PIC32MZW1 Software User Guide

Introduction

This document describes the software features supported by PIC32MZ W1 Family using the MPLAB® Harmony v3 framework. This framework is integrated with a development environment that works directly with the WFI32E01 module. The software release package enables a rich set of PIC32MZ1025W104 SoC features such as 802.11 b/g/n, Ethernet, USB, CAN, CAN-FD, SPI, I2C, SQI, UART, and JTAG, which are supported by the PIC32MZ W1 Family.

Features

Harmony v3 Peripheral Support

- Three UART modules (one high-speed UART with dedicated pads and two user configurable UART using Peripheral Pin Select)
- I2C Master and Slave with Address Masking
- Two SPI Ports (one dedicated high-speed SPI and one user configurable SPI using PPS)
- One USB OTG 2.0 (full-speed, device mode CDC console only)
- 35 Remappable GPIOs using PPS
- One Fast Ethernet (10/100) Reduced Media Independent Interface (RMII)
- Seven Timers
  - 16-bit timers/counters can be concatenated to form a single 32-bit timer
- Eight Channel Hardware DMA (Direct Memory Access) Controller with Automatic Data Size Detection supporting 32-bit CRC-checked Transfers of up to 64 Kbytes in size
- Node Version Manager (NVM) Read/Write Support
- Watchdog Timer (WDT) for Fail Safe Operations
- 20 External Analog Inputs for Sampling/Conversion
- Four Output Compare Ports

Harmony v3 System Services Support

- Clock Support up to 200 MHz
- Four On-chip Integer PLLs:
  - USB (UPLL)
  - Ethernet/Wi-Fi (EWPLL)
  - System (SPLL)
  - Bluetooth (BTPLL) (unused)
- Power-up Timer (PWRT) and Oscillator Start-up Timers (OST)
- Interrupts Enabled through Peripheral Libraries
- System Console for User Debug Log Messages
- Reset Source Selections:
  - Power-on Reset (POR)
  - Master Clear Reset (MCLR)
  - Software Reset (SWR)
  - Brown-out Reset (BOR)
- Four System Ports (A, B, C and K)
- Device Configuration (DEVCON) for all Peripheral-related Bits including Clocks, Programming and Debugging

Harmony v3 WLAN/ Networking Support

- MPLAB Harmony v3 TCP/IP Stack
- WLAN STA and AP Networking Modes
  - TLS v1.2 with symmetric crypto acceleration
  - DHCP client/server, DNS client/server, ICMPv4, iPerf
- Wireless Network Security Standard (WEP, WPA/WPA2-Personal, and WPA3)
- Protected Management Frames (802.11w)
- Wi-Fi Transmit Power Control
- Configurable Region Selection for Regulatory Compliance

Table of Contents

1. Quick References

1.1 Reference Documentation

For further study, refer to the following:

PIC32MZ1025W104 MCU and WFI32E01 Module with Wi-Fi® and Hardware-based Security Accelerator Data

Sheet (DS70005425)

PIC32WFI32E Curiosity Board User's Guide (DSxxxxxxxx)

Note: For Reference Manuals, refer to **Documents** page of the http://www.microchip.com/xxx (TBD-xref).

1.2 Hardware Prerequisites

PIC32WFI32E Curiosity Board Evaluation Kