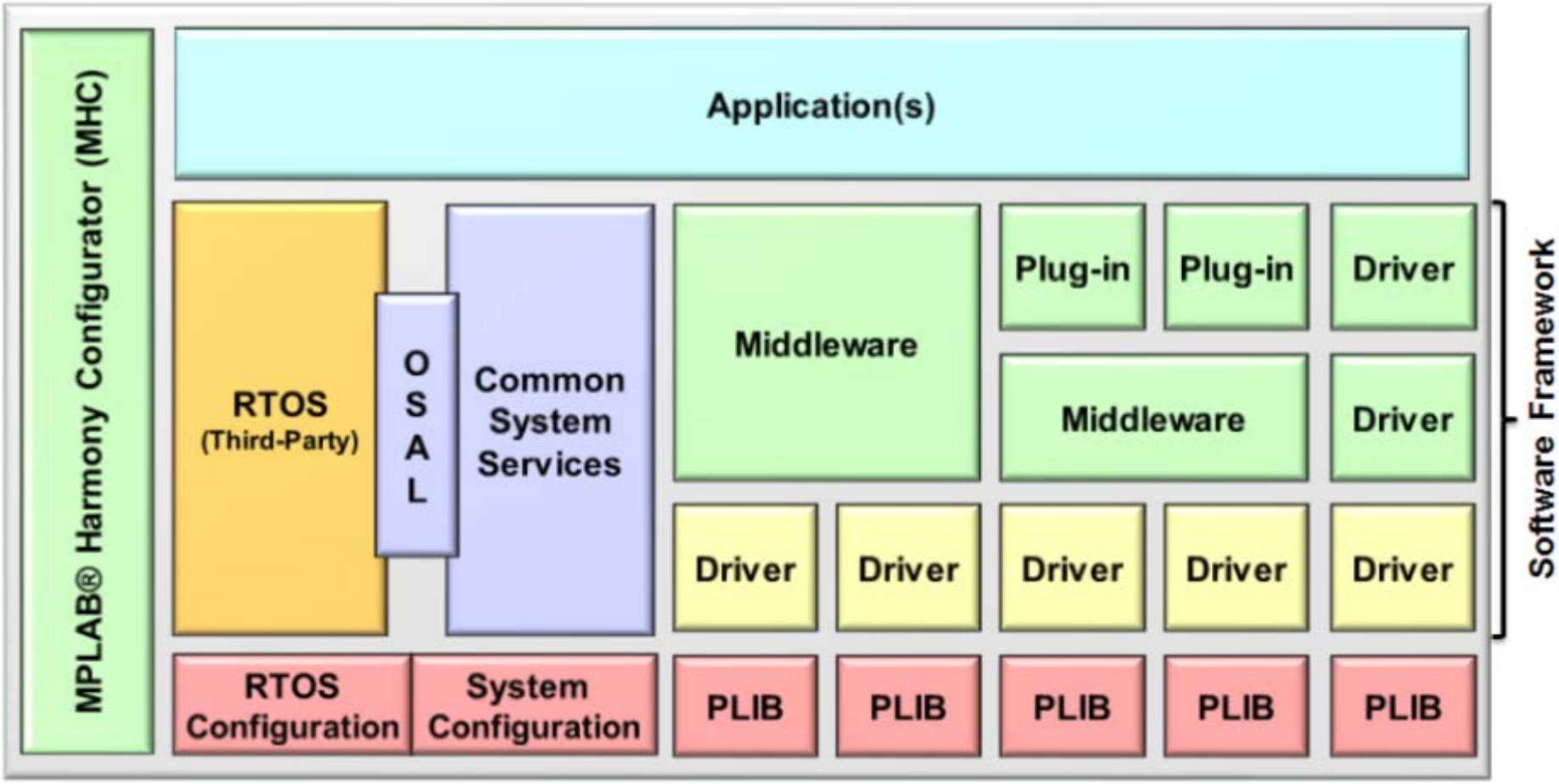
- MPLAB Harmony Configurator
-

*Harmony Configurator*



# Architecture

MPLAB HARMONY Integrated Software Framework

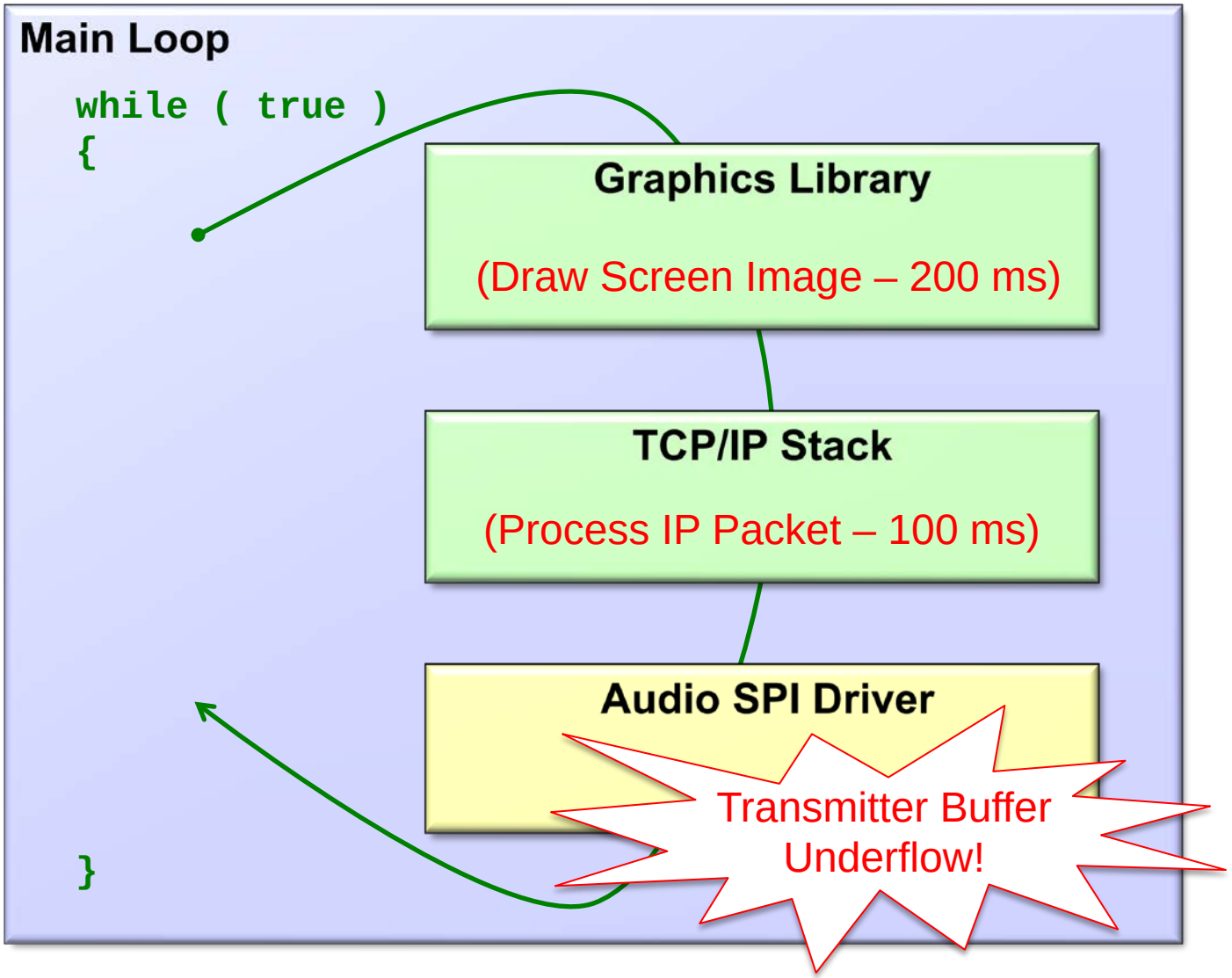


# What is a State Machine?

Refresher …..

- A Set of Input Events
- A Set of States
- A Set of Transitions
- An Initial State
- A function that maps input events to state transitions

# Why State Machines?



# System Configuration

MICROCHIP

MPLAB HARMONY Integrated Software Framework

## system_config.h

```
#define APP_TMR_HW_ID                   TMR_ID_3
#define SYS_CLK_FREQUENCY               0x0BEBC200
#define APP_LED_BLINKING_RATE           0x0002625A
```

## system_initialize.c

```
#pragma config FPLLODIV = DIV_1, FPLLMUL = MUL_20, FPLLIDIV = DIV_2, FWDTEN = OFF
#pragma config OSCIOFNC = ON, POSCMOD = HS, FSOSCEN = ON, FNOSC = PRIPLL
#pragma config ICESEL = ICS_PGx2

void SYS_Initialize( void )
{
    /* Call all library & application initialization routines */
}
```

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

    while(true)
    {
        SYS_Tasks();
    }

    return(EXIT_VALUE);
}
```

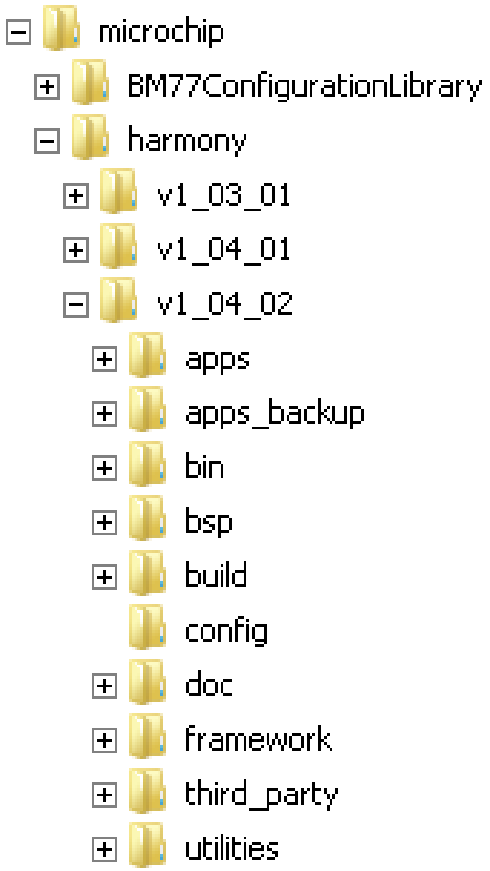
## system_tasks.c

```
void SYS_Tasks( void )
{
    /*Call all polled library & app "Tasks" routines*/
}
```

## system_interrupt.c

```
void __ISR ( _TIMER_3_VECTOR ) _ISR_TMR_3_stub ( void )
{
    /* Call the timer library's interrupt routine */
}
```

# Installation Folder



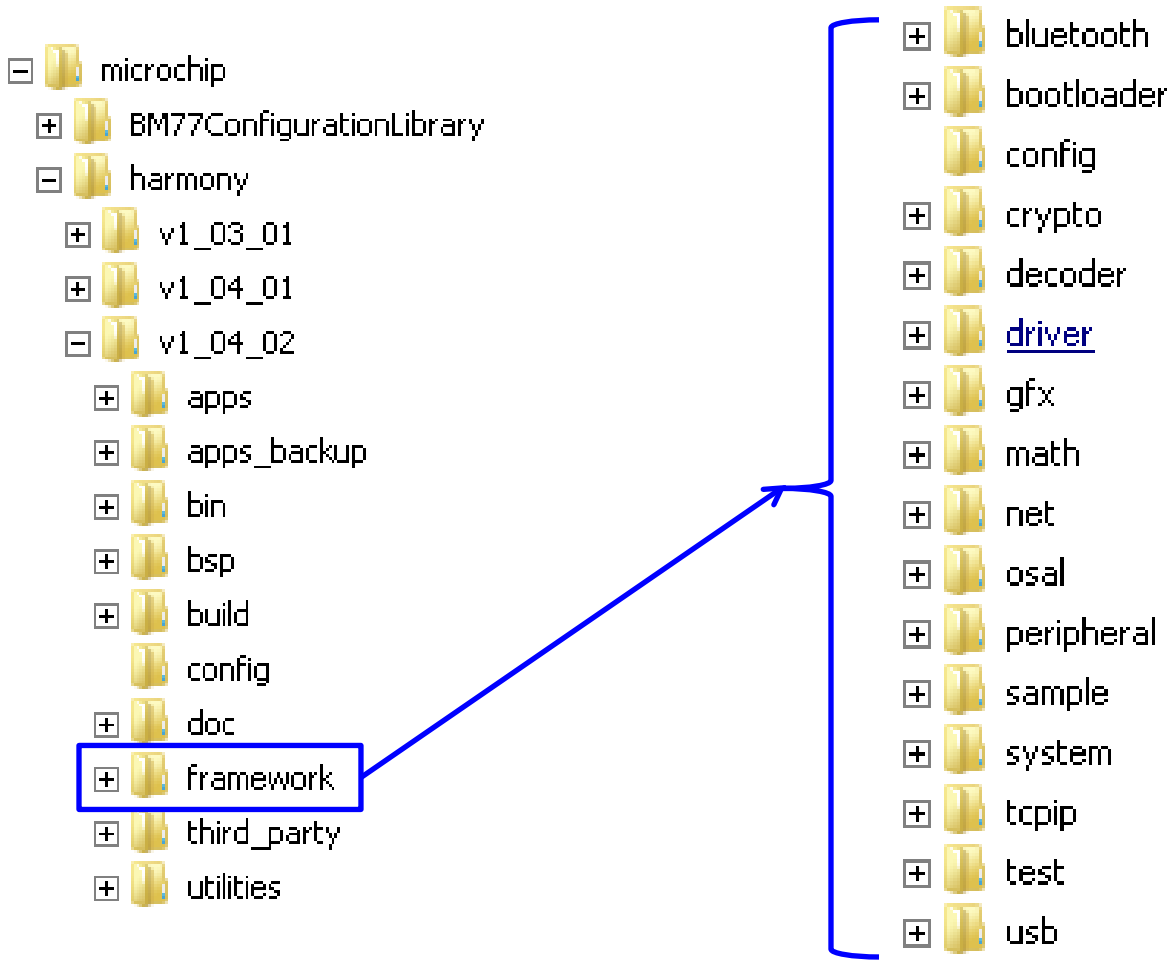

## Default Installation Directory

- Version Specific
- Windows

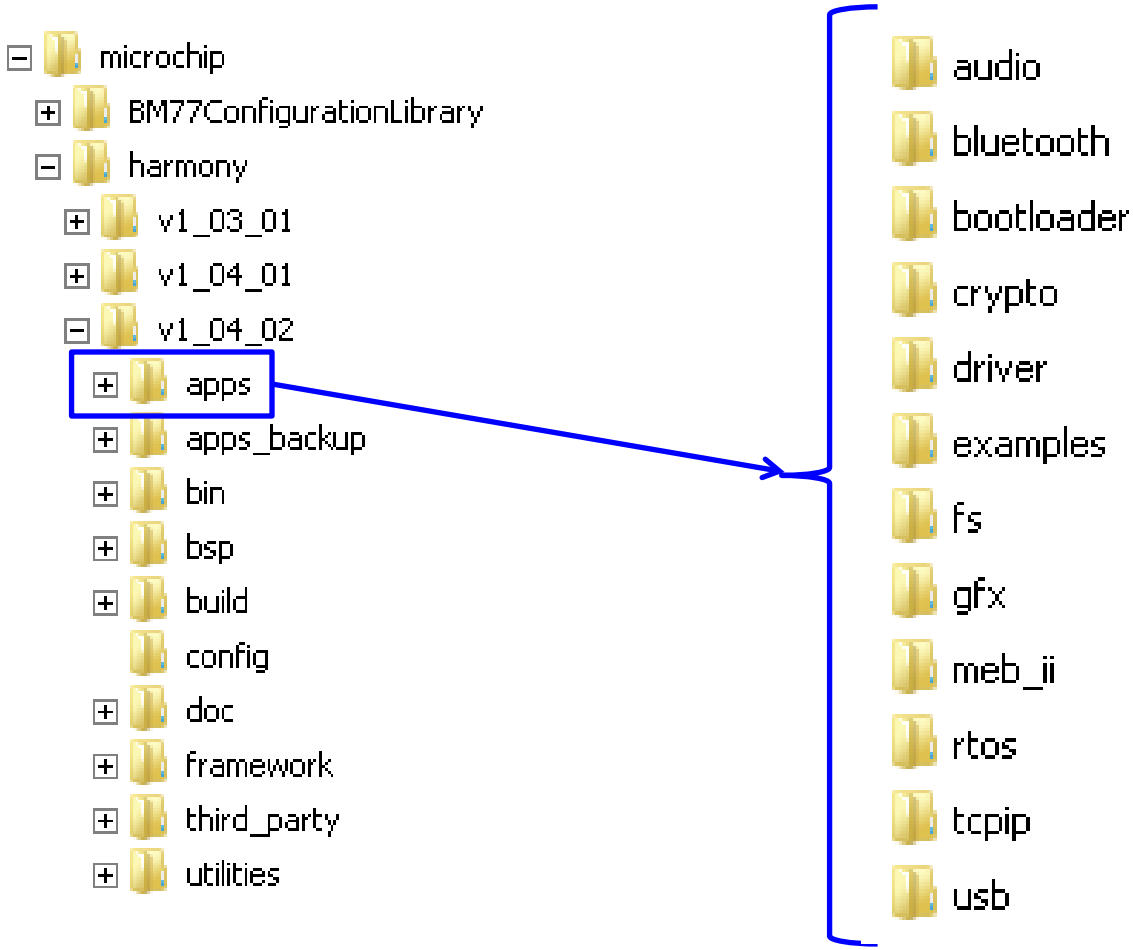
  C:\Microchip\harmony
- Mac/Linux

  ~/microchip/harmony
- Contains all collateral for a specific version of MPLAB® Harmony

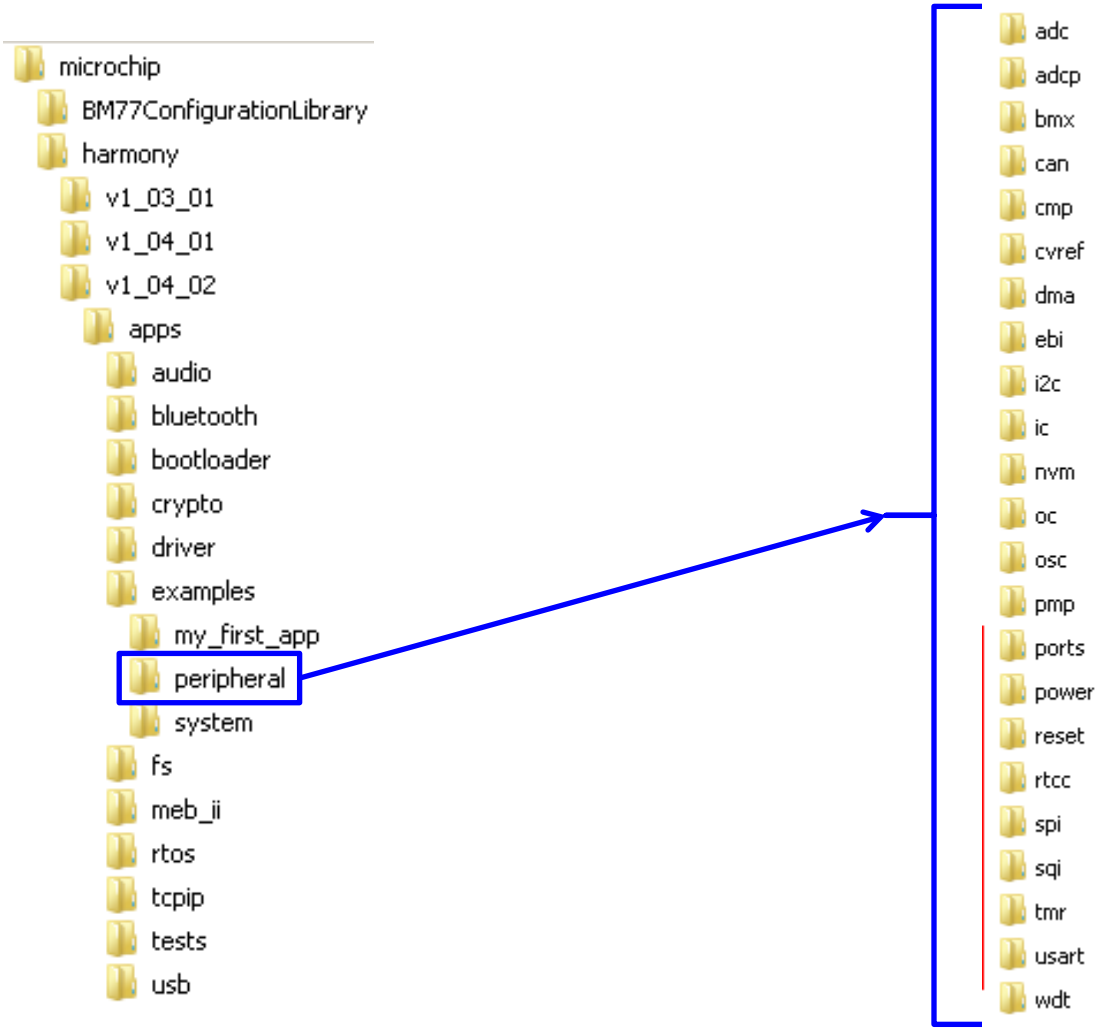
# Framework Libraries



# Demo Applications



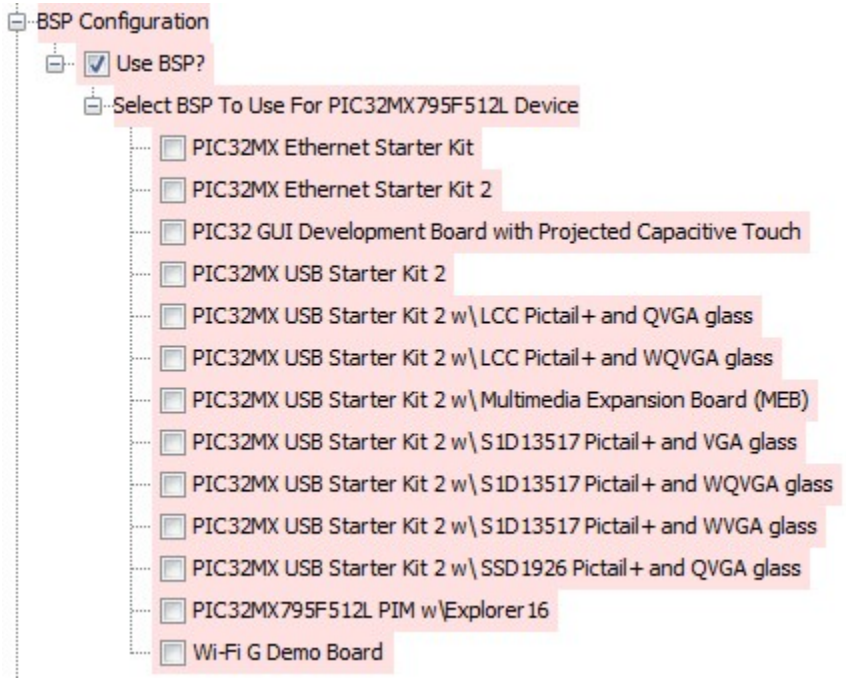
# Peripheral examples



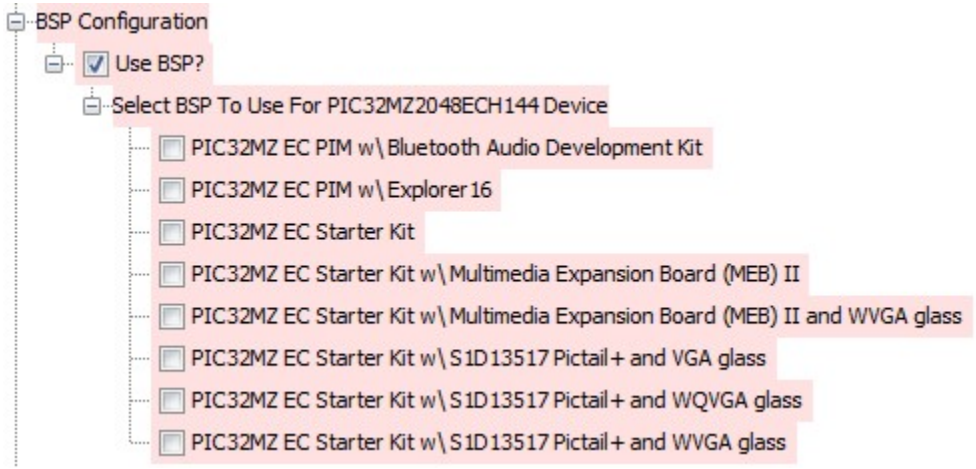
# Supported boards (BSP)

- **Most of Microchip PIC32 evaluation boards are supported in Harmony via Board Support Packages (drivers for external devices on these boards).**

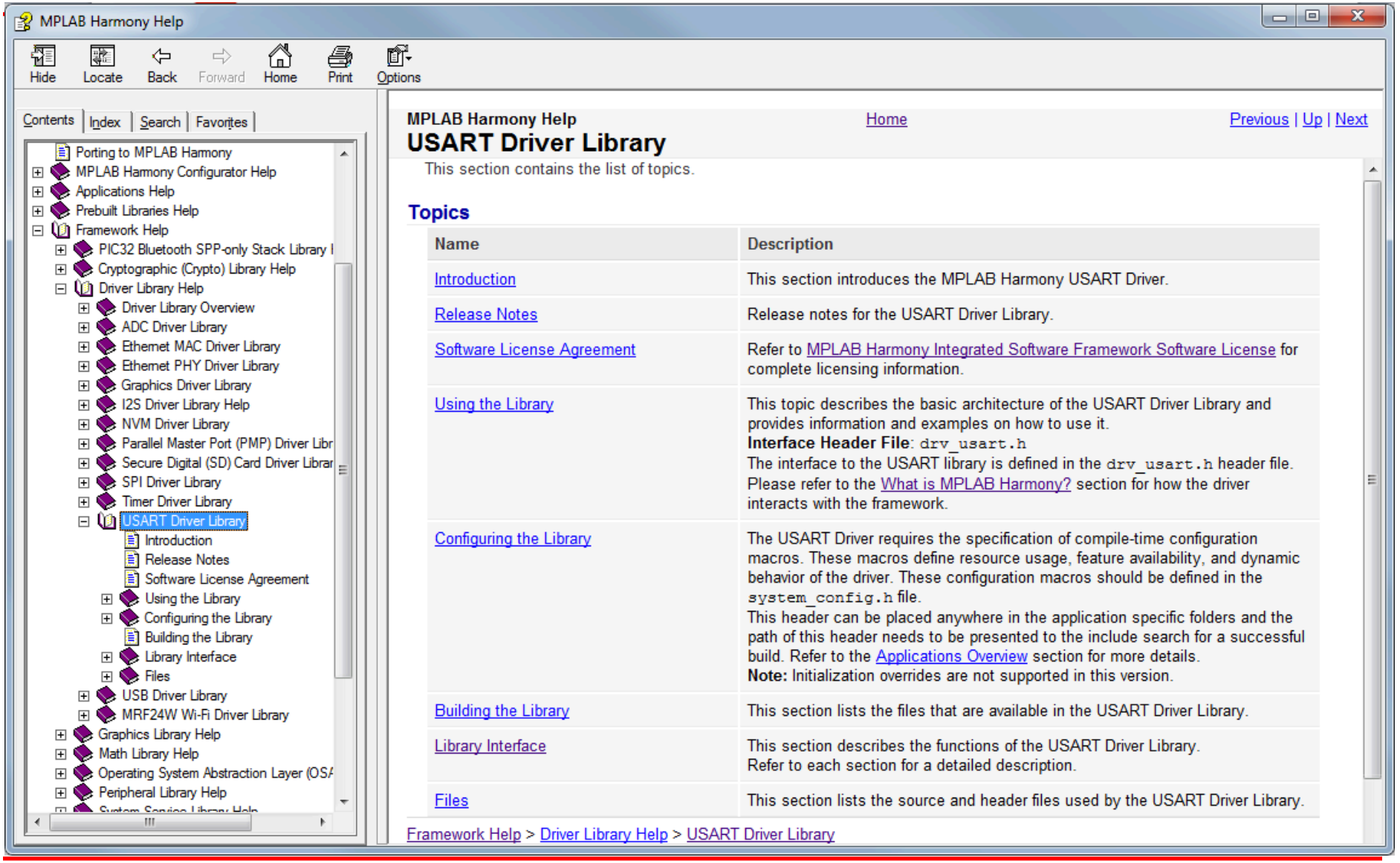
**PIC32MX795F512L**




**PIC32MZ2048ECH144**



# Library Help



# Ecosystem

MPLAB HARMONY Integrated Software Framework

- RTOS Support

µC/OS-II™ The Real-Time Kernel

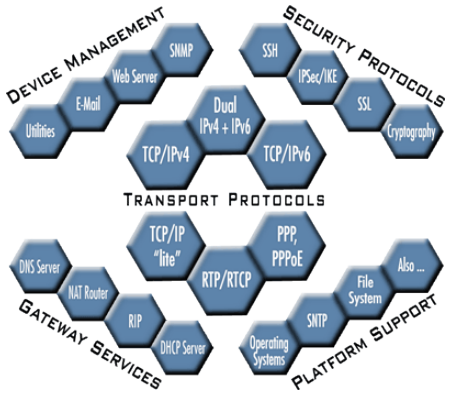
µC/OS-III™ The Real-Time Kernel

freeRTOS

OPENRTOS

- Third-Party Software

interniche technologies, inc.

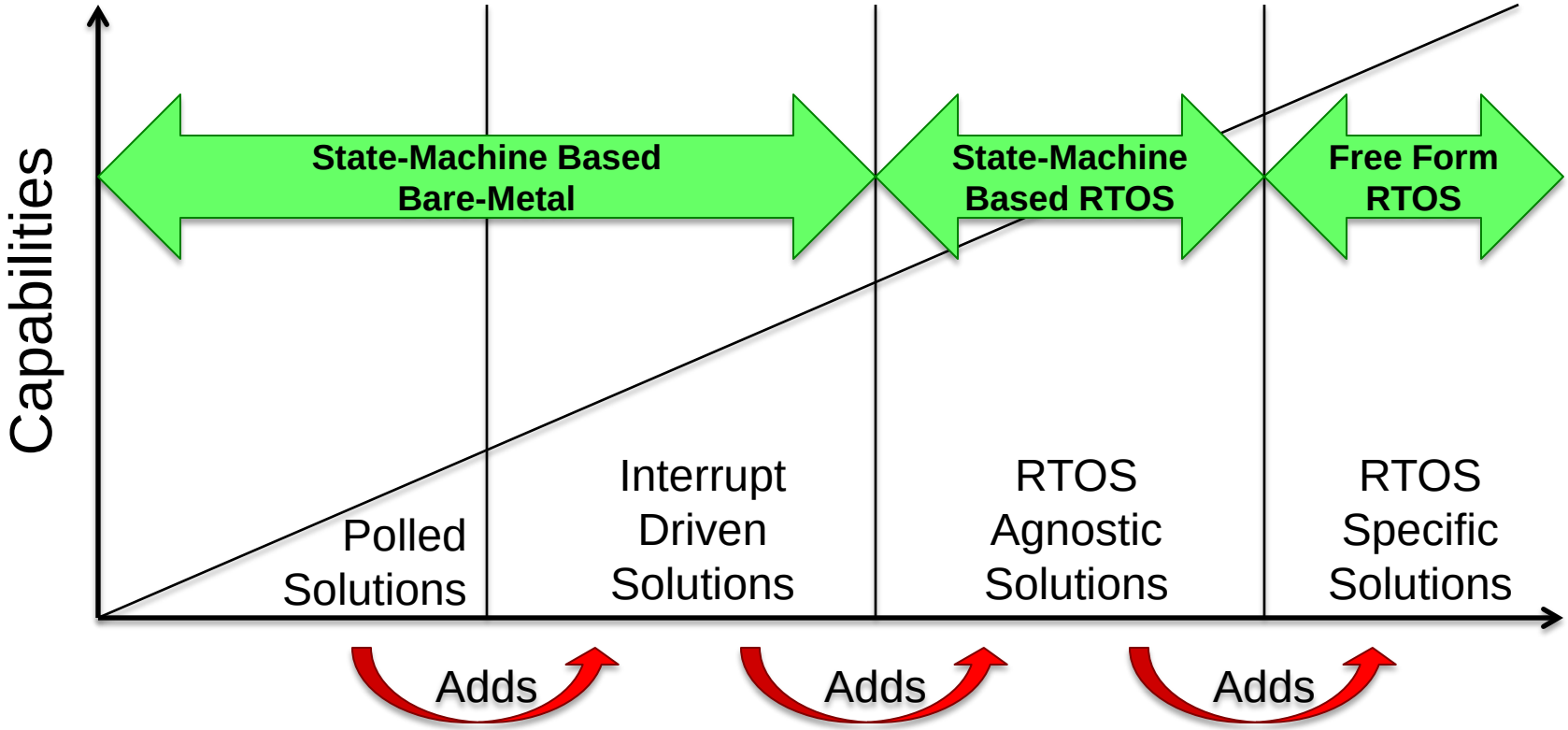



wolfSSL

# Agenda

- Overview
- **Getting Started + Hands-on Labs**
- Adding Features/Functionality + Labs
- Application Migration + Lab
- Harmony RTOS Application + Demo
- Third Party Ecosystem Solutions
- Wrap-up, next steps

# Harmony Flexibility

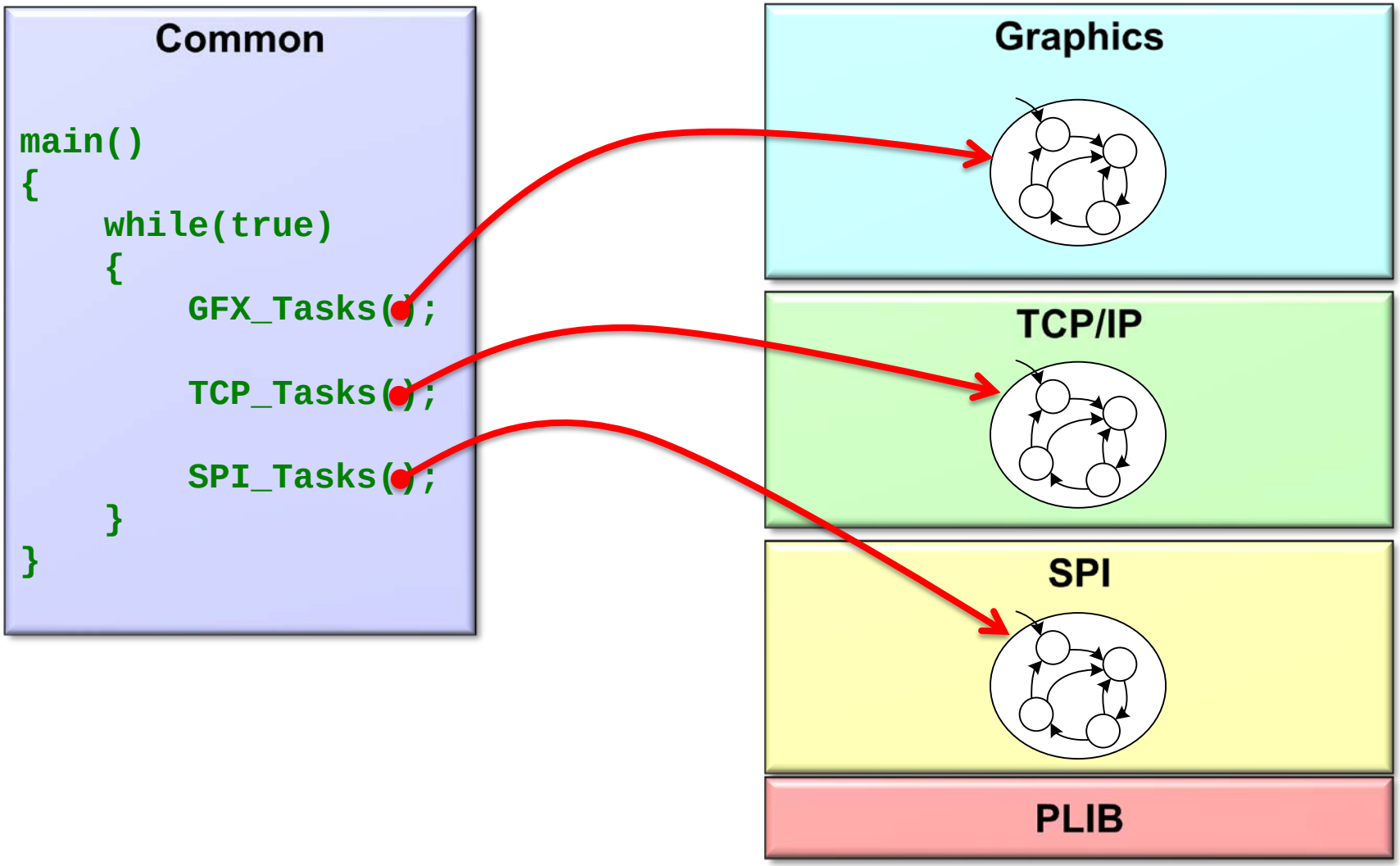



- **ISR Task Processing**
- **ISR Task Prioritization**

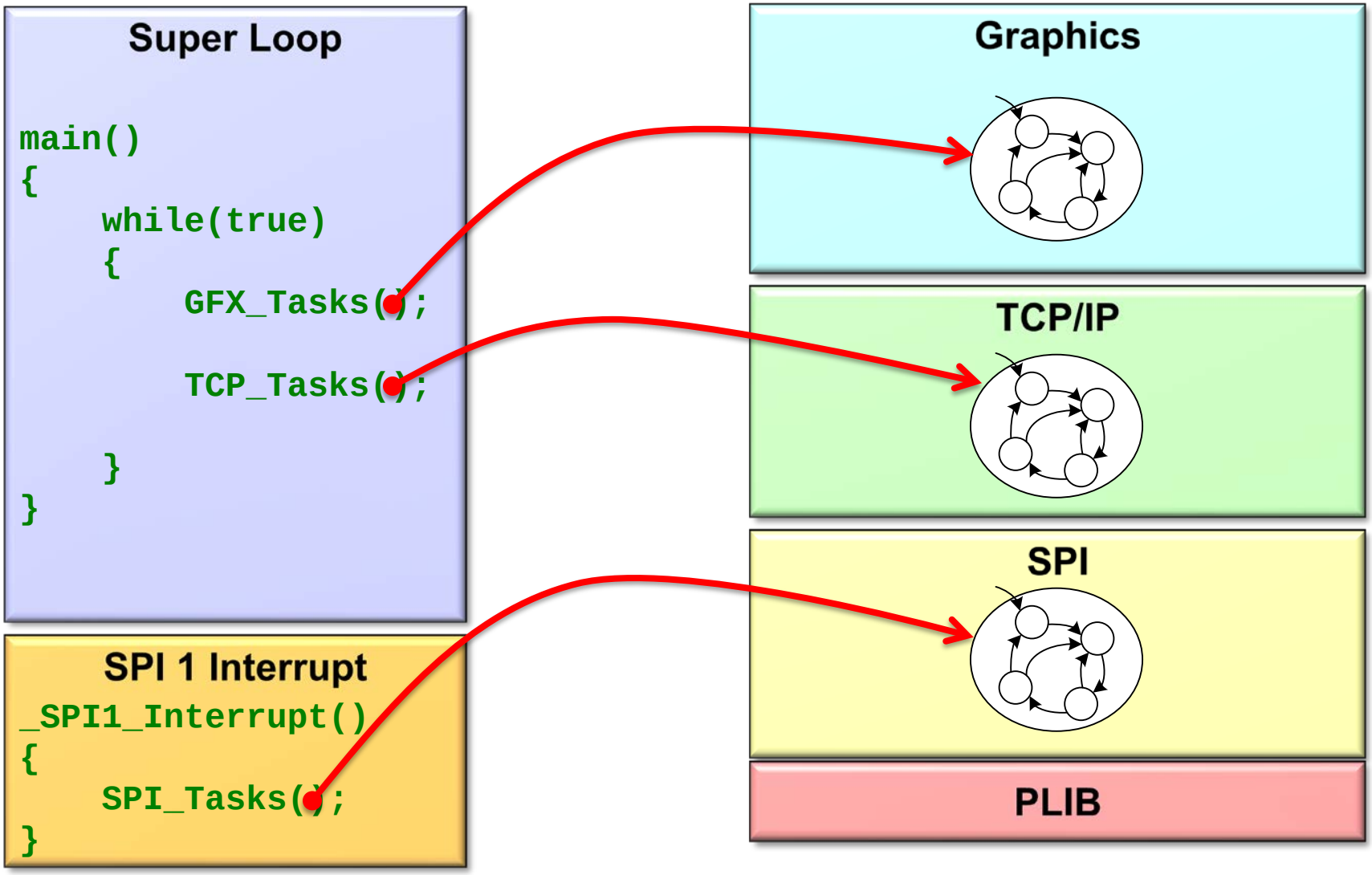
- **Multiple Polled Task Processing**
- **Polled Task Prioritization**

- **RTOS Specific Capabilities**
- **RTOS Programming Model**

# Super Loop – Polled



# Interrupt Driven



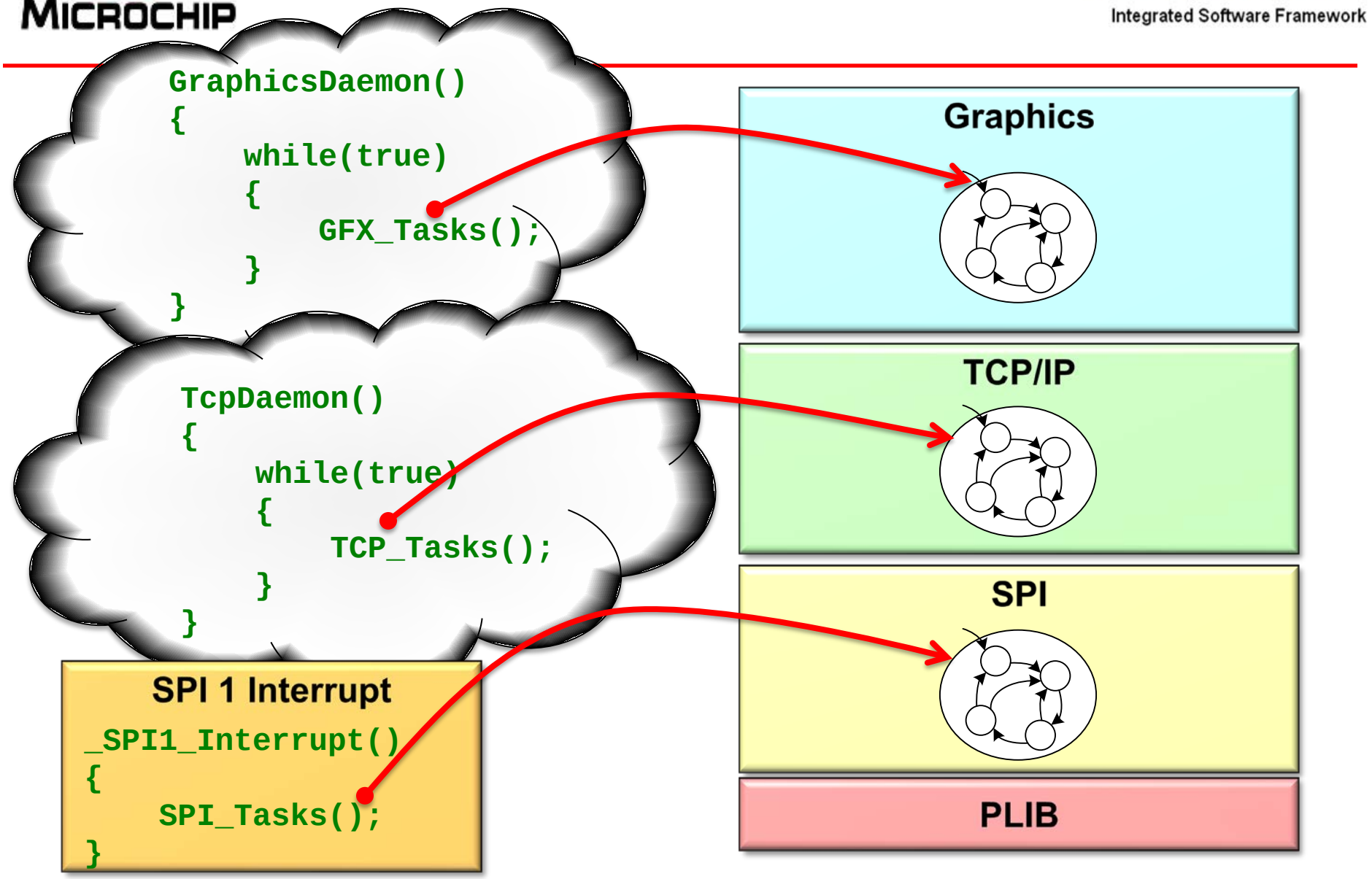
# Interrupt Driven

```
void __ISR ( _SPI_1_VECTOR ) _SPI1_Interrupt( void )
{
    /* Call the SPI driver's "Tasks" routine */
    SPI_Tasks( spi1Object.handle );

}
```

```
void SPI_Tasks( SYS_MODULE_OBJ handle )
{
    // Basic Logic:

    // Check the SPI 1 interrupt flag
    if ( /* The interrupt flag is set */ )
    {
        // Service the interrupt
        // Clear interrupt flag
        // Advance the driver's state machine
    }

    // Otherwise, do nothing.
}
```

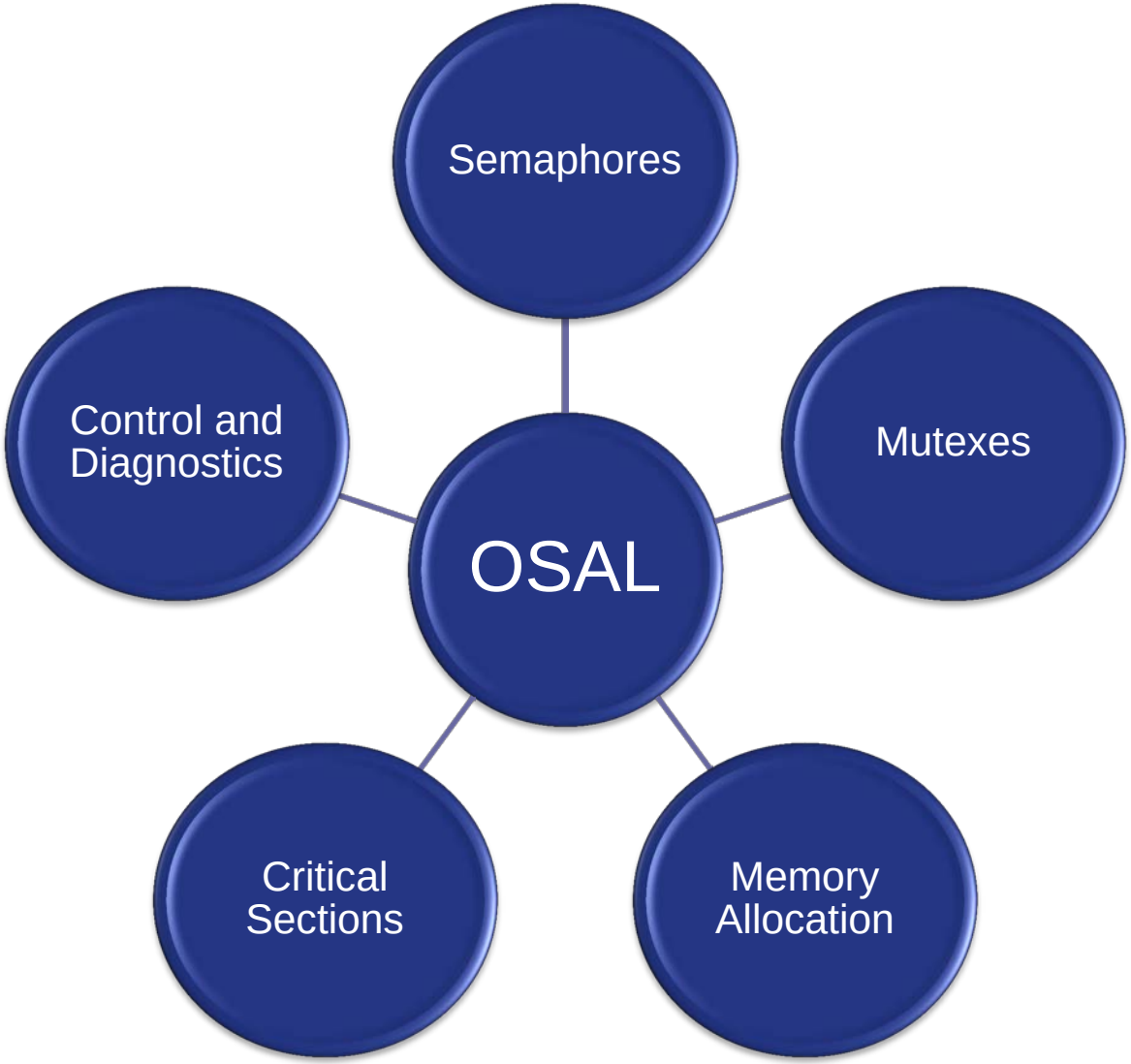
Avoids taking inappropriate action when flag not set. Safe for calling in polled loop.

# RTOS Driven

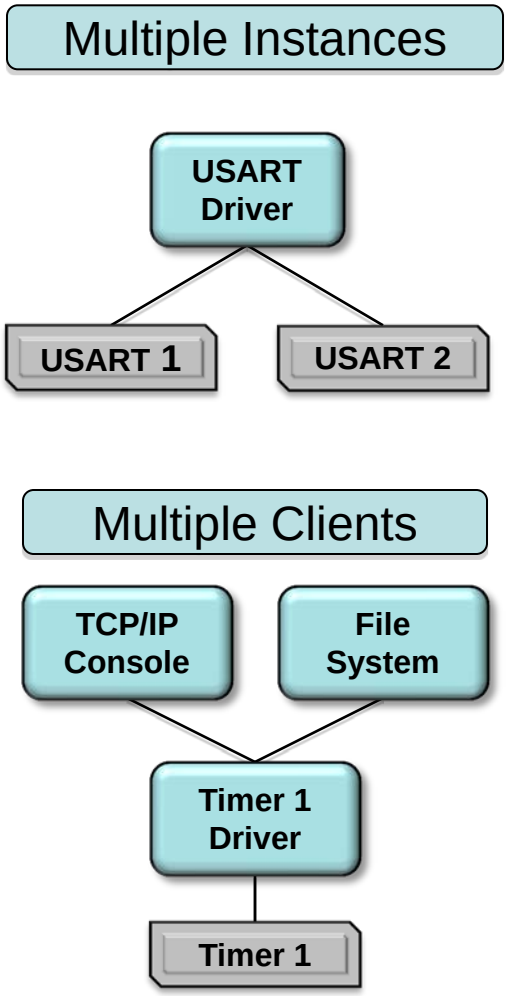

GraphicsDaemon()
{
while(true)
{
GFX_Tasks();
}
}
TcpDaemon()
{
while(true)
{
TCP_Tasks();
}
}
SPI 1 Interrupt
_SPI1_Interrupt()
{
SPI_Tasks();
}
Graphics
TCP/IP
SPI
PLIB

# OSAL Services

- **Multi-Thread Safety**
- **Multi-Thread Synchronization**
- **RTOS Compatibility**
- **OS Agnostic or “Bare Metal"**
- **Light-weight**



# Challenges of Cooperating Modules



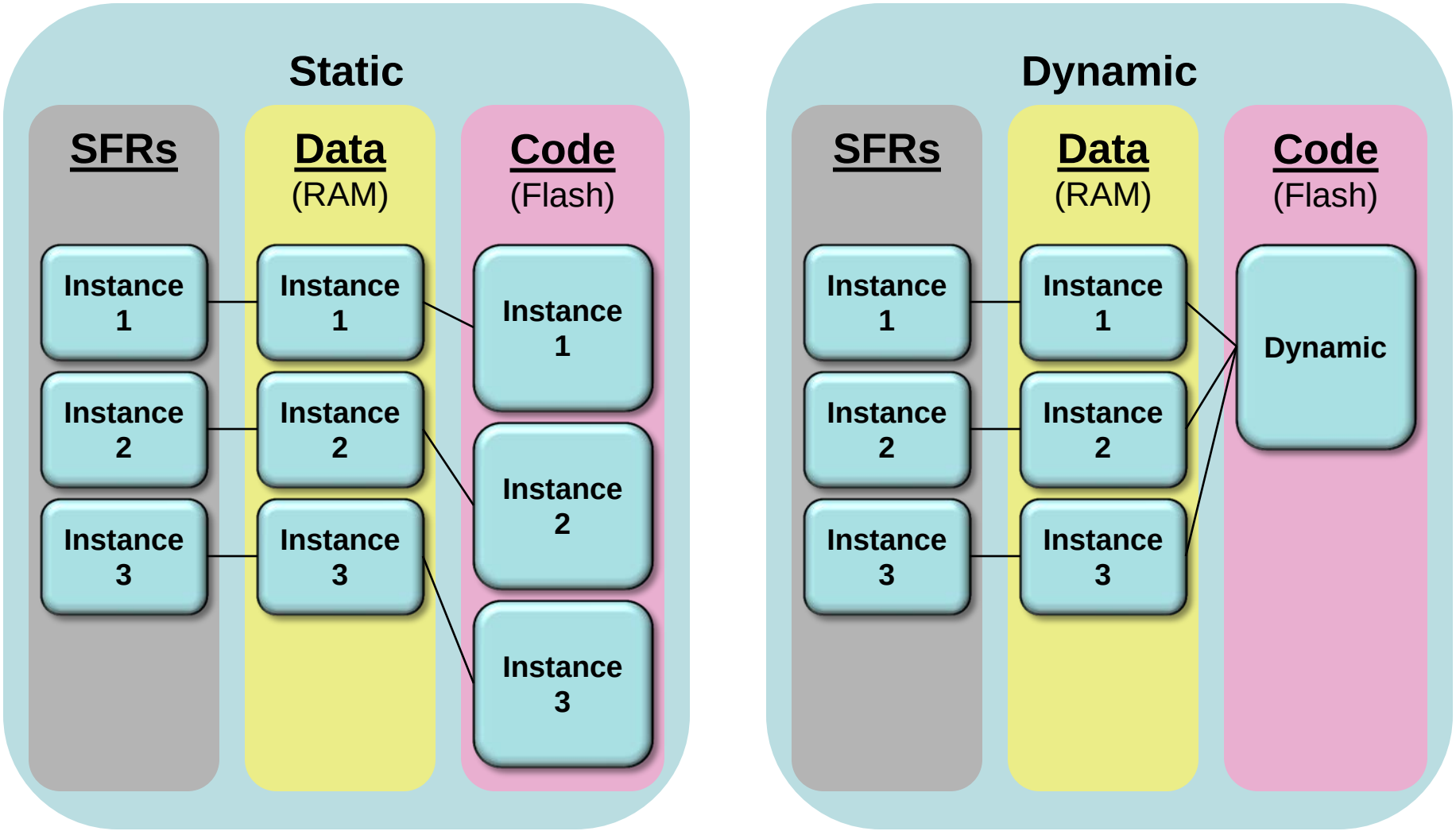

Some modules need to manage multiple instances of same hardware.

- How do we avoid duplicating driver code unnecessarily?
- How can one driver manage multiple instances of a peripheral?

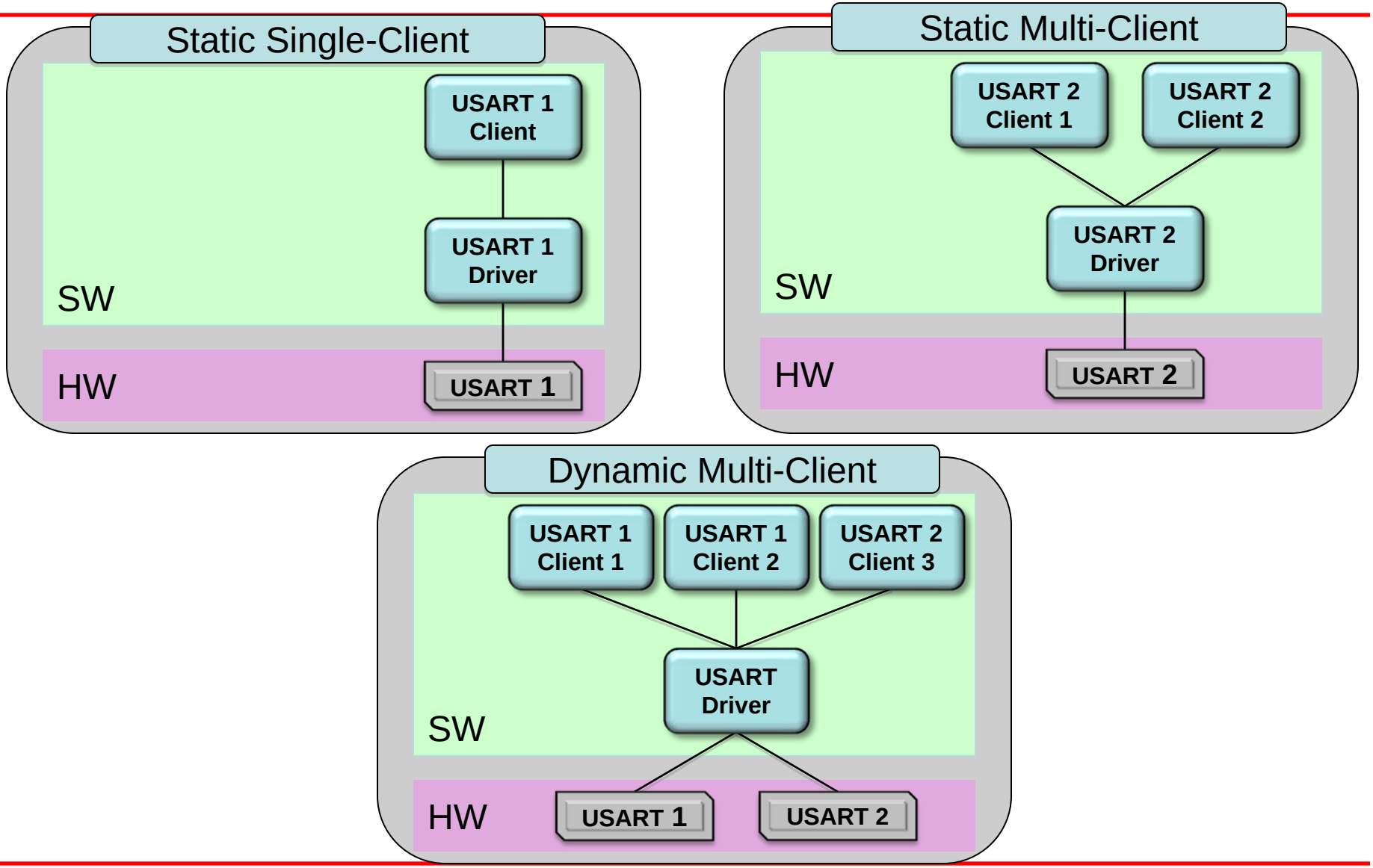
Unrelated “client” modules access same “server” module.

- How does server module keep requests separate?
- How does server module manage hardware correctly?

# Static vs. Dynamic



# Single- or Multi-Client



# Static vs. Dynamic

- We will use Static Drivers for the hands-on Labs
- We will see Dynamic Drivers in the RTOS multi-stack application demo

# Labs 1a and 1b

# Getting Started in MPLAB® Harmony

**(note: make sure to use Harmony v1.05 directory for all labs )**

# Class Introduction

- **In the labs you will be requested to follow specific and certain steps to configure the PIC32 MCU. Due to the time allotted for each lab the details as to why each configuration is requested is not provided. Upon completion of the class labs you are encouraged to further your knowledge and understanding on PIC32 setup and configuration options.**

- **Note: This class is designed to support both PIC32MZ EC Starter Kits:**
  - PIC32MZ Embedded Connectivity Starter Kit
    part # DM320006 (PIC32MZ2048ECH144 MCU)
  - PIC32MZ Embedded Connectivity Starter Kit w/Crypto Engine
    part # DM320006-C, (PIC32MZ2048ECM144 MCU)

# Class Prerequisites

- **The class is not designed to teach:**
  - MPLAB® X IDE fundamentals
  - MPLAB based Programming/Debugging fundamentals
  - ‘C’ language programming
  - In depth knowledge on PIC32 Product Family
  - In depth knowledge on MPLAB® Harmony Framework and it’s components
- **The class lab material assumes you have prior hands-on experience with the first three items and some basic knowledge on PIC32 32-bit MCUs.**
- **<u>Upon completion of this class you will</u>:**
  - Gain a basic understanding how to create an MPLAB® X IDE Harmony project
  - Gain a basic understanding and foundation on MPLAB® Harmony Framework
  - Gain hands-on experience with MPLAB® Harmony Framework components
  - Gain hands-on experience with MPLAB® Harmony Configurator (MHC) Tool
  - Gain overall understanding on key benefits and features of MPLAB® Harmony

# Let’s take a quick look at the PIC32MZ Hardware we will use

# PIC32MZ EC Starter Kit

- PIC32MZ2048ECH144
- Integrated debugger/ programmer or mini ICSP
- USB Powered
- 10/100 Ethernet on socket
- High Speed USB host, device, Dual Role and OTG
- 3 LEDs + 3 Push Buttons

Part#: DM320006

Part#: AC320101

- MEB/MEB II UART to USB Adapter

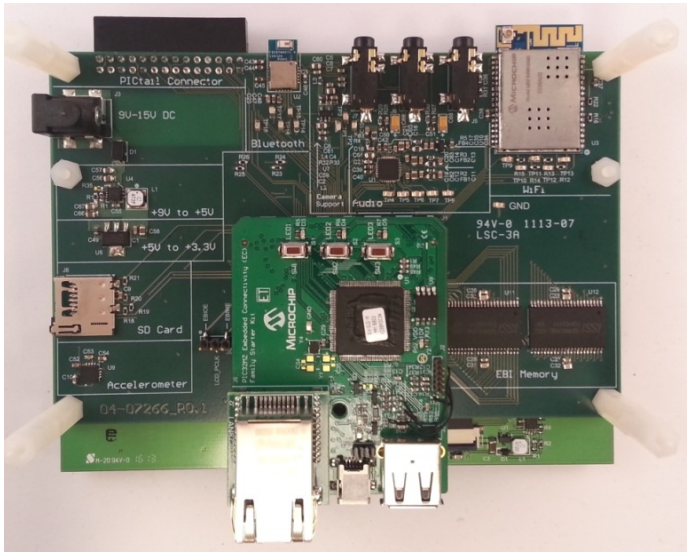
# Multimedia Expansion Brd. II

## Highly integrated, flexible development platform

- Controllerless graphics drive
- 4.3” WQVGA display
- Multi-touch PCAP using MTCH6301 (capacitive)
- VGA camera
- Wi-Fi (**MRF24WG0MA)**
- Bluetooth
- 24-bit stereo audio
- 3-axis accelerometer
- Temperature sensor

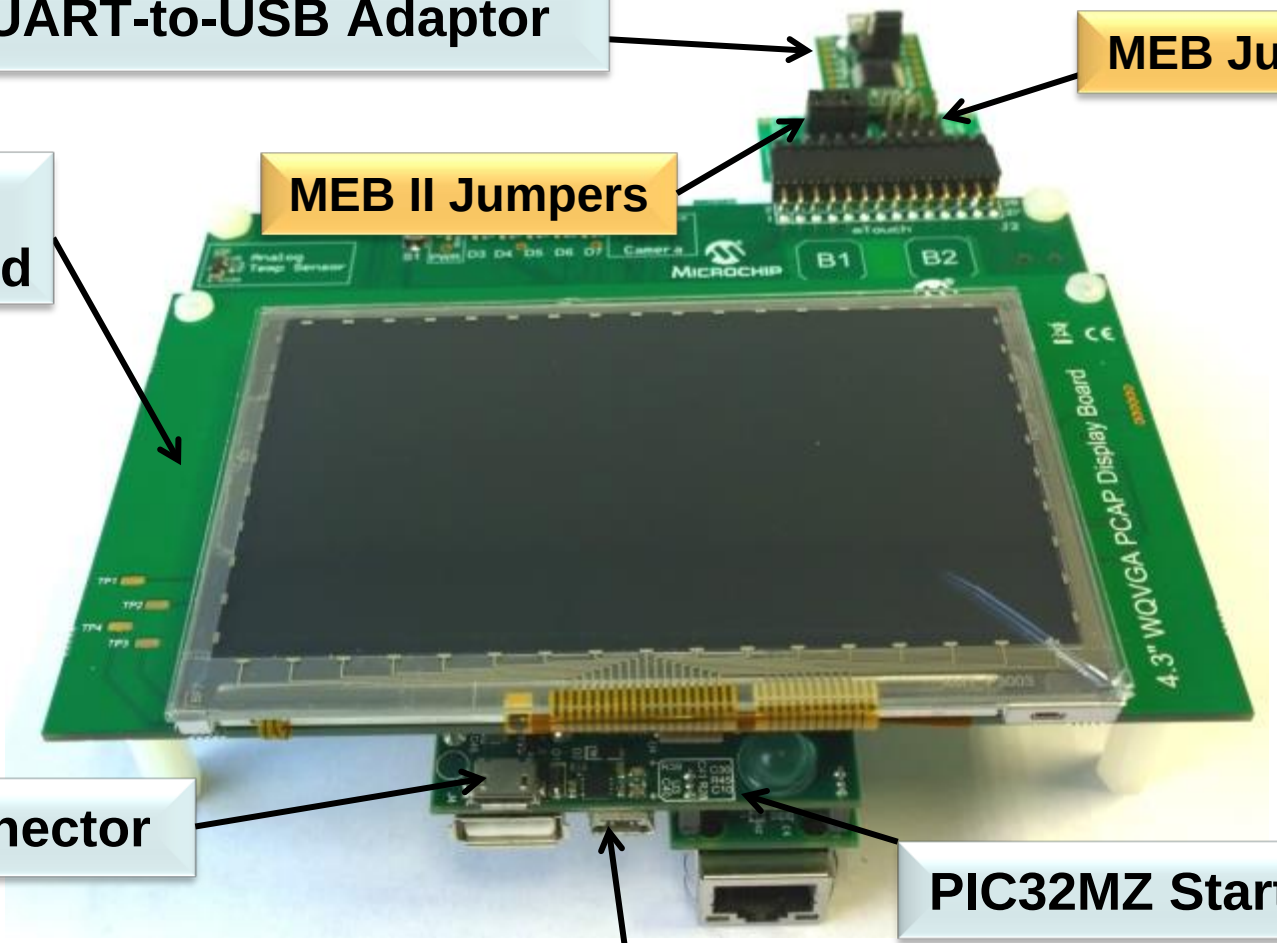
**Front View**

**Rear View**




Part#: DM320005-2

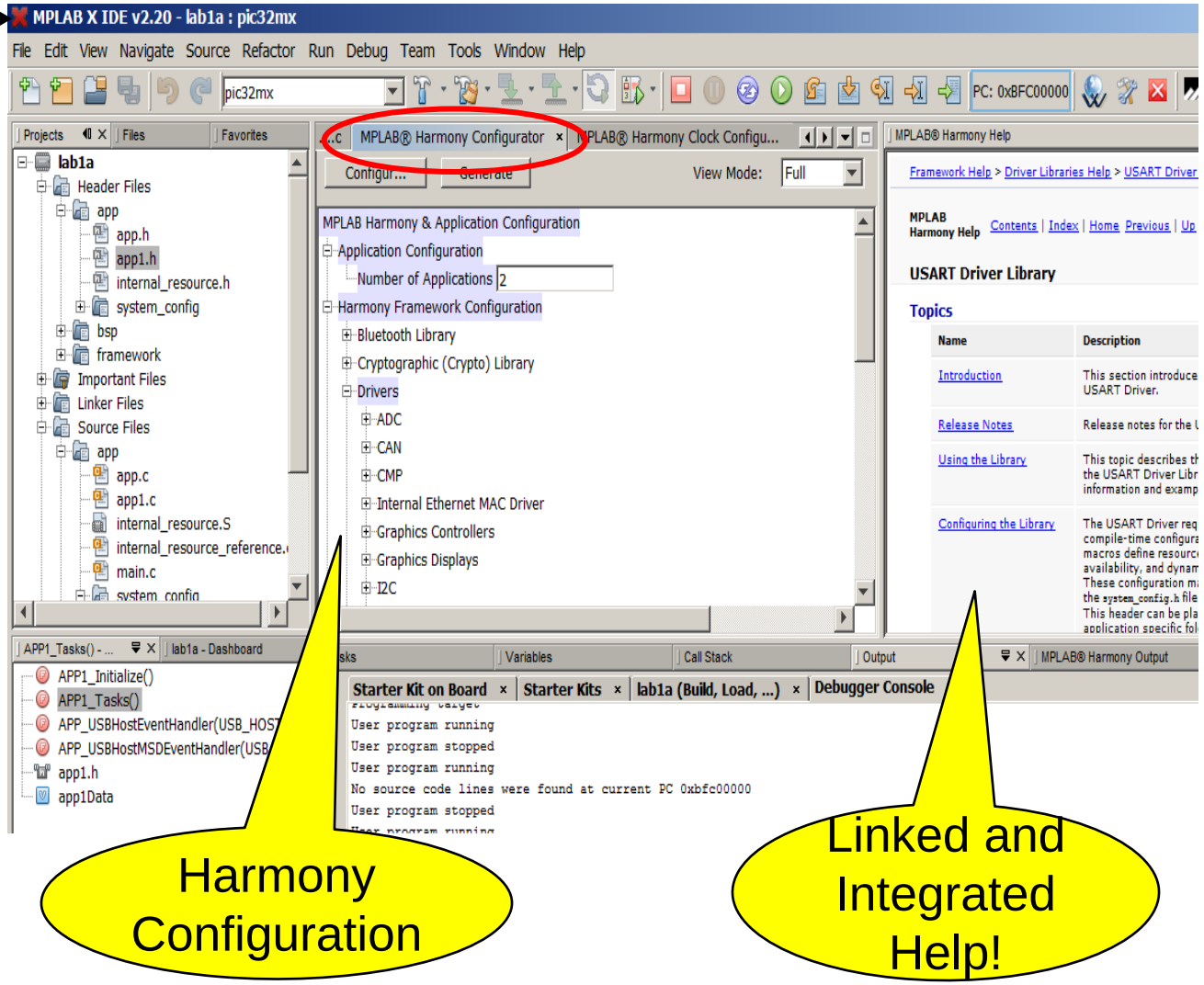
# Composite Platform



# MPLAB® Harmony Configurator (MHC)

MPLAB X

- **MHC enables:**
  - ✓ Create - Harmony project
  - ✓ Graphically select & configure - Harmony components
  - ✓ Initialize - Middleware state machines
  - ✓ Generated project can be compiled and executed
  - ✓ 3rd Party Solutions can be added and configured without updates to MHC

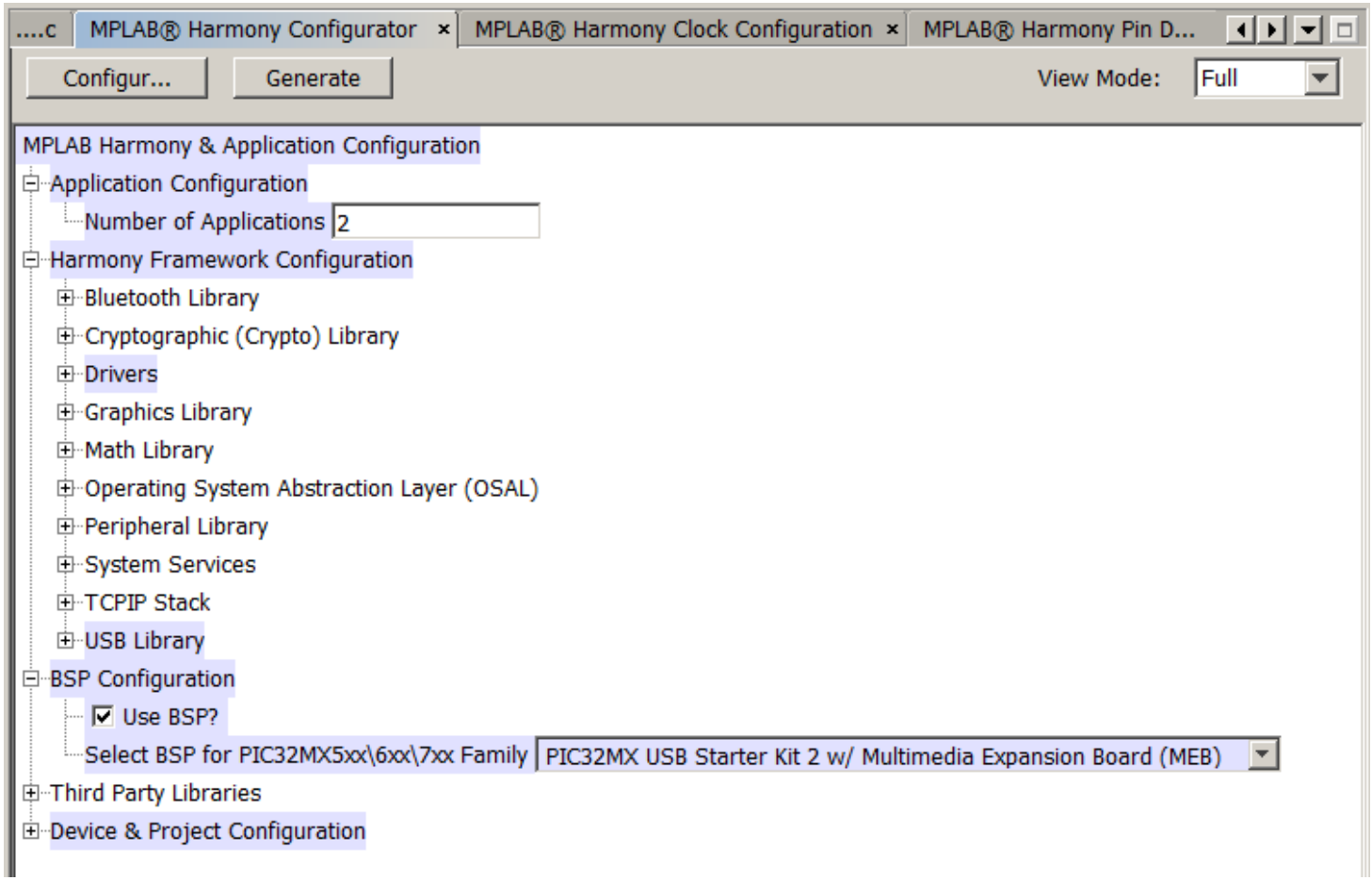


# MPLAB® Harmony Configuration

- **Key features we will review:**
  - MPLAB® Harmony Configurator (MHC)
  - MPLAB® Harmony Clock Configuration
  - MPLAB® Harmony Pin Diagram
  - MPLAB® Harmony Pin Table
  - MPLAB® Harmony & Application Configuration
    - This is where you will spend most of your time

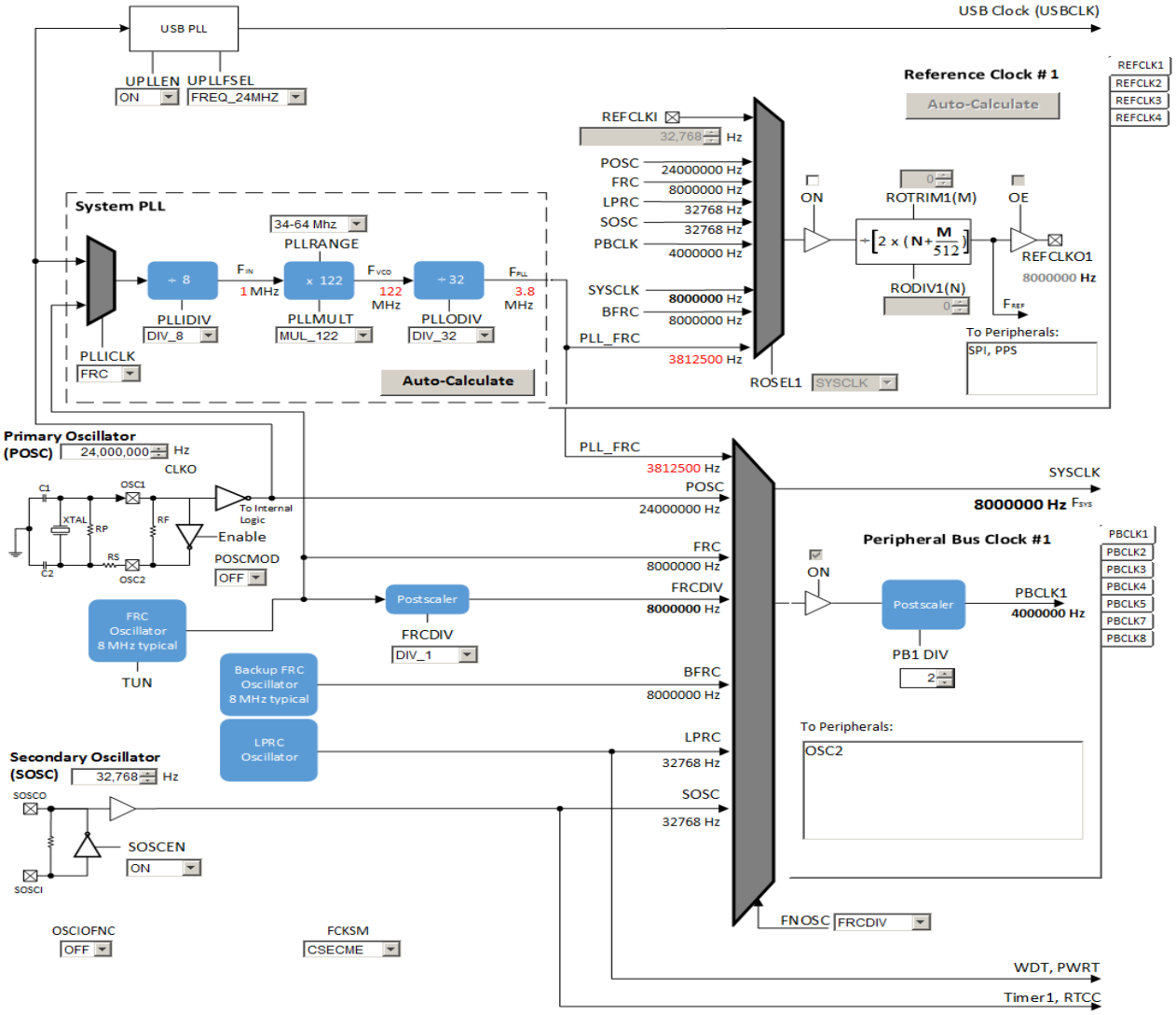
# MPLAB® Harmony Configuration

- **MPLAB® Harmony Configurator**



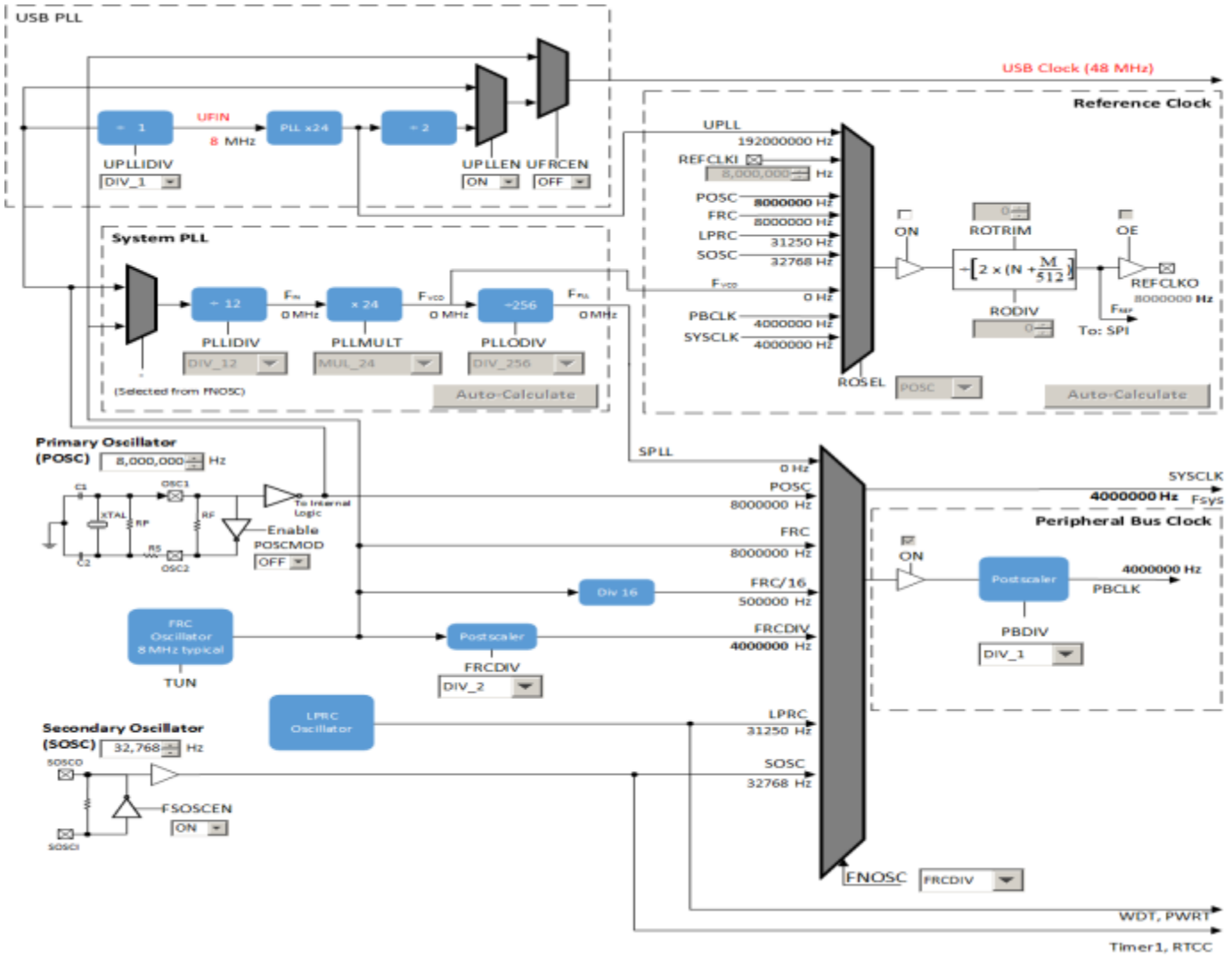
# MPLAB® Harmony Clock Configuration (PIC32MZ)

- Click and select clock configuration
- PIC32 clocks automatically configured in your project files
- Now that was easy!



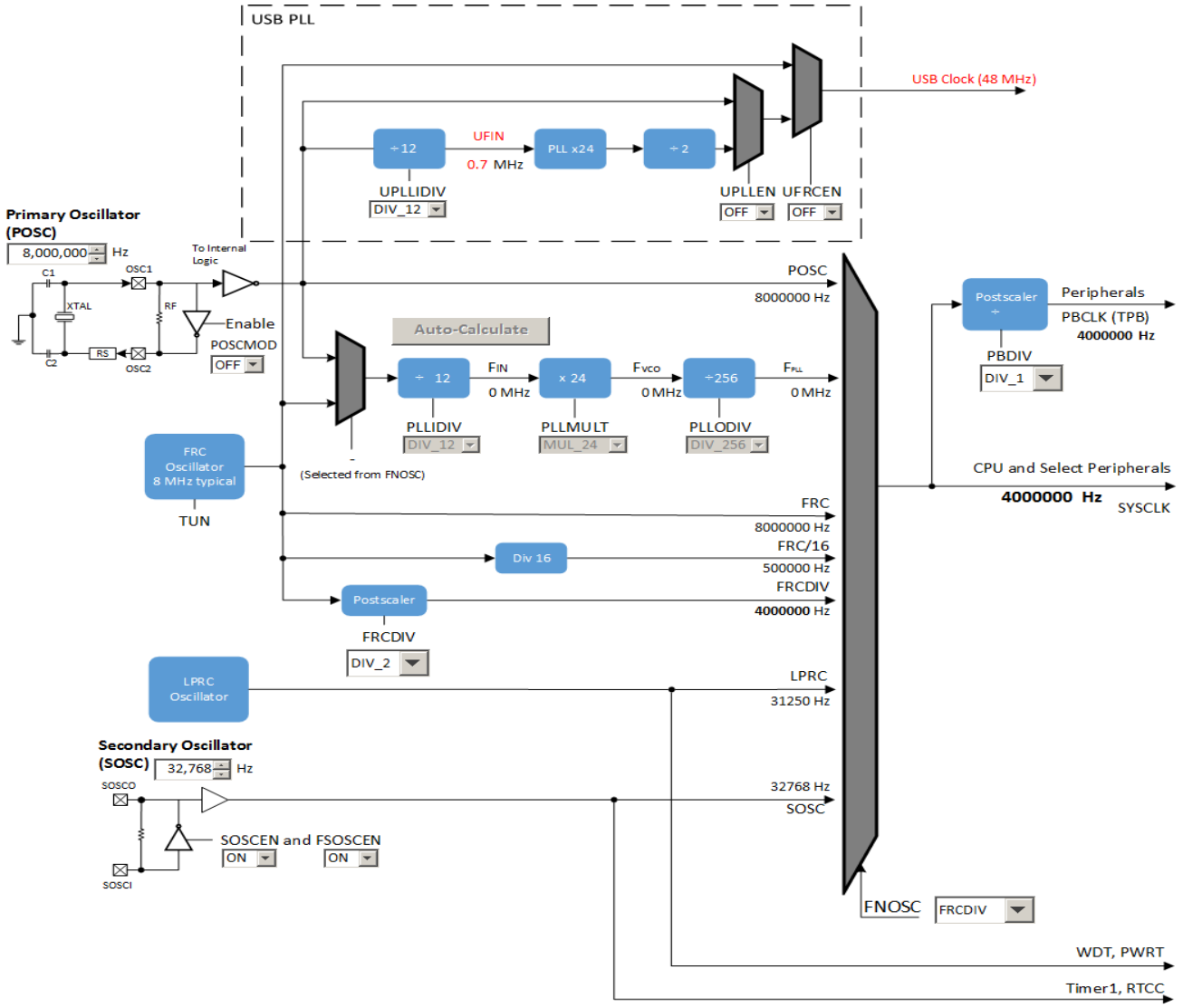
# MPLAB® Harmony Clock Configuration (PIC32MX)

- Click and select clock configuration
- PIC32 clocks automatically configured in your project files
- Now that was easy!



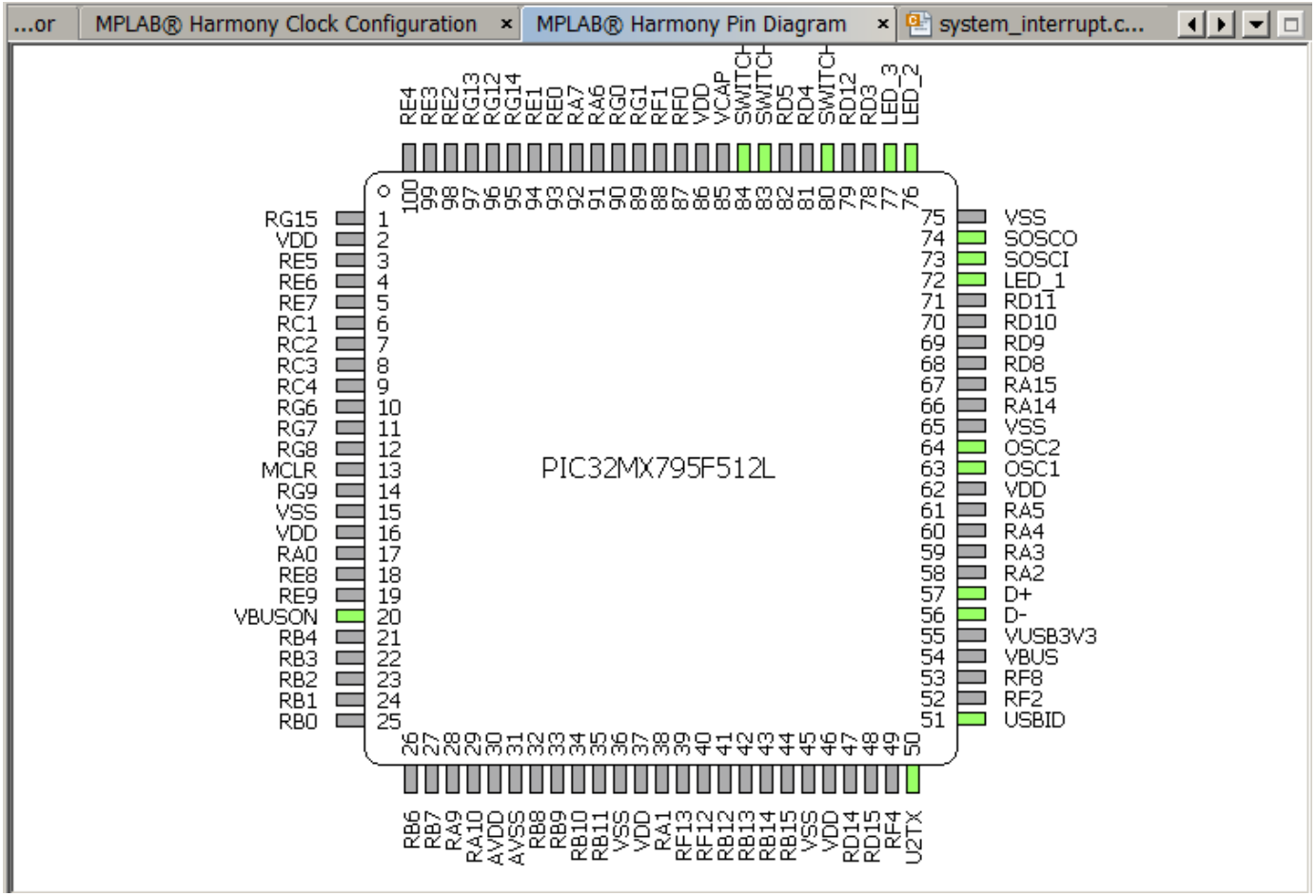
# MPLAB® Harmony Clock Configuration (PIC32MX)

- Click and select clock configuration
- PIC32 clocks automatically configured in your project files
- Now that was easy!



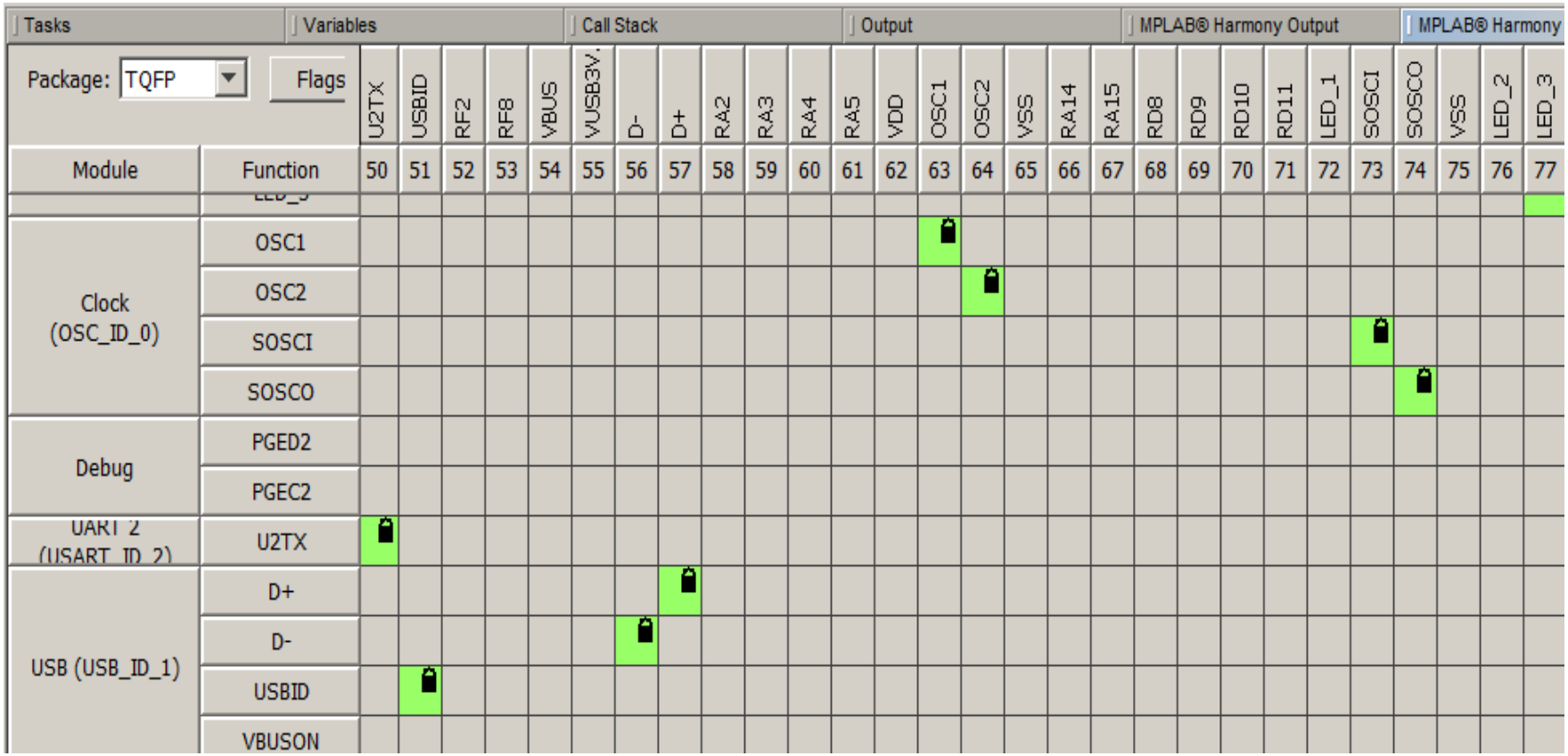
# MPLAB® Harmony Configuration

- **MPLAB® Harmony Pin Diagram – PIC32MX**



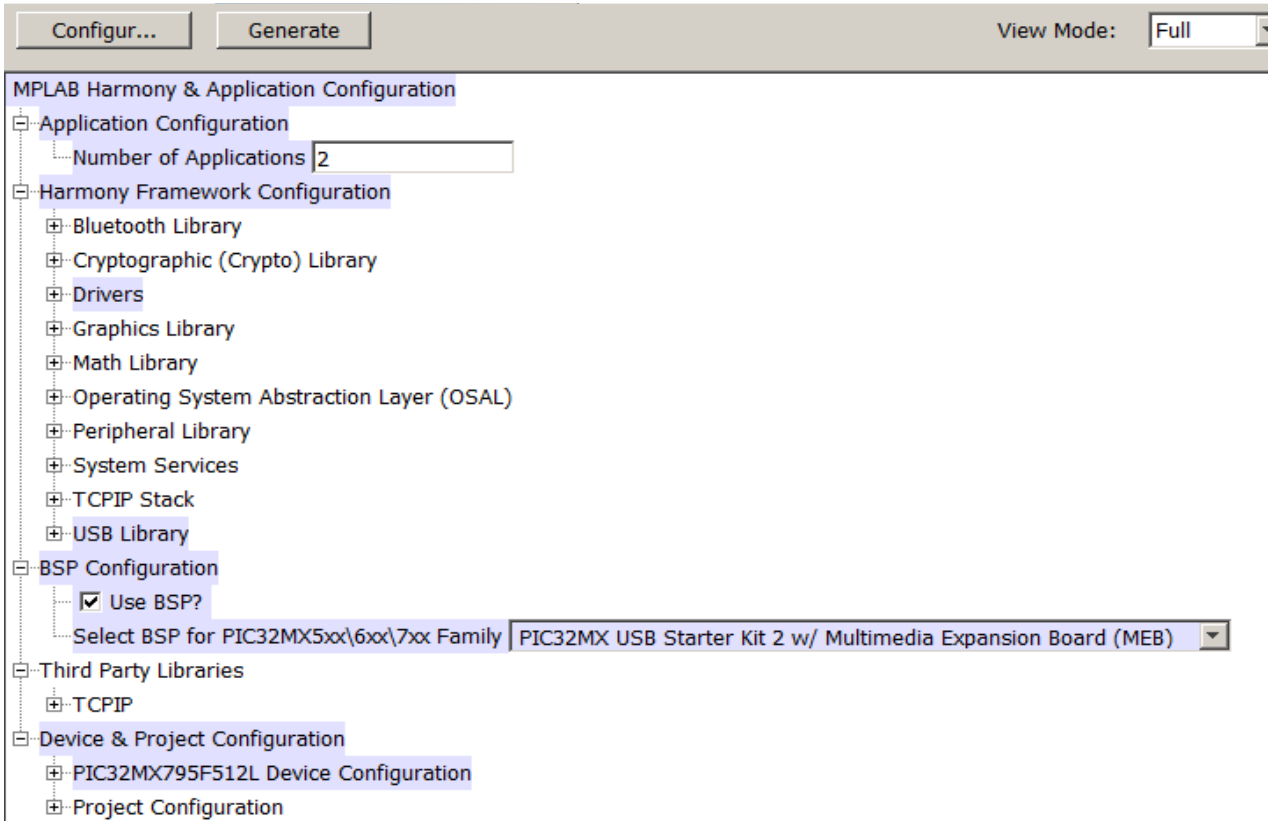
# MPLAB® Harmony Configuration

- **MPLAB® Harmony Pin Table**



# MPLAB® Harmony Configuration

- **MPLAB® Harmony & Application Configuration**
  - This is where you will spend most of your time



# Lab 1a

# Blinking an LED under interrupts using a state machine

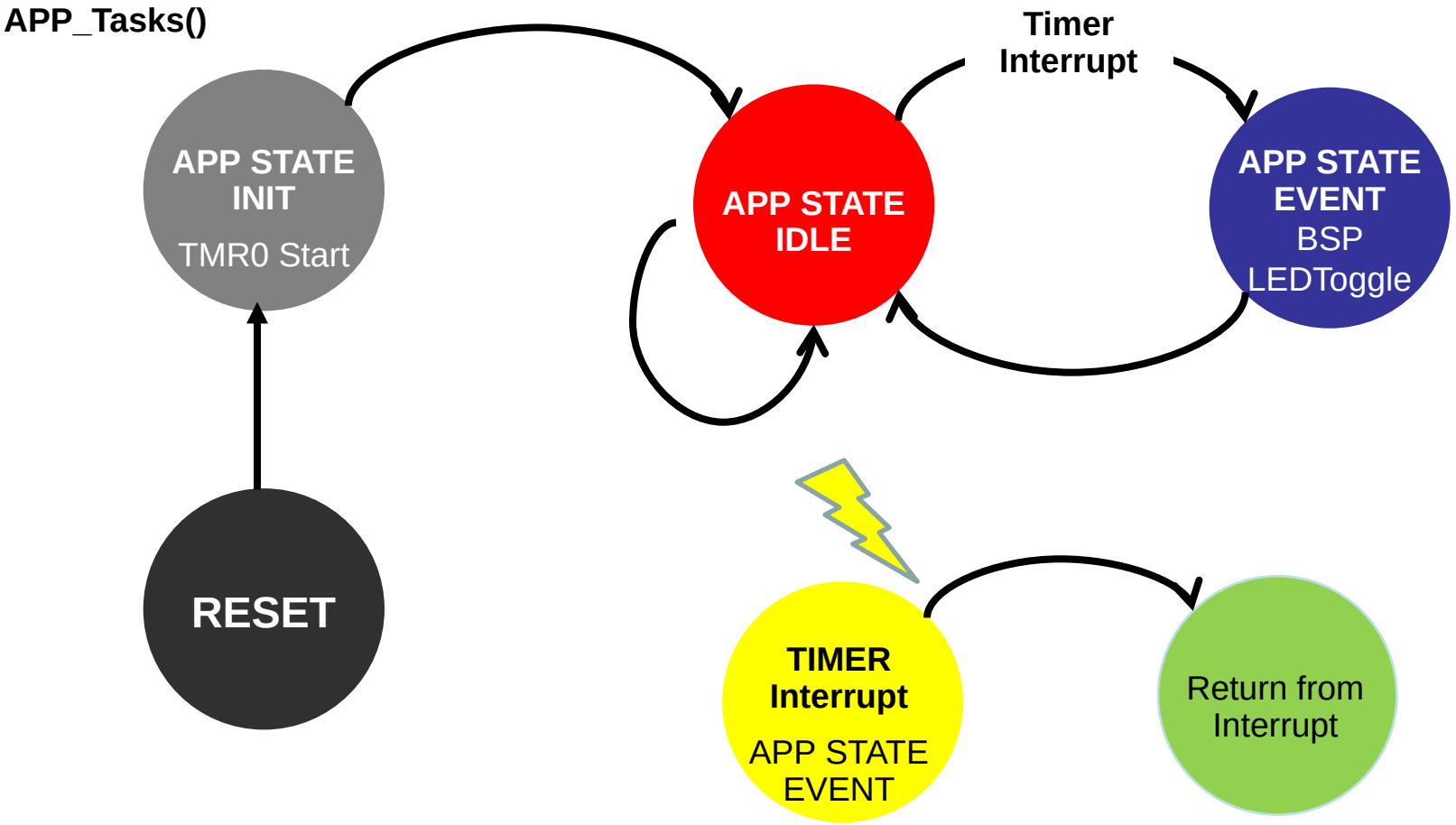
(20 mn)

**(NB : make sure to use Harmony v1.05 directory for all labs )**

# Lab 1a Objectives

- Create MPLAB® Harmony project using MPLAB® X IDE and MHC
- Begin to understand capabilities and benefits of MHC
- Create basic state machine code to “blink LED”
- Your first basic project will be ready to build upon in lab 1b

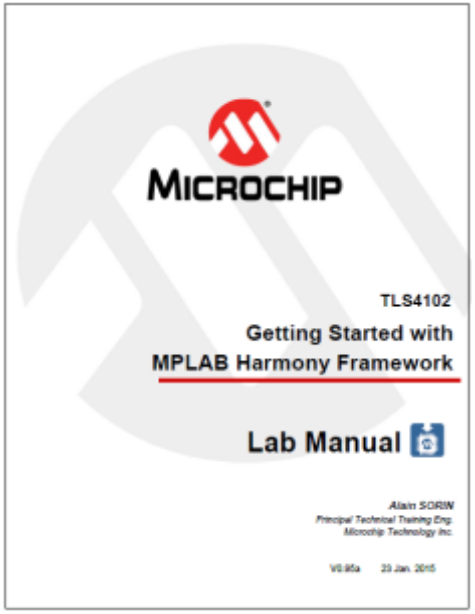
# appData.state state machine



# Lab 1a Summary Steps

- **Here are the steps to work through:**
- **Part 1: Create project and configure PIC32MZ**
  - Step 1 – Create MPLAB® X IDE and Harmony project
  - Step 2 – With MHC, set up Device Configuration
  - Step 3 – With MHC, set up BSP, and then Clocks
  - Step 4 – With MHC, set up Timer
  - Step 5 – With MHC, Generate Code
- **Part 2: Add Application code**
  - Step 6 - Add code in App Tasks function
  - Step 7 – In Timer ISR, increment (change) app state

# Please proceed to Lab1a using the Lab Manual



# Lab 1a Results

- **Build, Download and Run application**
- **Observe LED3 (D4) toggling every 1s**
  - PIC32MZ Starter Kit
  - Recall:

    BSP_LED_3 is LED3 on PIC32MZ EC starter kit

    or Led D4 on Multimedia Expansion Board II

# Lab 1a Summary

- MHC simplifies Harmony project creation process
- Understand state machine code execution flow in Harmony
- Understand basic “interrupt” execution model in Harmony
- Ready to add UART Tx in Lab 1b

# Lab 1b

# Enabling USART with MPLAB® Harmony Configurator (MHC)

(20 mn)

# Lab 1b Objectives

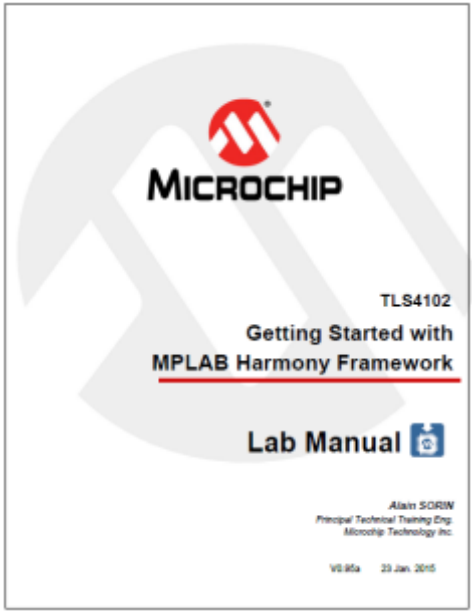
- Update Lab 1a to send single character over UART every 1s when LED turns ON
- Introduction to UART Static Driver for Tx task
- Continue learning MHC capabilities

This second project will be the starting base for Lab 2a

# Lab 1b
# Summary Steps

- **Here are the steps to work through:**
- **Part 1: With MHC, add UART functionality**
  - Step 1 - With MHC, add USART Static Driver
  - Step 2 - With MHC, connect UART2 TX to pin 61
  - Step 3 - With MHC, Generate Code
  - Step 4 - With MHC, Merge/Save Generated code
  - Step 5 - With MHC help, find USART API for byte op.
- **Part 2: Add Application code**
  - Step 6 - Add code to send USART byte

# Please proceed to Lab1b using the Lab Manual



# Lab 1b Results

- **Ensure UART-to-USB adaptor is connected to J2 header on MEB II**
- **Connect USB cable to PC and run Terminal application (9600 baud, 8N1)**
- **Build, Download and Run application**
- **Observe LED3 (D4) toggling and char ‘a’ sent every 1s**
  - PIC32MZ Starter Kit
  - Recall: BSP_LED_3 is D4

# Lab 1b Summary

- MHC makes it easy to add application functionality
- Introduction to the MPLAB® Harmony Clock Configuration tool
- Introduction to the MPLAB® Harmony Pin Table Tool
- Introduction to UART Static Driver
- Ready to add Graphics Library in Lab 2a

# MPLAB® Harmony Middleware Components

MPLAB HARMONY Integrated Software Framework

**Middleware**

- HS/FS/LS
- Multi-Instance
- Device
- Host
- HUB

*USB*

**Middleware**

- IPv4/IPv6
- Transport/App Layers
- WiFi g
- IPv6 Gold Cert

*TCP/IP*

**Plug-in Middleware**

- Graphics Library
- GDD X (GUI)
- Graphics Resource Converter (GRC)

*Graphics*

**Middleware**

- FAT,FAT16,FAT32
- MPFS
- Multi-FileSystem
- Multi-Drive
- Multi-Partition

*File System*

**Common System Services**

- Timers
- Interrupts
- PORTS
- Clocks
- Misc
- …more

*System Services*

**Driver PLIB**

- Abstracted Interface
- Multi-Client
- Dynamic/Static
- Peripheral Access

*Drivers/ PLIB*

**Middleware**

- Cryptographic Library
- Compression and Hash Libraries
- Embedded SSL Library (CyaSSL)

*Embedded Security*

**Middleware**

- Real Time Engineers Ltd.
- Wittenstein
- InterNiche Tech.
- wolfSSL
- Segger, Viosoft,
- Micrium

*3rd Party Partners*

**RTOS OSAL**

- FreeRTOS
- OPENRTOS
- uC/OS-II
- uC/OS-III
- ThreadX
- embOS

*OSAL/ RTOS*

**Middleware**

- 100+ functions DSP Library
- 29 functions Math Library

*DSP/ Math*

**Middleware**

- Bluetooth Data Stack
- Bluetooth Audio

*Bluetooth*

**System Configuration**

- MPLAB Harmony Configurator
- **Beta: MHGC MPLAB Harmony Graphics Composer v1.0 (graphics)**

*Harmony Configurator*

# USB Device Stack

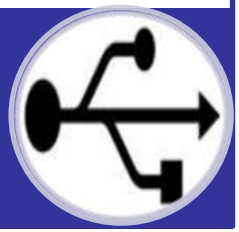
MICROCHIP

MPLAB HARMONY Integrated Software Framework

- USB 2.0 HS/FS/LS
- Multi-Instance Capable

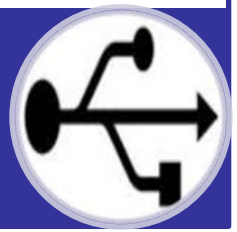
- **Classes**
  - CDC
  - HID
  - Audio
  - MSD
  - Generic

*USB Device*



- **Audio**
  - Speaker
- **CDC**
  - Single COM Port
  - Dual COM Port
  - MSD + CDC Composite Device
  - Serial Emulation
  - Serial Emulation MSD
- **HID**
  - HID Basic
  - HID Keyboard
  - HID Mouse
  - HID Joystick
  - HID + MSD
- **MSD**
  - Internal Flash
- **Generic**
  - Generic Driver

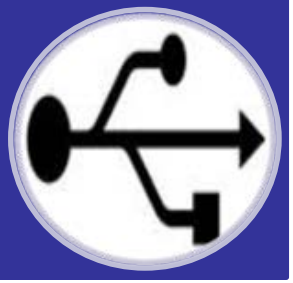
*Demos*



# USB HOST Stack

- USB 2.0 HS/FS/LS
- Multi-Instance Capable
- **Classes**
  - CDC
  - HID
  - Audio
  - MSD
  - Generic
  - HUB

*USB Host*



- **CDC**
  - Basic
  - CDC + MSD
- **MSD**
  - Simple Thumb Drive
  - Bootloader on USB stick

*Demos*

# TCP/IP + Wi-Fi

MICROCHIP

MPLAB HARMONY
Integrated Software Framework

- 10/100 Mbps Ethernet MAC with MII & RMII
- PHY (LAN8720/40/9303, ICP101A)
- BSD API wrapper
- IPv4 and IPv6
- UDP, TCP, TCP/IP Stack cmd. processor
- ICMPv4, ICMPv6
- HTTP server, SSL client and server
- SNMP and SNMPv3 server
- DHCP client, DHCP server
- SMTP client
- SNTP client
- FTP server, TFTP client, Telnet server
- Dynamic DNS client, DNS Resolver & server
- Neighbor Discovery Protocol (NDP)
- ARP module
- Microchip Announce module
- NetBIOS name server
- ZeroConf and MultiCastDNS (Bonjour)
- IPv6 Certification

**TCP/IP**

- HTTP Server
- BSD Server
- BSD Client
- BSD UDP Server
- BSD UDP Client
- SNMPv3 MCHP
- SNMPv3 FAT
- TCP Server
- TCP Client
- TCP Client + Server
- UDP Server
- UDP Client
- UDP Client + Server
- Web Server NVM
- Web Server FAT
- wolfSSL

**TCP/IP Demos**

- Easy Config Demo
- Multi-Interface HTTP Server
  - Ethernet
  - WiFi

**WiFi g Demos**

- MRF24WG WiFi MAC

**WiFi g**

# Graphics

MICROCHIP

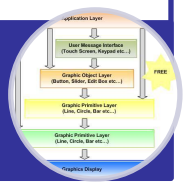
MPLAB HARMONY
Integrated Software Framework

- Quick / easy way to create GUIs for embedded apps
- Simplifies coding for GUI screens with an ability to draw, resize, and delete screen objects
- Ability to import various graphical resources, including custom fonts & bitmap images
- Imports all required driver/board support files into MPLAB X
- GDD X generates code based on Microchip Graphics (GFX) Library

*Graphics Display Designer (GDD X)*

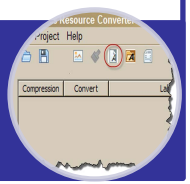
- Modular design – compile only what you need
- Not dependent on display size or resolution
- Advance features such as alpha blending, gradient, font anti-aliasing
- Utilities to import fonts and images

*GFX Library*



- Converts images, bitmaps (.bmp) and JPEG (.jpg/.jpeg), fonts, operating system's installed fonts or True Type fonts directly from files (.ttf), and binary files into formats suitable for Graphics Library
- MPLAB X Plug-in

*Graphics Resource Converter*



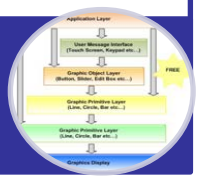
# Graphics Library

MICROCHIP

MPLAB HARMONY
Integrated Software Framework

- Feature set overview:
  - Configurable graphic resolution
  - Up to 24-bit or 65K colors
  - 2D objects such as line, circle, text, rectangle, polygon, bar
  - 3D objects such as buttons, meter, dial, list box, edit box, check box, radio buttons, window, group box, slider.
  - Image, animation
  - Variety of user input device such as touch screen, keypad etc...
  - Object and Primitive layers interface
  - Multiple fonts, Unicode fonts

*GFX Library*



- **Low Cost Controller-less Graphics Demo**
- Demo 1
  - WQVGA (480x272)
  - LCCG PICtail + Graphics Display Powertip + USBStarter Kit II.
- Demo 2
  - PIC32 GUI Dev Board w/ Cap Touch
- Demo 3
  - MEB II + PIC32MZ EC Starter Kit

- **Object Layer Demos - Shows how objects are used with user messages interacting with objects**
- Demo 1
  - QVGA (320x240)
  - Display Controller, SSD1926
  - PIC32MX Starter Kit + GFX PICtail
- Demo 2
  - WQVGA (480x272) LCCG
  - PIC32MZ Starter Kit + Multimedia Expansion Board II

- **Primitive Layer Demos - Shows how primitive functions are used**
- Demo1
  - WQVGA (480x272)
  - Display Controller, S1D13517
  - PIC32MX Starter Kit + GFX PICtail

*Demos*

- **Primitive Layer Demos (cont.)**
- Demo 2
  - QVGA (320x240)
  - Display Controller, SSD1926
  - PIC32MX Starter Kit + GFX PICtail
- Demo 3
  - WQVGA (480x272)
  - Low Cost Controllerless Graphics
  - PIC32MZ Starter Kit + Multimedia Expansion Board II
- Demo 4
  - MEB II + PIC32MZ EC Starter Kit
- Demo 5
  - MEB + PIC32 USB Starter Kit II
- Demo 6
  - PIC32 Bluetooth Audio Development Kit

- **Epson Controller S1D13517 Features Demo**
- Demo 1
  - Graphics Display Truly 7” 800x400
  - PIC32 USB Starter Kit + GFX PICtail
- Demo 2
  - Graphics Display Truly 5.7” 640x480
  - PIC32 USB Starter Kit + GFX PICtail

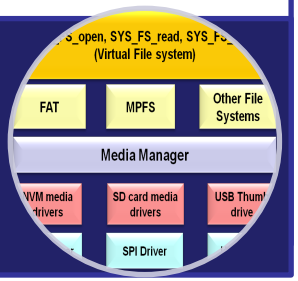
- **JPEG image playback from SD Card Demo**
  SSD1926 (Solomon Systech Controller JPEG decoder)
  QVGA (320x240)
  PIC32MX Starter Kit + GFX PICtail

*Demos*

# File System

- Hardware and File System Type Independent “Virtual” File System API
- RTOS friendly
- Multi instance capable
  - Can create multiple disks
  - Can use multiple media – NVM, SD card, USB thumb drive
  - Multiple file system types
- Support any physical media, which has its own Harmony compliant drivers available
- Intended by design to work with other Harmony Middleware – TCP/IP, USB and Graphics

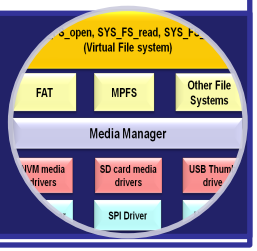
*File System*



# File System

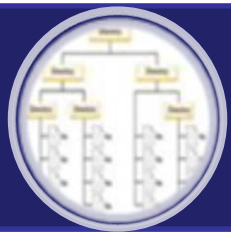
- FAT12/16/32
- MPFS (Microchip File System)
- Multi-Files System
- Multi-Drives
- Multi-Partitions
- SD Card
- Flash Drive

*File System*




- Single Media
  - Demos
    - FAT12 image on NVM Flash
    - MPFS image (2 files) on NVM Flash
    - FAT LFN on SD Card
  - Platforms
    - PIC32 Bluetooth SK
    - PIC32 USB SK II
    - PIC32 USB SK III
    - PIC32MZ EC SK
- Dual Media
  - Demos
    - NVM + SD Card with MPFS + FAT (read/write/verify)
    - NVM + SD FAT (Search/open/read - write on other)
    - USB Flash + SD (PIC32MZ EC SK + MEB II only)
- Platforms
  - Explorer 16 + PIC32MX795F512L PIM + PICtail
  - Explorer 16 + PIC32MX270F256D PIM + PICtail
  - Explorer 16 + PIC32MX470F512L PIM + PICtail
  - PIC32MZ EC Starter Kit + MEB II

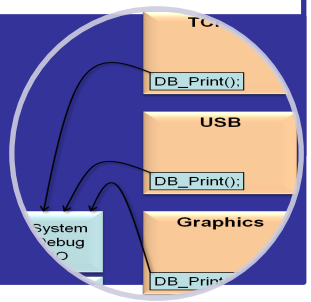
*Demos*



# System Services

- Manage shared resources:
  - Provides Common API
  - Avoid conflicts between modules
  - Avoid duplication
- Examples:  software timers, interrupts, ports, debug output, Reset control, etc.
- Applications, drivers, and middleware and other libraries do not have to implement these features
- Creates consistency across all modules

*System Services*

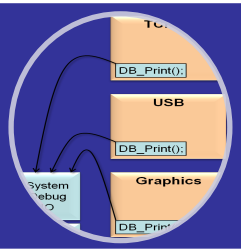


# System Services

- **Core System Services supported**
- **FS, MSG, TMR, INT, PORTS, RESET, WDT, CMD, ERR/DEBUG, IO, RAND**

- System Services provided:
  - Timers
  - Interrupts
  - PORTS
  - Reset
  - Watch Dog Timer
  - Command Console IO
  - Debug/Error Output
  - Random Number Generator
  - …

*System Services*



# Drivers / PLIBs

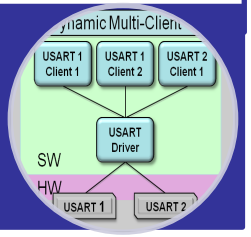
MICROCHIP

MPLAB HARMONY
Integrated Software Framework

- Provides simple hardware independent abstracted interface to peripherals.
- Manages peripheral access control to avoid conflicts
- Manages peripheral state

- Dynamic/Static
  - Static Single/Multi-Client
  - Dynamic Multi-Client
- RTOS Thread Safe
- Blocking or Non-blocking
- H/W Access via PLIB

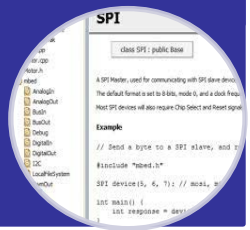
## Drivers




- Provides hardware dependent low level direct interface to hardware peripheral
  - PLIBs do not manage the state of the peripheral or control access to the peripheral (driver function)

- Provides common functional interface for PIC32 MCU compatibility
- Exposes All Peripheral Features
- Part-Specific implementations

## PLIBs

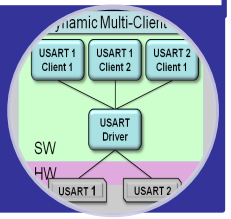


# Drivers / PLIBs

- **PIC32MX and PIC32MZ MCU PLIBs available**

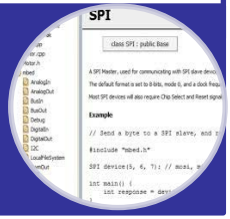
- ADC
- Ethernet MAC & PHY
- Graphics
- Wi-Fi
- Timer
- NVM
- HS-USB
- CAN
- SPI
- UART
- PMP
- SD Card
- I2S

**Drivers**




- PLIBs available for all PIC32 MCUs
  - PIC32MX
  - PIC32MZ

**PLIBs**



# Embedded Security

- **Software & Hardware Crypto (PIC32MZ)**

- AES-128, -192 & -256
  - Direct, CBC & CTR
- 3DES (CBC)
- ECC
- RSA-1024, -2048
- Elliptic curve Diffie–Hellman (192 – 521)
- Elliptic Curve Digital Signature Algorithm (192 – 521)

*Cryptographic Library*


- Huffman: dynamic + static
- SHA-1, -256, -384 & -512
- HMAC
  - SHA-1, -256, -384 & -512
- MD5
- Cryptographically Secure RNG

*Compression & Hash Libraries*


- Lightweight SSL library written in ANSI C and targeted for embedded and RTOS environments
- Up to TLS 1.2 and DTLS 1.2
- Full client and server support
- Progressive list of supported ciphers
- Key and Certificate generation
- OCSP, CRL support

- **(*Available from wolfSSL)**
  - www.wolfSSL.com

*CyaSSL Embedded SSL Library**

# OS Abstraction Layer (OSAL)

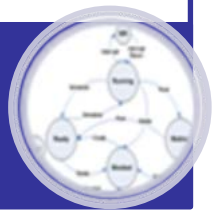
Provides consistent interface to specifically targeted RTOS so that Harmony Drivers, Middleware, etc. may be made thread safe using a single interface.

- **OSAL Services**
  - Semaphores
  - Mutexes
  - Queues
  - Memory Allocation
  - Critical Sections
  - OSAL Control and Diagnostics
- Minimal API to allow Broadest Compatibility

*OSAL*

- None (removes OSAL)
- Basic (Bare Metal)
- FreeRTOS
- OpenRTOS
- Micrium µC-OS/II
- Micrium µC-OS/III
- emBOS
- ThreadX

*Supported RTOSes*



# Harmony Libraries Status

MICROCHIP

MPLAB HARMONY Integrated Software Framework

| USB | TCP/IP | Graphics | File System/ Memories | System Services | Drivers/ PLIB |
|---|---|---|---|---|---|
| • USB 2.0 Host & Device<br>• Device Stack v1.0<br>• Device Multi-Instance<br>• Device Classes: CDC, HID, Audio, MSD, Generic<br>• USB Host Stack (Beta)<br>• Host support for multiple peripherals<br>• Host class support: CDC, MSD | • IPv4/IPv6<br>• 10/100 Mbps Enet MAC with MII & RMII<br>• HTTP Server (MPFS, FAT FS)<br>• BSD TCP/UDP Server<br>• BSD TCP/UDP Client<br>• TCP Server & Client<br>• UDP Server & Client<br>• SNMP Server (MPFS, FAT FS)<br>• wolfSSL (SSL Library)<br>• MRF24WG Wi-Fi MAC<br>• Broad Range of Wi-Fi Demos | • Java-based MHC<br>• Java- based GDDX<br>• Java-based GRC<br>  • MHC integrated.<br>• GFX Library<br>• Touch | • Hardware and File System Type Independent "Virtual" File System API<br>  • FAT32<br>  • MPFS (Microchip FS)<br>  • RTOS friendly<br>  • Multi instance capable<br>  • Multi-File System<br>  • Multi-Partition<br>  • Multi-Drive<br>  • SD Card<br>  • Flash Drive<br>  • Harmony Compatible FatFs file system | • Clk<br>• Command<br>• Devcon<br>• DMA<br>• File System<br>• Interrupt<br>• Ports<br>• Random | Drivers:<br>• PIC32MX:<br>  • Ethernet, WiFi, USB<br>  • UART, SPI, CAN<br>  • GFX, Timer, PMP<br>• PIC32MZ:<br>  • Ethernet, WiFi, HSUSB<br>  • UART, SPI, CAN<br>  • GFX, Timer, PMP<br>PLIBs:<br>• PIC32MX320_340_360_420_440_460<br>• PIC32MX575/675/695/775<br>• PIC32MX534/564/664/764<br>• PIC32MX1XX/2XX<br>• PIC32MX1XX/2XX (256KB/64KB)<br>• PIC32MX330_350_370_430_450_470<br>• PIC32MX320_340_360_420_440_460<br>• PIC32MX1XX/2XX/5XX (beta) |

| Security | OSAL/RTOS | MHC |
|---|---|---|
| • Cryptographic Library<br>• Compression and Hash Libraries<br>• Embedded SSL Library (CyaSSL) | Ports available now:<br>• PIC32MX & PIC32MZ<br>• FreeRTOS<br>• OPENRTOS<br>• Micrium, µC/OS-III, µC/OS-II<br>• Express Logic, ThreadX<br>• Segger, embOS<br>OSAL available now:<br>• FreeRTOS, OPENRTOS,<br>• Micrium µC/OS-III & µC/OS-II<br>• Express Logic/ThreadX<br>• Segger/embOS | • Static Drivers<br>• Configuration for all modules<br>• Pin Manager<br>• Clock Management<br>• 3rd party configuration<br>• Integrated documentation<br>• All Harmony projects under MHC control |

# Harmony Libraries Roadmap

- **Current Version – v1.05 (released June 2015)**
  - Support for PIC32MX1xx/2xx 28/36/44-pin and PIC32MZ EF
  - Support for MPLAB XC32 C/C++ Compiler v1.40
  - C++ compatibility issues were corrected
  - AAC_Decode support
  - TFTP Support
  - Documentation for SST25VF064 Flash device
  - USB Host Stack Beta
  - 12-bit High-Speed SAR ADC Peripheral Library added
  - All Graphics demo regenerated using MPLAB Harmony Graphics Composer (MHGC)

- **Upcoming Releases:**

  *V1.06 - July 2015*
  *V1.07 - October 2015*

  - USB Device Stack into Production
  - USB Host Stack into Production
  - TCP/IP Stack into Production
  - **MRF24WN WiFi Driver**
  - File System Support to Production
  - **External Serial Flash Support in File System**
  - BT/Audio component Beta and Production (Standard and Premium)
  - MHC Improvements
  - Added **Bootloader** targets
    - SD Card
    - Ethernet Pull
    - I2C Master
    - Live-Update with State Retention
  - System Services, Drivers and PLIBs to Production

# Agenda

- Overview
- Getting Started + Hands-on Labs
- **Adding Features/Functionality + Labs**
- Application Migration + Lab
- Harmony RTOS Application + Demo
- Third Party Ecosystem Solutions
- Wrap-up, next steps

# Lab 2a

# Enabling Graphics with MHC

40 mn

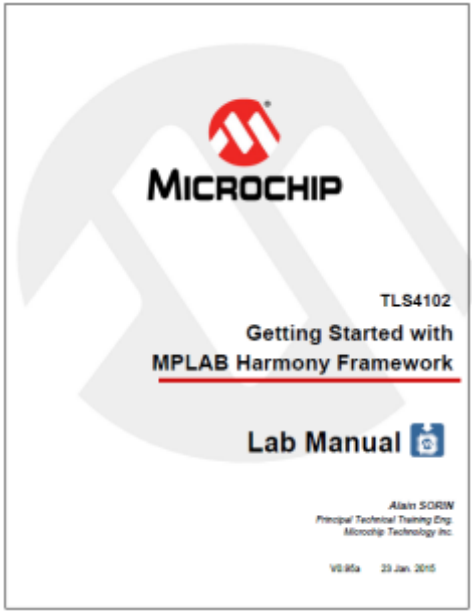
# Lab 2a Objectives

- Enable Harmony Graphics in your application with MHC
- Basic understanding on accessing several GFX APIs
- Additional hands-on experience with MHC

# Lab 2a
# Summary Steps

- **Here are the steps to work through:**
- **With MHC add Graphics functionality**
  - Step 1 - Add pre-built GRC generated files to project
  - Step 2 - With MHC, enable Graphics into your project
  - Step 3 - With MHC, Generate Code
  - Step 4 - With MHC, Merge/Save Generated code
  - Step 5 - Add code: Initialize screen, draw box, add text
  - Step 6 - Add code: Toggle rectangle, fill color for LED

# Please proceed to Lab2a using the Lab Manual



# Lab 2a Results

- **Ensure UART-to-USB adaptor is connected to J2 header on MEB II**
- **Connect USB cable to PC and run Terminal application (9600 baud, 8N1)**
- **Build, Download and Run application**
- **Observe LED3 (D4) toggling and char ‘a’ sent every 1s**
  - Widget, named D4, will toggle at same rate as LED if graphics library is working correctly.

# Lab 2a Summary

- Continued hands-on experience with the MPLAB® Harmony Configurator
- Introduction to some basic GFX Library APIs
- Continued familiarization with the integrated Harmony help docs
- Ready to add USB Host MSD in Lab 2b

# Lab 2b

# Enabling USB Host MSD and File System

60 mn

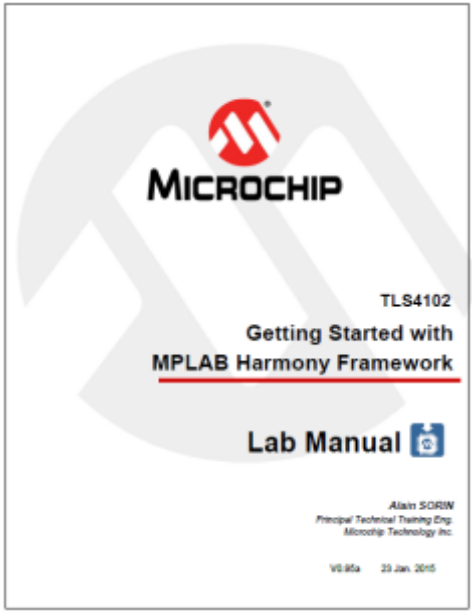
# Lab 2b Objectives

- Enabling and configuring Harmony USB Host MSD and File System within your application with MHC
- Further touch on MHC and basic understanding on MSD File System APIs
- Add code to access a pre-canned message from USB pen drive and generate a UART Tx task to display message contents

# Lab 2b
# Summary Steps

- **Here are the steps to work through:**
- **Add USB Host and File System Components**
  - Step 1 - Enable USB Host with MSD
  - Step 2 - With MHC, Generate Code, Merge Auto-Generated and custom code
  - Step 3 - With MHC, Reconfigure Timer Services
  - Step 4 - With MHC, Generate Code, Merge Auto-Generated and custom code
  - Step 5 - With MHC, create two Application Configurations
  - Step 6 - With MHC, Generate Code, Merge Auto-Generated and custom code
  - Step 7 - Add USB and MSD callback functions
  - Step 8 - Start Timer Service as 1 second event
  - Step 9 - Use Timer Service as 1 second event
  - Step 10 - Add USB MSD application states
  - Step 11 - Add USB MSD application state execution code
  - Step 12 - Update Initialization code in app.c

# Please proceed to Lab2b using the Lab Manual



# Lab 2b Results

- **Ensure UART-to-USB adaptor is connected to J2 header on MEB II**
- **Connect USB cable to PC and run Terminal application (9600 baud, 8N1)**
- **Build, Download and Run application**
- **Observe LED3 (D4) toggling and char ‘a’ sent every 1s**
  - PIC32MZ Starter Kit
  - Recall: BSP_LED_3 is D4

# Lab 2b Summary

- MHC makes it easy to add Harmony USB and File System middleware components
- Introduction to basics of USB Host MSD File System APIs

# Agenda

- Overview
- Getting Started + Hands-on Labs
- Adding Features/Functionality + Labs
- **Application Migration + Lab**
- Harmony RTOS Application + Demo
- Third Party Ecosystem Solutions
- Wrap-up, next steps

# Lab 3

# Solution Migration in MPLAB® Harmony

30-minutes

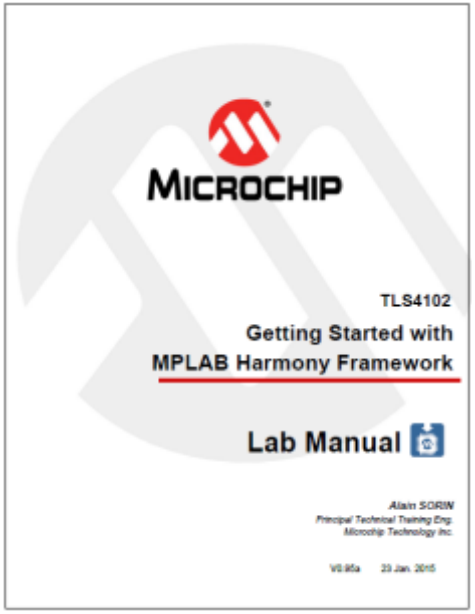
# Lab 3 Objectives

- **Upon completion of this lab, you will be able to…**
  - Use MHC and take an existing PIC32MZ project and quickly migrate to PIC32MX.
    - No code will be developed for this section
  - Further familiarize with the MHC tool and its capabilities
  - Understand the basics on how to efficiently migrate solutions and libraries across PIC32 families.

# Lab 3
# Summary Steps

- **Using the existing/open project we will:**
- **Part 1: Create New Configuration**
  - Step 1 – Create new MPLAB® X IDE Configuration
  - Step 2 – Choose Device and Tools
  - Step 3 – Set new “pic32mx” configuration as active
- **Part 2: Configure PIC32MX and Harmony**
  - Step 4 – With MHC, Configure PIC32MX BSP
  - Step 5 – With MHC, Configure PIC32MX Clocks
  - Step 6 – With MHC, Configure PIC32MX UART Driver
  - Step 7 – With MHC, Configure Graphics Library
  - Step 8 – With MHC, Configure USB Host MSD Library
  - Step 9 – With MHC, Generate and Save code

# Please proceed to Lab3 using the Lab Manual



# Lab 3 Summary

- MHC makes it easy to migrate solutions across the PIC32 product family
- You gained a better understanding on how to use MPLAB® X IDE and MHC to seamlessly migrate your application solution across the PIC32 MCU product families

# Agenda

- Overview
- Getting Started + Hands-on Labs
- Adding Features/Functionality + Labs
- Application Migration + Lab
- **Harmony RTOS Application + Demo**
- Third Party Ecosystem Solutions
- Wrap-up, next steps

# Lab 4 Demo

# Using MPLAB® Harmony in a RTOS based application

45 mn

# Session Objectives

- Overview of adding Harmony to a RTOS based application
- Gain a brief understanding of how Harmony drivers and middle-ware operate in a multi-threaded preemptive environment
- Brief overview of the term Operating System Abstraction Layer (OSAL)

# System Configuration

MICROCHIP

MPLAB HARMONY Integrated Software Framework

## system_config.h

```
#define APP_TMR_HW_ID                   TMR_ID_3
#define SYS_CLK_FREQUENCY               0x0BEBC200
#define APP_LED_BLINKING_RATE           0x0002625A
```

## system_initialize.c

```
#pragma config FPLLODIV = DIV_1, FPLLMUL = MUL_20, FPLLIDIV = DIV_2, FWDTEN = OFF
#pragma config OSCIOFNC = ON, POSCMOD = HS, FSOSCEN = ON, FNOSC = PRIPLL
#pragma config ICESEL = ICS_PGx2

void SYS_Initialize( void )
{
    /* Call all library & application initialization routines */
}
```

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

    while(true)
    {
        SYS_Tasks();
    }

    return(EXIT_VALUE);
}
```

## system_tasks.c

```
void SYS_Tasks( void )
{
    /*Call all polled library & app "Tasks" routines*/
}
```

## system_interrupt.c

```
void __ISR ( _TIMER_3_VECTOR ) _ISR_TMR_3_stub ( void )
{
    /* Call the timer library's interrupt routine */
}
```

# System Configuration – System Initialization

MICROCHIP

MPLAB HARMONY Integrated Software Framework

system_config.h

```
#define APP_TMR_HW_ID                   TMR_ID_3
#define SYS_CLK_FREQUENCY               0x0BEBC200
#define APP_LED_BLINKING_RATE           0x0002625A
```

system_initialize.c

```
#pragma config FPLLODIV = DIV_1, FPLLMUL = MUL_20, FPLLIDIV = DIV_2, FWDTEN = OFF
#pragma config OSCIOFNC = ON, POSCMOD = HS, FSOSCEN = ON, FNOSC = PRIPLL
#pragma config ICESEL = ICS_PGx2

void SYS_Initialize( void )
{
   /* Call RTOS specific initialization first*/
   /* Call all library & application initialization routines */
}
```

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

    while(true)
    {
        SYS_Tasks();
    }

    return(EXIT_VALUE);
}
```

system_tasks.c

```
void SYS_Tasks( void )
{
    /*Call all polled library & app "Tasks" routines*/
}
```

system_interrupt.c

```
void __ISR ( _TIMER_3_VECTOR ) _ISR_TMR_3_stub ( void )
{
    /* Call the timer library's interrupt routine */
}
```

# System Configuration – RTOS – System Initialize

**system_initialize.c**

```
#pragma config FPLLODIV = DIV_1, FPLLMUL = MUL_20,
#pragma config FPLLIDIV = DIV_2, FWDTEN = OFF
#pragma config OSCIOFNC = ON, POSCMOD = HS,
#pragma config FSOSCEN = ON, FNOSC = PRIPLL
#pragma config ICESEL = ICS_PGx2

void SYS_Initialize( void )
{
 /* Call RTOS specific initialization functions */


 /* Call Harmony and application specific initialization functions */

}
```

# System Configuration – System Tasks

MICROCHIP

MPLAB HARMONY Integrated Software Framework

system_config.h

```
#define APP_TMR_HW_ID                   TMR_ID_3
#define SYS_CLK_FREQUENCY               0x0BEBC200
#define APP_LED_BLINKING_RATE           0x0002625A
```

system_initialize.c

```
#pragma config FPLLODIV = DIV_1, FPLLMUL = MUL_20, FPLLIDIV = DIV_2, FWDTEN = OFF
#pragma config OSCIOFNC = ON, POSCMOD = HS, FSOSCEN = ON, FNOSC = PRIPLL
#pragma config ICESEL = ICS_PGx2

void SYS_Initialize( void )
{
    /* Call all library & application initialization routines */
}
```

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

    while(true)
    {
        SYS_Tasks();
    }

    return(EXIT_VALUE);
}
```

system_tasks.c

```
void SYS_Tasks( void )
{
    /*Call all polled library & app "Tasks" routines*/
}
```

system_interrupt.c

```
void __ISR ( _TIMER_3_VECTOR ) _ISR_TMR_3_stub ( void )
{
    /* Call the timer library's interrupt routine */
}
```

# System Configuration – RTOS – System Tasks

## system_task.c

```
void SYS_Tasks( void )
{
   /* Create your RTOS Tasks/Threads */

   /* Start Scheduler */


}
```

### Task A

```
void Task_A_Func(void* pvParameter)
{
   while(true)
   {
      /*perform Harmony or
      application specific task
      job*/
   }
}
```

### Task B

```
void Task_B_Func(void* pvParameter)
{
   while(true)
   {
      /*perform Harmony or
      application specific task
      job*/
   }
}
```

# System Configuration – System Interrupts

MICROCHIP

MPLAB HARMONY Integrated Software Framework

system_config.h

```
#define APP_TMR_HW_ID                   TMR_ID_3
#define SYS_CLK_FREQUENCY               0x0BEBC200
#define APP_LED_BLINKING_RATE           0x0002625A
```

system_initialize.c

```
#pragma config FPLLODIV = DIV_1, FPLLMUL = MUL_20, FPLLIDIV = DIV_2, FWDTEN = OFF
#pragma config OSCIOFNC = ON, POSCMOD = HS, FSOSCEN = ON, FNOSC = PRIPLL
#pragma config ICESEL = ICS_PGx2

void SYS_Initialize( void )
{
    /* Call all library & application initialization routines */
}
```

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

    while(true)
    {
        SYS_Tasks();
    }

    return(EXIT_VALUE);
}
```

system_tasks.c

```
void SYS_Tasks( void )
{
    /*Call all polled library & app "Tasks" routines*/
}
```

system_interrupt.c

```
void __ISR ( _TIMER_3_VECTOR ) _ISR_TMR_3_stub ( void )
{
    /* Call the timer library's interrupt routine */
}
```

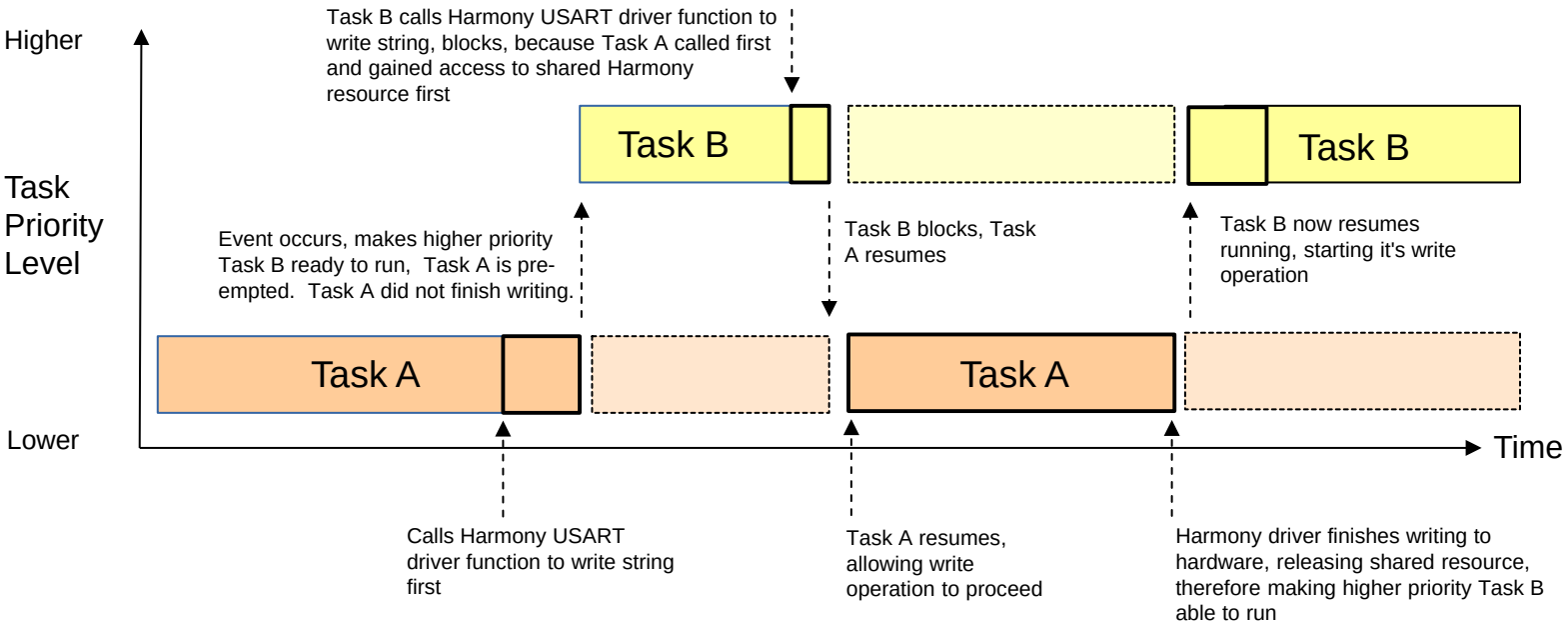
# System Configuration – RTOS – System Interrupts

Integrated Software Framework

**system_interrupt.c**

```
/*example of RTOS unaware interrupt*/
void __ISR ( _TIMER_3_VECTOR ) _ISR_TMR_3_stub ( void )
{
    /* no RTOS API calls here, timing critical work only */
}


/*example of RTOS aware or RTOS friendly interrupt*/
void __ISR(_Timer_2_VECTOR) ISR_TMR_2_stub (void)
{
    /* Call RTOS ISR Enter function*/

    /* Call applicable RTOS API, and/or Harmony functions here*/

    /* Call RTOS ISR Exit function*/
}
```

# Task Operation



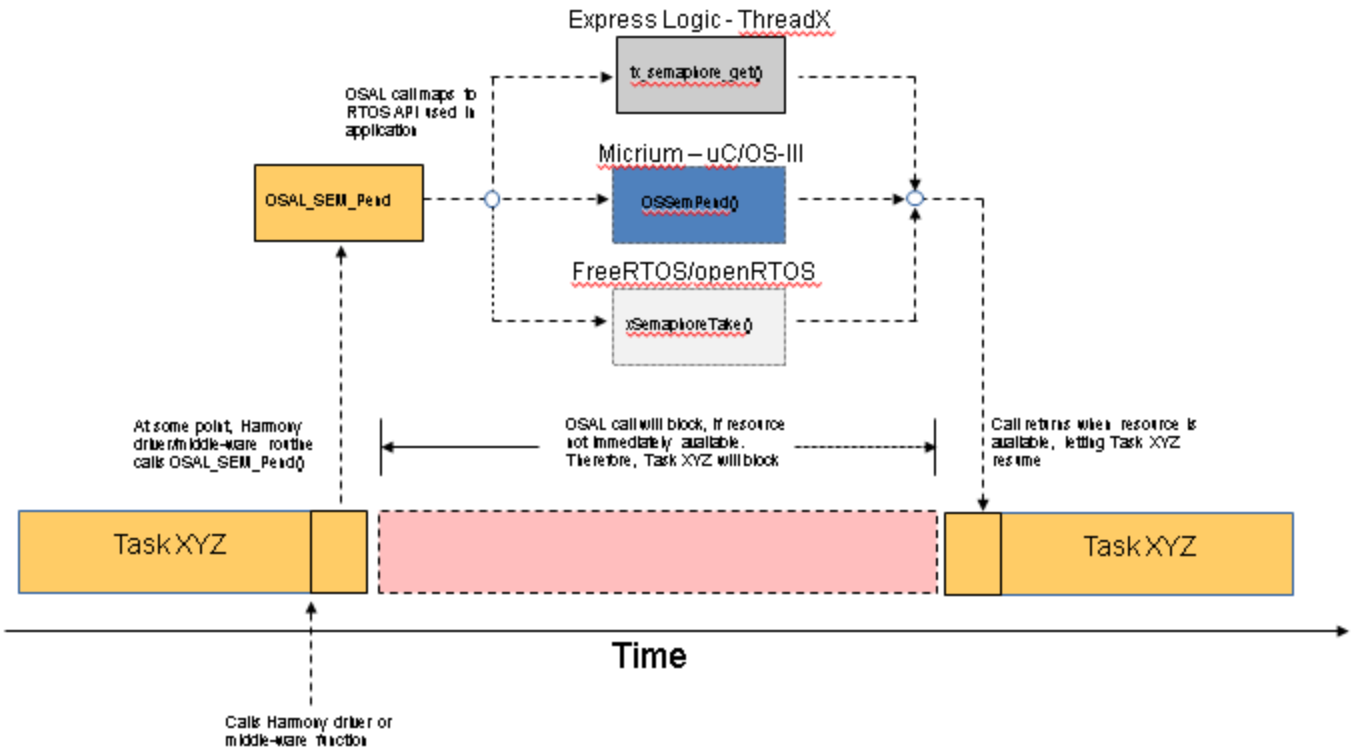

## Task A

```
signed char* pTaskA_str = "Hello World";

void Task_A_Func(void* pvParameter)
{
   while(true)
   {
      /*perform Harmony or application specific task
      job*/
      DRV_USART_Write(UART_1_Handle,
         (void*)pTaskA_str,
      (size_t)(strlen(pTaskA_str)) );
   }
}
```

## Task B

```
signed char* pTaskB_str = "Go Rattlers!";

void Task_B_Func(void* pvParameter)
{
   while(true)
   {
      /*perform Harmony or application specific
      task job*/
      DRV_USART_Write(UART_1_Handle,
         (void*)pTaskB_str,
      (size_t)(strlen(pTaskB_str)));
   }
}
```

# Operating System Abstraction Layer



# Multi-Threaded Harmony Demos

- **UART driver demo**
  - Two tasks try to access the same hardware UART peripheral to print out an ASCII string.  Each tasks prints its own unique string
  - Each tasks will run and try to print its own unique string, when its associated push button switch is depressed.
  - By connecting a USB cable from the starter kit to a PC, and opening an ASCII terminal emulator program (TeraTerm, RealTerm, etc), the user can see the ASCII strings being correctly printed.
  - Settings for the serial port connection:
    - 9600 baud, 8 data bits, 1 stop bit,  no hardware flow control

- **USB stack demo**
  - This demo is a take home but the concept is the same as the UART driver demo, just with a more complex piece of software. This demo shows the complexity of running multiple threads and using the Harmony USB stack in a thread safe manner.

# Demo Summary

- Demos show how a Harmony driver is setup and can operate correctly inside of a multi-tasking pre-emptive environment.
- In this case, FreeRTOS was chosen as the systems execution environment, due to the popularity of the RTOS and the open source model.

# Agenda

- Overview
- Getting Started + Hands-on Labs
- Adding Features/Functionality + Labs
- Application Migration + Lab
- Harmony RTOS Application + Demo
- **Third Party Ecosystem Solutions**
- Wrap-up, next steps

# Third Party Partners
# MPLAB® Harmony Ecosystem

30-minutes

# Ecosystem Partners

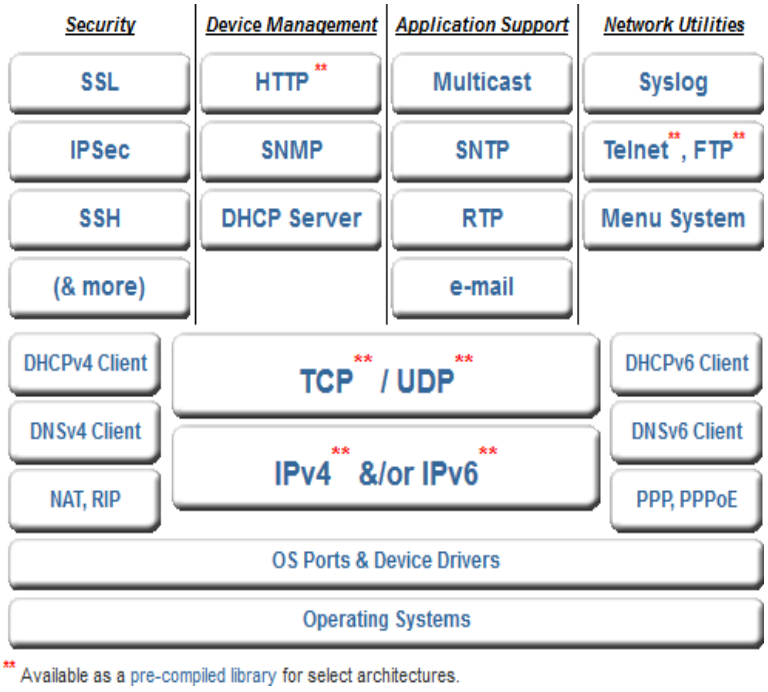
- InterNiche Technology, Inc.
- wolfSSL
- Real Time Engineers Ltd., FreeRTOS
- Wittenstein High Integrity Systems, OPENRTOS
- Express Logic Inc., ThreadX
- Segger, embOS (emWin planned)
- Micrium Inc., uC/OS-II and uC/OS-III

# Guiding Philosophy

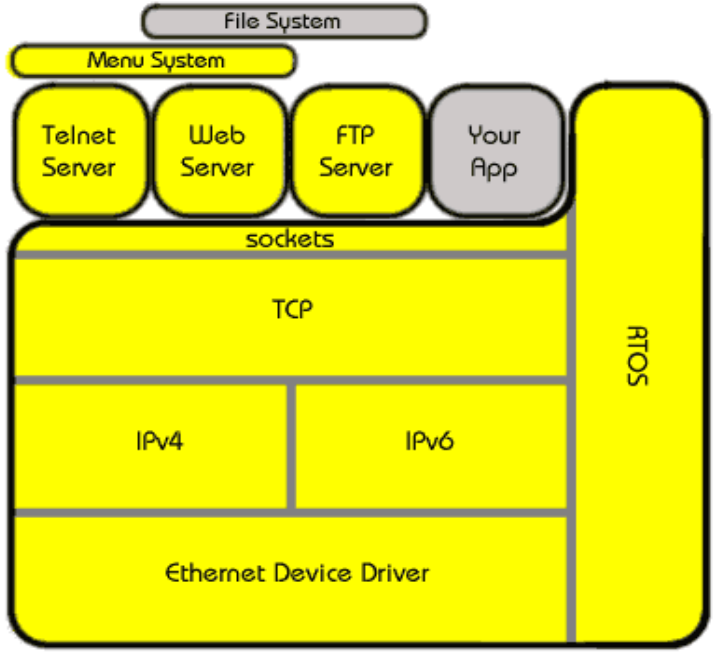
- Products are RFC compliant
- Products are NON-GPL
- Footprint generally more important than speed
- Technical Support is as valuable as are the licensed products
- Support includes assistance in product's use
- All products work together in all meaningful combinations
- Extensive technical documentation, including Command Reference tailored to each customer's order

# Two “Distribution Flavors”

## Source Code




## Pre-Compiled (5 modules)




www.iNiche.com

www.TCPIPStack.com

# Source Code - à la carte

- Licensed and Delivered via InterNiche
- ANSI “C”
- Many modules available
- Continually enhanced and updated
- SuperLoop and NicheTask™ included with stacks
- Supports any non-MPU RTOS (including homegrown)

# MHC Integration (Harmony)

- InterNiche's current products are Harmony Compliant
- Upcoming release will be fully integrated with MPLAB® Harmony Configurator (MHC)
- Available through microchipDIRECT, InterNiche and InterNiche distributors
  - Complete list at www.iniche.com

# Product Offerings - embLibs

- Pre-Compiled Libraries
  - Available through microchipDIRECT and other distributors
  - Node-Locked to specific PIC32 microcontrollers
  - Limited modules (TCP, DUAL, HTTP, FTP, Telnet)
  - Limited technical support
  - Inexpensive
  - Well documented
  - Useful programming examples included
- Machine dependencies provided in source code
  - PHY

# Serving the world through:

- Source Code availability through InterNiche, microchipDIRECT and regional distributors:
  - Wittenstein (EMEA)
  - IXXAT (Germany, Austria, Switzerland)
  - Complete list at http://www.iniche.com/distributors.php
- Libraries available through multiple online stores

# About wolfSSL


**Founded**: 2004

**Location:** Bozeman, MT
Seattle, WA
Portland, OR

**Our Focus**: Open Source Embedded Security (for Applications, Devices, and the Cloud)

**Products**:
- CyaSSL
- CyaSSL FIPS
- wolfCrypt
- wolfSSH
- wolfSCEP
- wolfSSL Inspection
- yaSSL

# CyaSSL Embedded SSL Library

## Features

- C-language based SSL/TLS library
- Standards up to **TLS 1.2** and **DTLS 1.2**
- Focused on size and speed optimization, progressive
- Minimum footprint size of **20-100 kB**
- Minimum RAM usage: **1-36kB**

- Web server integration (NGINX, Lighttpd, Mongoose, GoAhead)
- OpenSSL Compatibility Layer
- Hardware Crypto Support (including Microchip PIC32MZ)
- Suite-B Compatible, FIPS in process





wolfSSL

# CyaSSL Embedded SSL Library

## Algorithms

MD2, MD4, MD5, SHA-1, SHA-2, SHA-3, RIPEMD

DES, 3DES, AES, Camellia

ARC4, RABBIT, HC-128, ChaCha20

AES-GCM, AES-CCM, Poly1305

RSA, ECC, DSS, DH, EDH

HMAC, PBKDF2

Hash Functions

Block Ciphers
Stream Ciphers
Authenticated Ciphers
Public Key Options
Password-based Key Derivation

## Supported Environments

Win32/64, Linux, Mac OS X, Solaris, ThreadX, VxWorks, FreeBSD, NetBSD, OpenBSD, embedded Linux, Haiku, OpenWRT, iPhone (iOS), Android, Nintendo Wii and Gamecube through DevKitPro, QNX, MontaVista, OpenCL, NonStop, Tron/itron/microitron, Micrium's uC OS, FreeRTOS, SafeRTOS, Freescale MQX, Nucleus, TinyOS, HP/UX

**OR**

Bare Metal / No OS





wolfSSL

# wolfCrypt Embedded Crypto Engine

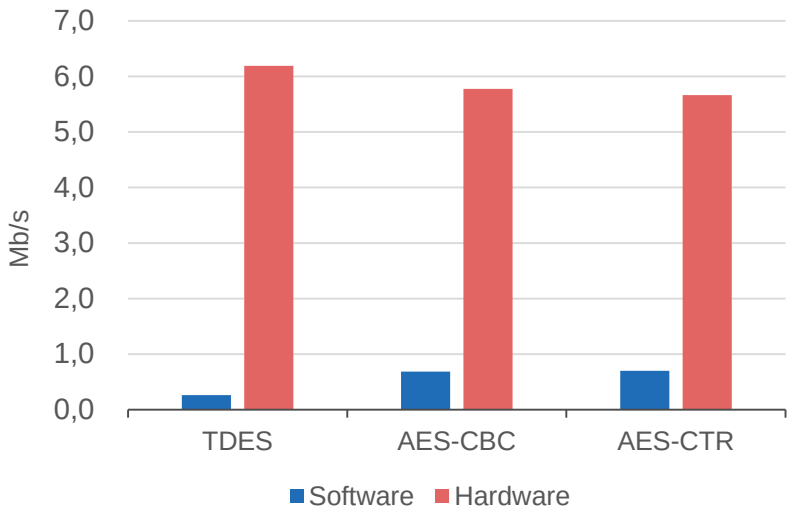
## Features

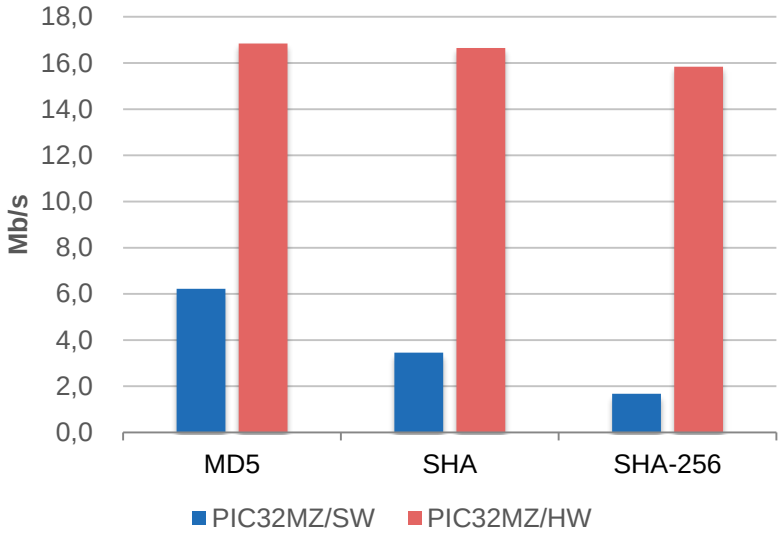
- ECC, up to 521-bit
- Hash-based PRNG
- PIC32MZ, AES-NI, Cavium, STM32, Kinetis, hardware crypto support
- Progressive list of supported ciphers (see previous slide)
- Key and certificate generation
- Small footprint, low runtime memory
- Modular design, assembly optimizations

# CyaSSL / wolfCrypt Hardware Cryptography

## Hardware Crypto on PIC32MZ

- Random Number Generation
- Hash Functions (MD5, SHA, SHA-256)
- Block Ciphers (AES, DES, 3DES)



# Why CyaSSL - wolfSSL?

- CyaSSL has been written from the ground up. wolfSSL owns the Copyright.
- Built for portability, modularity and performance.
- Strong, collaborative partnership with MICROCHIP.
- Dedicated Support via support @wolfssl.com and direct phone support.
- Free Presales Support!
- Commitment to new Ciphers, new Features and addressing ongoing security threats!  Microchip customers have a viable solution for SSL/TLS.

# Who Are We?

FreeRTOS is owned, developed, maintained, distributed and supported by **Real Time Engineers Ltd**.

Real Time Engineers Ltd. have been working in close partnership with the world's leading chip companies for more than 12 years to provide award winning, commercial grade, and **completely free** high quality software that carries no risk of IP infringement

# Technical Strengths

- True real time behaviour
- Targets mid powered 32-bit processors, such as the PIC32, right down to 8-bit microcontrollers, with 35 officially supported individual architectures, and 18 officially supported tool chains
- ROM/Flash footprint from 5K to 12K bytes
- RAM footprint ≈250 bytes, RAM is also required for task stacks

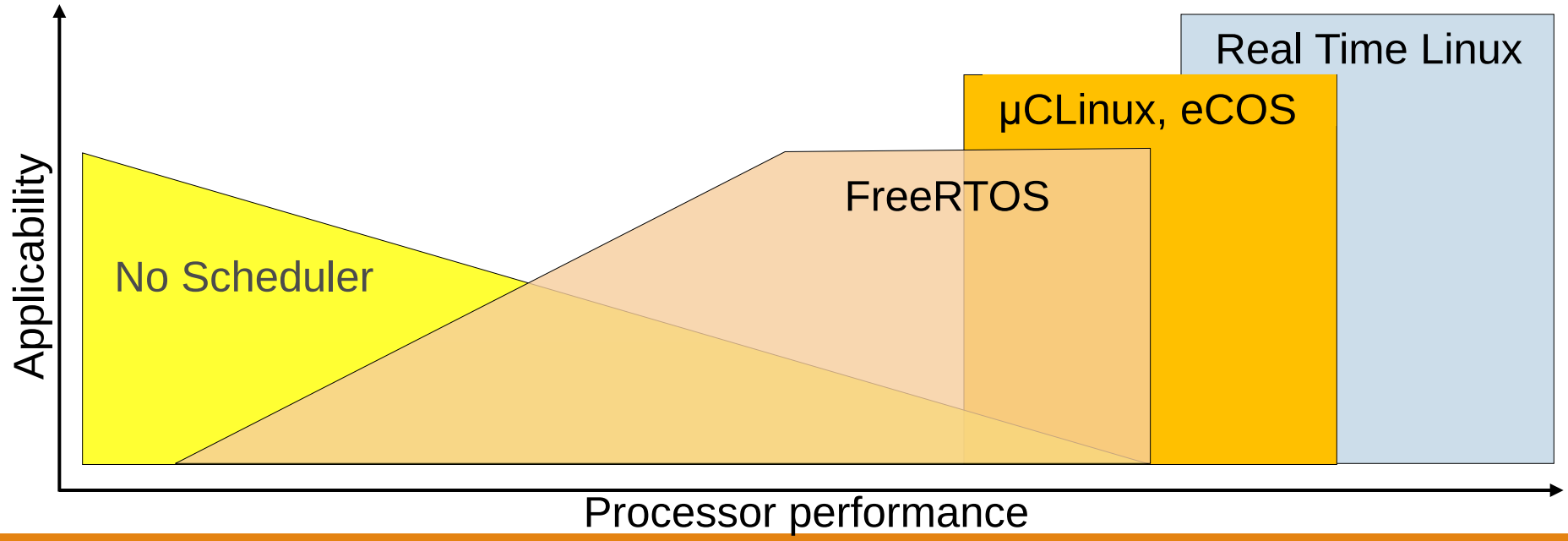
# Ease of Use

Each port is distributed with at least one pre-configured and buildable example, ensuring the best 'out of the box' experience possible

Pre-configured examples can be used as a starting point for new applications

Developer learning resources:

- Quick start guide
- Tutorial books
- Professional training services
- Asserts to catch common mistakes



# Quality at Every Level

- The authors are familiar with aerospace grade processes and procedures
- The core source code conforms to the MISRA coding standard
- The source code uses a strict style guide
- Selected FreeRTOS versions have passed mathematical verification using formal methods
- SAFE**RTOS** is a TÜV safety certified version available from our official safety engineering partner – WITTENSTEIN high integrity systems, a global engineering company with >1500 employees

# Why FreeRTOS?

FreeRTOS offers lower project risks and a lower total cost of ownership than commercial alternatives because:

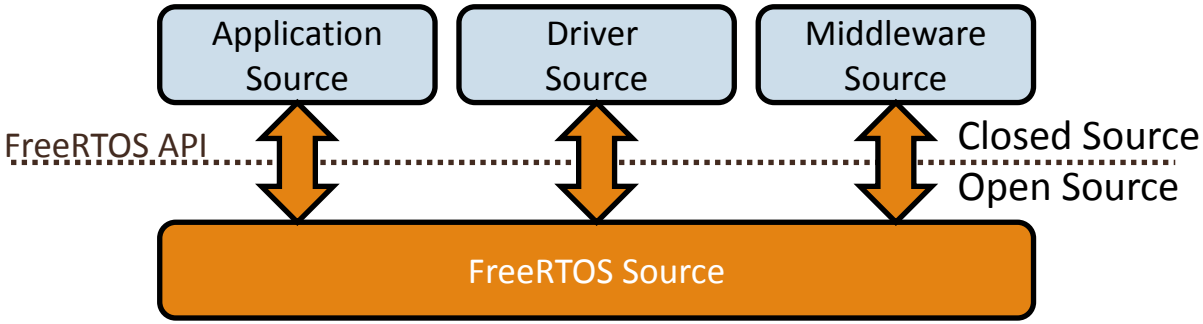
- It is fully supported and documented
- Most people take products to market without ever contacting us
  - But with the complete peace of mind that they could opt to switch to a fully indemnified low cost commercial license (with dedicated support) at any time throughout their product's lifecycle
  - Commercial licenses for FreeRTOS are provided under license from Real Time Engineers Ltd. by WITTENSTEIN high integrity systems

# FreeRTOS Licensing and Support

**Moderated** open source – Real Time Engineers Ltd. wrote and own 100% of the official code to remove the risk of accidental IP infringement

**Modified** GPL license – the modification removes the requirement to open source code that links with FreeRTOS

Free forum support, direct from Real Time Engineers Ltd.



# Technical Summary

Prioritised scheduling
- Pre-emptive option
- Co-operative option
- Time slicing option

Tickless mode for low power

Excellent and extensible execution trace capability

Multiple kernel aware IDE plug-ins

Flexible and easy to use queues

Binary, counting and recursive semaphores

Mutex with priority inheritance

Event groups and flags

Software timers

Stack overflow detection

Selection of memory allocation schemes

Many ports never completely disable interrupts

No ports perform non-deterministic operations with interrupts disabled


# OPENRTOS

From WITTENSTEIN high integrity systems (WHIS)

**We provide:**
OPEN**RTOS** - a fast, lightweight, small footprint Real Time Operating System, supplied with one year's free support as standard, and with integrated middleware available.

WHIS can also provide customisation services, consultancy and training.

**Small and Fast**
Typical ROM requirements 4-9K bytes
Typical RAM requirements 250 bytes
Typical Stack requirements 400 bytes/task

**Proven & Robust**
Stable solution for high-availability applications.
Millions of product deployments worldwide.

**Simple**
Easy-to-program environment for developers familiar with C and C++.

## About WITTENSTEIN high integrity systems

Experts in embedded RTOS and Middleware technology with a specialisation in safety certified software. Supplying advanced RTOS and Middleware components across a broad range of market sectors and applications, from basic embedded designs, up to complex safety systems demanding the highest levels of certification.

WHIS is part of WITTENSTEIN Group, a technology company founded in 1948 with a presence in over 40 countries.

## WHIS Highlights

- RTOS Ecosystem of FreeRTOS, OPEN**RTOS** & SAFE**RTOS**
- FreeRTOS, Top-in-class in the 2011, 12, 13 & 14 EETimes embedded systems surveys in two areas: RTOS currently used, and RTOS considered for the next project
- Produced SAFE**RTOS,** the first RTOS to be pre-certified to IEC 61508 Safety Integrity Level 3 by TÜV SÜD, the highest possible for a software only component.
- SAFE**RTOS** was the first pre-certified safety critical RTOS to be ROM'ed in a main line processor.

**Wittenstein Group HQ**

## FreeRTOS, OPENRTOS

WHIS are the commercial partners of the FreeRTOS project from Real Time Engineers ltd. FreeRTOS is the market leading Real Time Operating System distributed under a modified GPL license, and is free to download and use within commercial applications.

OPEN**RTOS** shares the same code base as FreeRTOS. However OPEN**RTOS** is supplied with a full commercial license that **removes** the restrictions of the FreeRTOS modified GPL license.

## Free/OPENRTOS Highlights

- Supports 34 processor architectures & 18 toolchains
- Downloaded 107,000 times in 2013.
- Millions of product deployments

FreeRTOS™ is a market leader, with over 100,000 downloads per year — FreeRTOS

We added commercial licensing, middleware, support & indemnification — OPEN**RTOS**®

We rebuilt for the **safety sector**, adding multiple **certification standards** — SAFE**RTOS**®

We provide a leading RTOS ecosystem that has achieved global recognition

## OPENRTOS Licensing

**Preserve your anonymity with OPENRTOS**

OPEN**RTOS** has the same codebase as FreeRTOS, but it's licensing removes the FreeRTOS requirements listed below- providing complete confidentiality, IP indemnification, as well as peace of mind with the included free one year's professional support.

> **If you're using FreeRTOS in your application you must:**
>
> - Offer to provide the FreeRTOS code to all of the users of your application;
> - Open source any changes that you make to the FreeRTOS Kernel;
> - Publicize that you are using FreeRTOS.

## OPENRTOS Features

- Pre-emptive, Co-operative, Round Robin or Hybrid scheduling options.
  - Provides flexibility to select the scheduling algorithm that best fits your processor loading profile.
  - Priority base, pre-emptive scheduling provides excellent results for systems that exhibit high peak processor loading, such as communication systems.
- Any number of tasks can be created - system RAM constraints are the limiting factor.
- Each task is assigned a priority - any number of priorities can be used.
  - The higher the relative priority, the more responsive the Task becomes.
- Queues are used to safely send data between tasks, and to send data between interrupt service routines and tasks.
- Binary semaphores and counting semaphores make use of the queue primitive – ensuring code size is kept to a minimum.
  - Used for synchronization between tasks or between tasks and an interrupt
- Mutexes with priority inheritance and Recursive mutexes.
  - Used to provide mutual exclusion of resources
- Software Timers and Event Groups.

## Ease of Programming

- Kernel core supplied in 4 source files.
- Concise and well documented API with only 101 function calls.
- High quality source code, complying with MISRA coding standards.
- Lots of online supporting material and usage examples.

## Robustness

- Free/OPEN**RTOS** has a very large user base, downloaded over 107,000 times alone last year.
- Free/OPEN**RTOS** is probably the most peer reviewed RTOS in the world.
- Confidence assured by the activities undertaken by the SAFE**RTOS** sister project.

## OPENRTOS Benefits

- RTOS kernel **< 10Kb**
- Pre-emptive Priority Based Task Scheduling which results in:
  - Greater use of limited processor resources.
  - Efficient management of variable load systems with high peaks of activity.
  - Improved responsive times.
- Low overhead:
  - Context switch operations as low as 84 cycles on embedded processors.
  - Imperceptible boot time.

# Why ThreadX

- **Advanced Technology** *– Unique and unparalleled*
- **Easy to Use** –*This means faster time-to-market and lower development cost.*
- **Full Source Code** –*This means lower support cost.*
- **Fast** - *This means better real-time performance from a less expensive processor.*
- **Small** - *This means lower memory cost and smaller physical size.*
- **Tools Integration** - *This means faster time to market.*
- **Royalty Free** - *This means lower production cost.*
- **Proven** - *This means low development risk.*

---

- **Great Value** - *This means lower costs, greater profit.*

# Time To Market

| | Ind ave | ThreadX | Micrium | Integrity | VxWorks | LynxOS |
|---|---|---|---|---|---|---|
| **Devel time Months** | 13.9 | 10.5 | 11.3 | 17.9 | 16.3 | 13.7 |
| **% on/ahead of schedule** | 53% | 70% | 60% | 57% | 49% | 64% |
| **Months behind** | 3.8 | 3.1 | 3.7 | 3.6 | 3.3 | 2.5 |
| **Ave Delay Months** | 1.79 | 0.95 | 1.48 | 1.55 | 1.69 | 0.91 |
| **Percent delay/number of developers** | 12.1% | 19.9% | 26.4% | 10.2% | 11.9% | 7.4% |
| **Delay/proj time** | 12.8% | 9.1% | 13.1% | 8.6% | 10.3% | 6.6% |
| | | | | | | |
| **SW Developers/project** | 14.7 | 4.8 | 5.6 | 15.2 | 14.2 | 12.3 |
| **Average Developer months/project** | 204.3 | 50.4 | 63.3 | 272.1 | 231.5 | 168.5 |
| **Developer months lost to schedule** | 26.3 | 4.6 | 8.3 | 23.5 | 23.9 | 11.2 |
| | | | | | | |
| **Total developer months/ project** | 230.6 | 55.0 | 71.6 | 295.6 | 255.4 | 179.7 |
| | | | | | | |
| **At $10,000/developer month** | | | | | | |
| **Average developer cost/project** | $2,043,300 | $504,000 | $632,800 | $2,720,800 | $2,314,600 | $1,685,100 |
| Average cost to delay | $262,542 | $45,830 | $82,880 | $235,296 | $239,455 | $111,930 |
| | | | | | | |
| **Total developer cost/project** | **$2,305,842** | **$549,830** | **$715,680** | **$2,956,096** | **$2,554,055** | **$1,797,030** |

**Percent of Final Design outcomes within 20% of pre-design expectation - Excellent (best) Outcome**

| **FOR:** | **Micrium** | **ThreadX** |
|---|---|---|
| **Performance** | 71.0% | 81.3% |
| **Systems Functionality** | 58.1% | 81.3% |
| **Features & Schedule** | 48.6% | 75.0% |

*Source: 2012 EMF Developer Survey*

# Advanced Technology

## ThreadX Technology

- **Complete Multithreading Facilities**
  - Threads
  - Application Timers
  - Message Queues
  - Counting Semaphores
  - Mutexes
  - Event Flags
  - Block and Byte Memory Pools
- **Priority-based Preemptive Scheduling**
  - Real-time response to external events
- **Round-Robin Scheduling**
  - For threads of equal priority
- **Fully Deterministic Performance**
  - ThreadX performs in a consistent, fast manner regardless of the number of threads or other objects

- **Preemption-Threshold™ Scheduling**
- **Event Trace**
- **Event Chaining™**
- **Performance Metrics**
- **Run-time Stack Analysis**
- **Optimized Timer Interrupt Processing**
- **Priority Flexibility**
- **Priority Inheritance**
- **FIFO and Priority Thread Suspension**

# Small Size

| ThreadX Services | Typical Size In Bytes |
|---|---|
| Core Services (required) | 6,000 |
| Queue Services | 1,300 |
| Event Flag Services | 1,000 |
| Semaphore Services | 500 |
| Mutex Services | 1,200 |
| Block Memory Services | 600 |
| Byte Memory Services | 1,000 |

*ThreadX can be as small as 2.0KB*

*Typical size is only 6KB – 15KB*

# High Performance

| ThreadX Service | Service Time | Suspend | Resume | Context Switch |
|---|---|---|---|---|
| Suspend | 0.56us | 0.84us | - | - |
| Resume | - | - | 0.56us | 1.1us |
| Queue Send | 0.40us | 1.1us | 0.86us | 1.4us |
| Receive | 0.34us | 1.2us | 0.96us | 1.5us |
| Get Semaphore | 0.16us | 0.96us | - | - |
| Put Semaphore | 0.18us | - | 0.64us | 1.2us |
| Context Switch | | | | 0.38us |
| Interrupt Response | 0.0us (min) | | | 0.36us (max) |

*Timings based on 200MHz processor*

# Low Cost

- **Royalty-Free Pricing**
  - No per-unit royalties for deployed copies of RTOS
- **Small one-time project license fee**
  - License fee independent of production units
- **Ideal for high-volume products**
    - No accounting or reporting of shipments
    - Reduces manufacturing cost
- **Field Proven**
  - Proven reliability and bullet-proof performance
    - Over 2 billion units prove reliability
  - Lower support costs
    - No surprises, no bugs found in field
  - Lower risk of product recall, returns
    - Proven to work in the field
  - Lower warranty contingency cost/risk
    - Fewer, less likely returns, failures = lower cost

# Resource Saving RTOS Kernel – embOS

## Very small footprint

- only half the size of most competitive products

  ROM: <<2 kB, RAM: <<1kB (Kernel only)

## True Zero Interrupt Latency

- No additional latency introduced by e

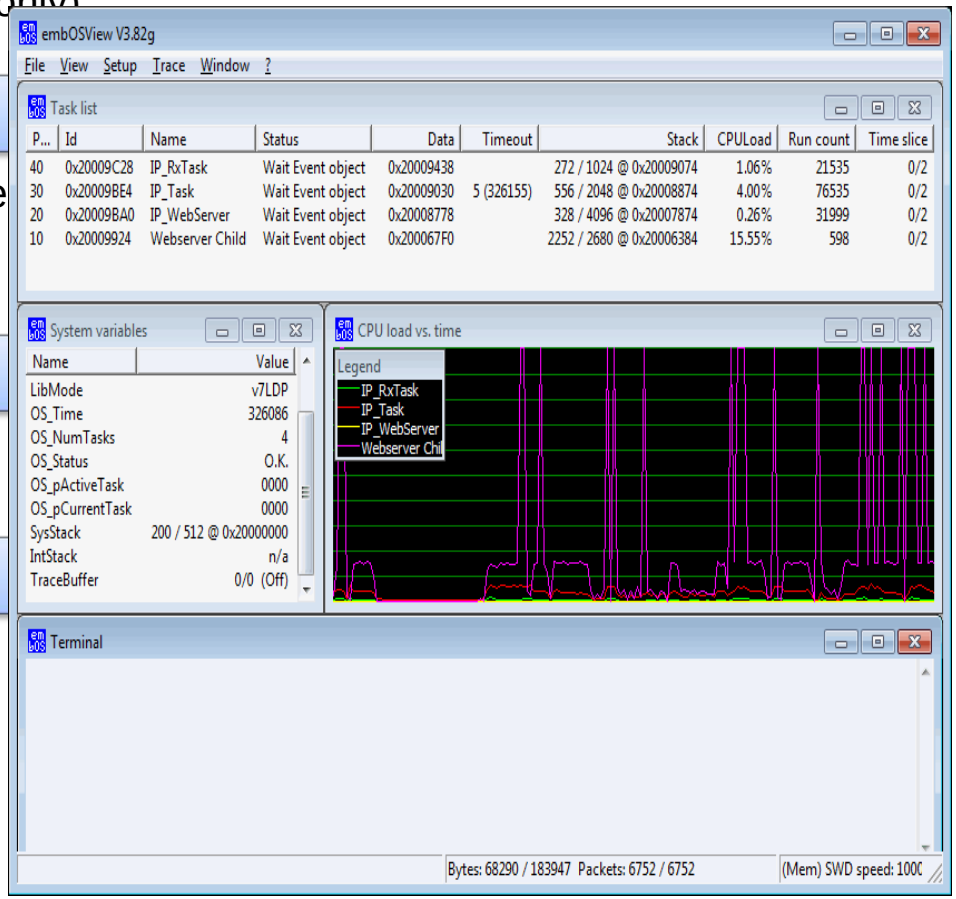
  for zero latency interrupts

## Fast context switch

- Independent from number of tasks

## Free profiling tool included

- embOSView

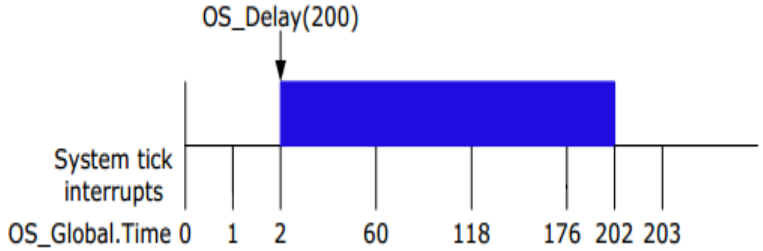


# embOS Tickless operation

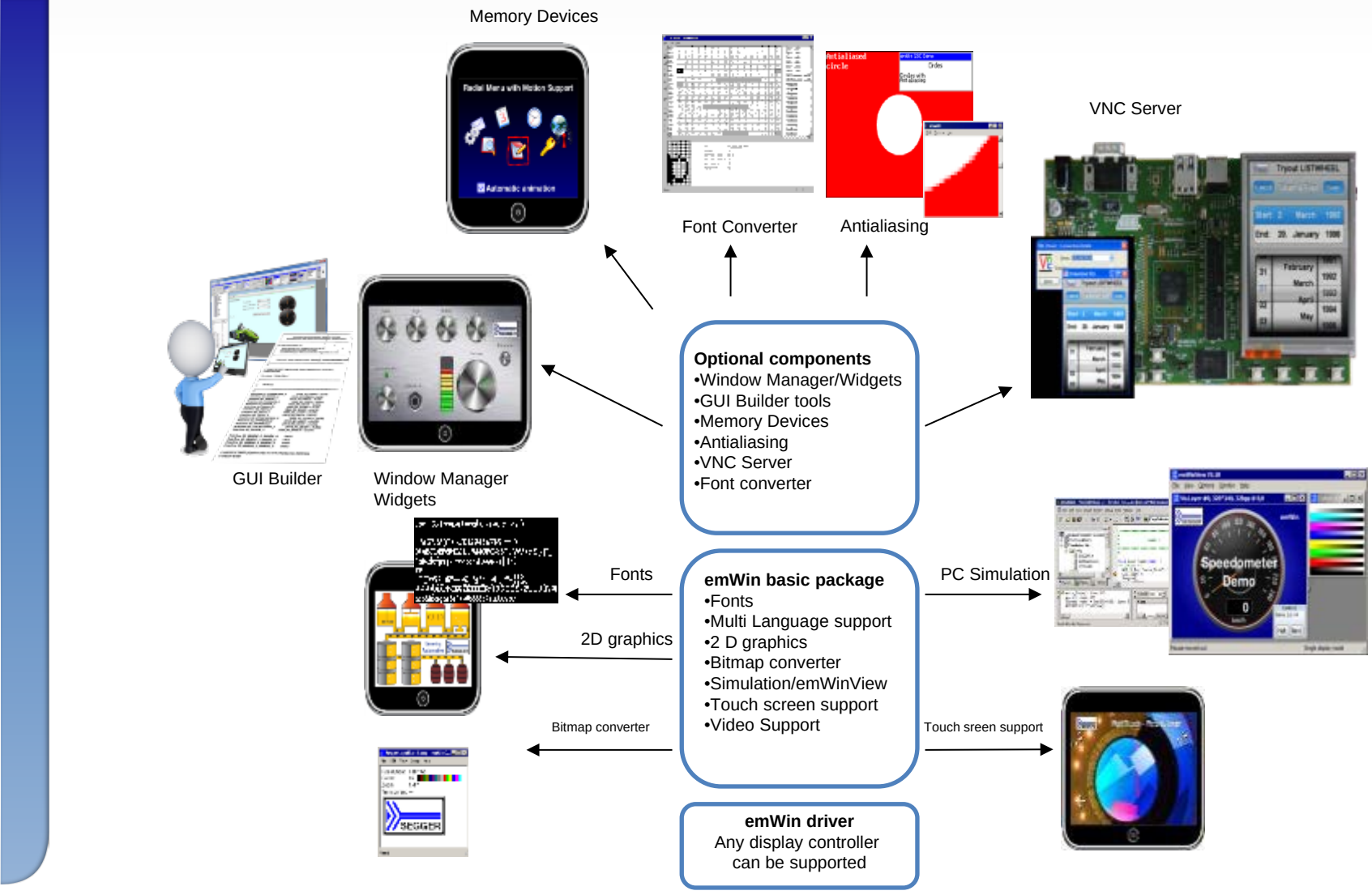

The embOS tickless low power support reduces power consumption by creating a variable length system tick which allows the processor to continue sleeping if there is nothing to do.

This can make a huge difference in battery life.

Instead of having 200 interrupts (each one for each system tick) we need only 4 interrupts.



# Small Footprint Quality GUI – emWin



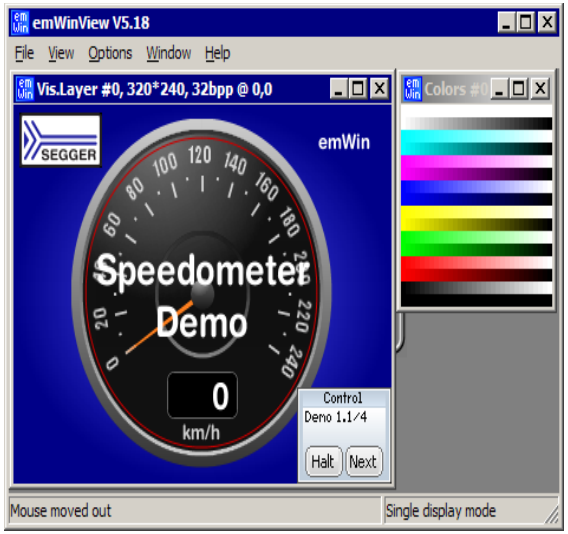
# Graphical Debugging & Motion/Gesture Support

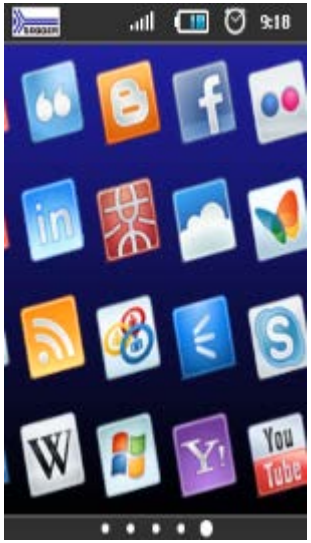
## Graphical debugging with emWinView

- Allows Single Draw Operations while stepping through code
- View individual Layers
- Composite View of Layers
- Magnification of each Layer
- Open Multiple Views at the same time
- Displays Virtual Pages

## Motion and Gesture Support for Touch Screens

- Full Screen Slide Effects
- On Screen Motion Effects
- Panning, Zooming, and Rotation
- Multi-Touch



# embOS & emWin

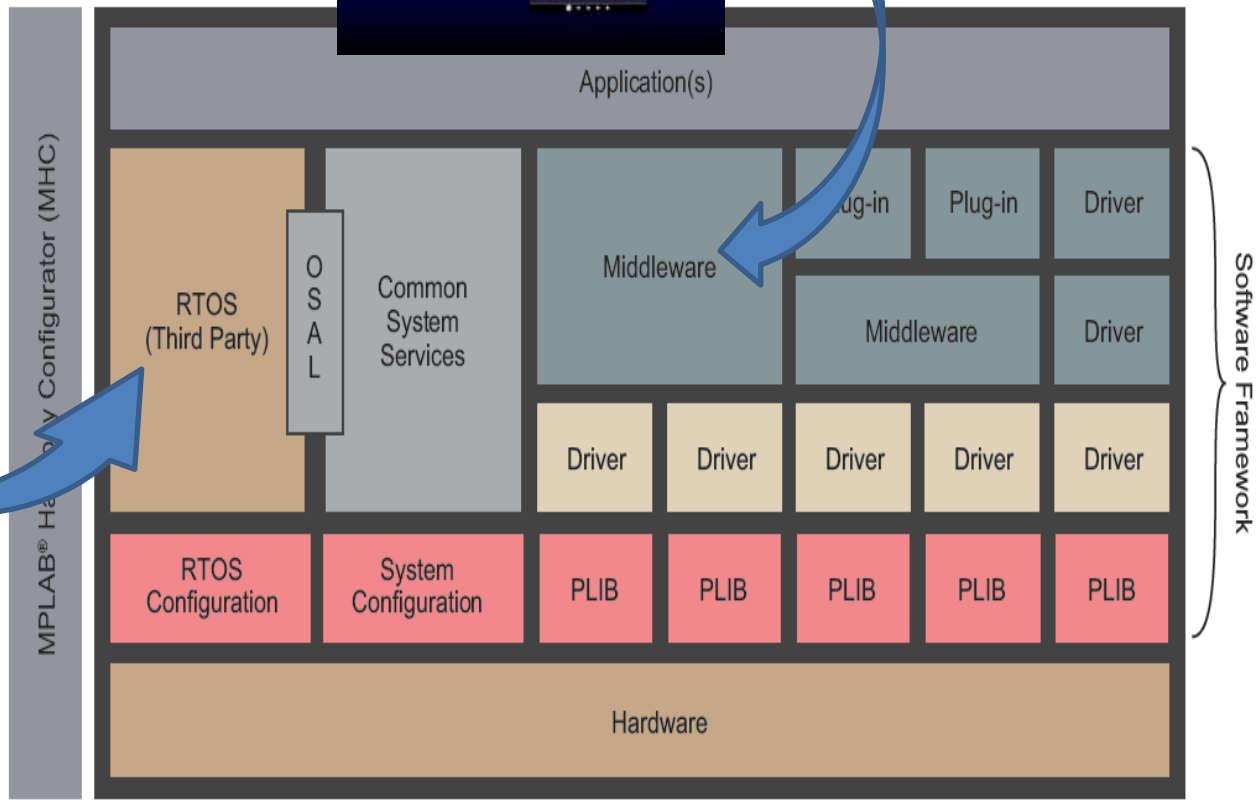
www.segger.com

MPLAB HARMONY
Integrated Software Framework
FASTER PIC32 DEVELOPMENT WITH FEWER RESOURCES

Harmony support for embOS available NOW!




Harmony support for emWin available 2015



# An Overview of Micriμm

- Incorporated in 1999
- Headquarters in South Florida, with an engineering office in Montreal


- Provider of high-quality embedded software
- Known for clean code, thorough documentation, and top-notch technical support

Micriμm

# Micriμm's Wide Market Reach

## Medical

PHILIPS
BAYER
SIEMENS medical
Hill-Rom
Abbott Medical Optics
Medtronic
invitrogen
RESPIRONICS

## Aerospace / Automotive

BOEING
AIRBUS
GUIDANT
Lear
THALES
Rockwell Collins
LOCKHEED MARTIN
NORTHROP GRUMMAN

## Communications

CISCO
MOTOROLA
NORTEL NETWORKS
BUSINESS WITHOUT BOUNDARIES
tyco
freescale semiconductor
BROADCOM
PMC PMC-SIERRA

## Industrial / Home

Honeywell
JENN-AIR
SAFELINE
METTLER TOLEDO
invensys
APC

## Consumer

LG
WD Western Digital
harman/kardon
WESCAM
lexicon
LINE 6
ANTHEM

# Micriμm's RTOS Model

- Micriμm offers a complete RTOS
  - Kernel, file system, GUI, and protocol stacks
  - All modules written for embedded systems
    - Everything fits in less than 1 MByte

- Each component is sold separately


- Model facilitates streamlined application development

## μC/OS-II

- Track record of over 16 years
- Suitable for the most demanding safety-critical applications
  - DO-178B (Level A)
  - IEC-62304
  - IEC-61508

## μC/OS-III

- Full-featured kernel offering round-robin scheduling and deferred interrupt handling
- Built-in performance measurement capability

# A History of PIC32 Support

- Micriμm's first PIC32 projects date to the original release of the family's devices
- MPLAB has served as the basis for all of Micriμm's PIC32 examples
- An MZ port was recently added to the well-established μC/OS-III PIC32MX port

# uC/OS Now Available in Harmony

- **μC/OS-II and μC/OS**-III sources integrated into Harmony
- The kernel's full source code can be evaluated at no cost
- This code can also be used free of charge in academic projects
- Developers planning to use the code to develop a product, however, must purchase a license
- Additional licensing information can be obtained from **Micriμm**

# Agenda

- Overview
- Getting Started + Hands-on Labs
- Adding Features/Functionality + Labs
- Application Migration + Lab
- Harmony RTOS Application + Demo
- Third Party Ecosystem Solutions
- **Wrap-up, next steps**

# Recap : Key Benefits

- **Cross-Micro Compatibility**
  - Common, consistent APIs
  - Easier to grow into larger parts
  - Easier to shrink into smaller parts
- **Code Interoperability**
  - Drivers and libraries work together with minimal effort
  - Applications port to different boards with minimal effort
- **Faster Time to Market**
  - Able to develop applications more quickly
  - Able to easily add features
- **Microchip More Responsive to Customers**
  - Field & Apps teams able to develop applications more quickly
  - Reduced support burden frees resources for new development
- **Improved Satisfaction**
  - Eliminates conflicts
  - Improved code quality

# MPLAB® Harmony – Features

## Want to know what's inside?
## Check out the Features tab on the Harmony Page

Features | Downloads | Archived Downloads | Documentation | Frequently Asked Questions (FAQs)

Back To Top

### MPLAB Harmony Features

| Feature | Description |
|---|---|
| TCP/IP Network Stack and WiFi support | The MPLAB Harmony TCP/IP Stack provides a foundation for embedded network applications by handling most of the interaction required between the physical network port and your application. It includes modules for several commonly used application layers, including HTTP for serving web pages, SMTP for sending e-mails, SNMP for providing status and control, Telnet, TFTP, Serial-to-Ethernet, and much more. In addition, the stack includes light-weight and high-performance implementations of the TCP and UDP transport layers, as well as other supporting modules such as IP, ICMP, DHCP, ARP, and DNS.<br><br>The Wi-Fi software library, in conjunction with the MRF24WG0MA module, allows an application to:<br>▪ Join an existing 802.11 Wi-Fi network<br>▪ Create a 802.11 Wi-Fi network |
| USB Device Stack | The MPLAB Harmony USB Device Library (referred to as the USB Device Library) provides embedded application developers with a framework to design and develop a wide variety of USB Devices. A choice of Full Speed only or |

# MPLAB® Harmony – Resources


**www.Microchip.com/Harmony**

- **Download Harmony**
- **Getting Started with MPLAB® Harmony**
- **Documentation**
- **Release Notes**
- **Third Party Solutions**
- **Forum/Support**

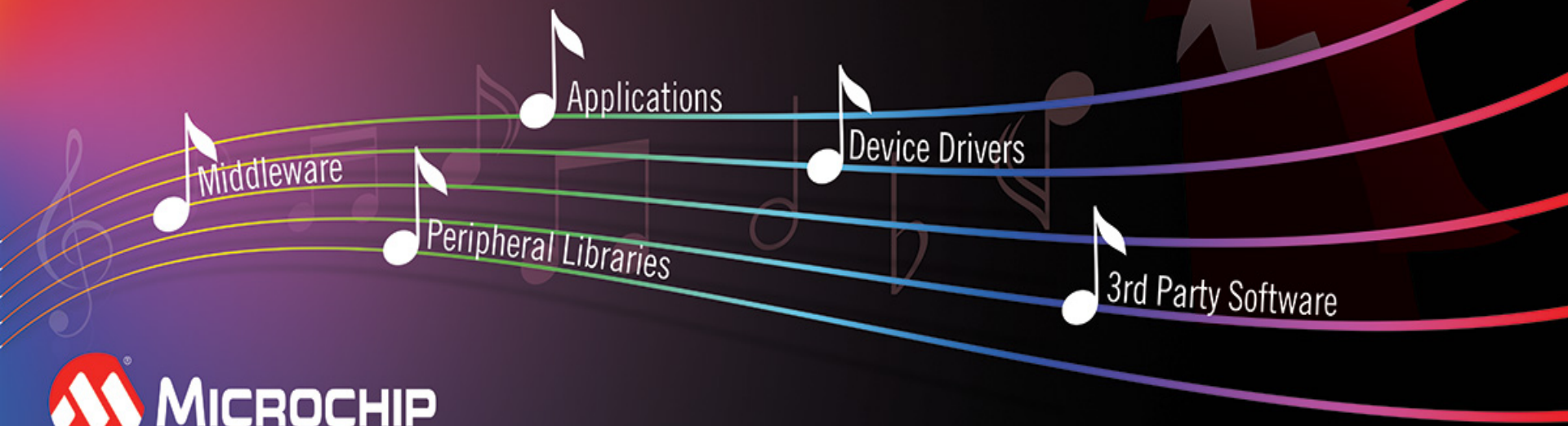
# Questions?

# BE THE MAESTRO!

MPLAB® HARMONY

FASTER PIC32 DEVELOPMENT WITH FEWER RESOURCES

***Thank You!***




MICROCHIP

MICROCHIP.COM/HARMONY

# C Language Refresher



```
typedef enum _app_states
{
    APP_STATE_INIT = 0,
    APP_STATE_EVENT,
    APP_STATE_IDLE

} APP_STATES;

typedef struct _app_data
{
    APP_STATES state;
    int        count;
    char       command;

} APP_DATA;

APP_DATA app1Data, app2Data;

app1Data.state = APP_STATE_INIT;
app2Data.state = APP_STATE_INIT;
```

## "enum" Types

- Used to give names to constants
- Better than "#define"
  - Groups related names together
  - Associates names with a type
  - Values auto-generated
  - Improves type checking
  - Improves debugging

## "struct" Types

- Groups related variables
- Better than separate variables
  - Access member/field with '.'
  - Avoids name collisions

`app1Data.state` different from `app2Data.state`

- Reference struct by one pointer:

```
APP_DATA *pApp = &app1Data;
pApp->state = APP_STATE_IDLE;
```

# Dev. Tools needed for the Hands-on labs

- **MPLAB® X IDE v3.00**
- **MPLAB® XC32 C/C++ Compiler v1.34**
- **MPLAB® Harmony v1.04.02**
- **Select one PIC32 MZ Starter Kit:**
  - DM320006 - PIC32MZ Embedded Connectivity Starter Kit **OR**
  - DM320006-C - PIC32MZ Embedded Connectivity Starter Kit w/Crypto Engine
- **DM320005-2: Multimedia Expansion Board II (MEB II)**
- **AC320101: USB-to-UART converter board**
- **9V power supply**
- **mini USB-to-USB cable**
- **USB Pen Drive**
- **Windows® XP or Windows® 7 PC with two USB ports**
  - Apple MAC & Linux will be supported from Harmony v1.04
- **Tera Term or other terminal emulator program**
- **Instructor Only (for demo labs)**
  - DM320003-2: PIC32 USB Starter Kit II
  - DM320005, Multimedia Expansion Board (MEB)

# 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.

**TRADEMARKS:**

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

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

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

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

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

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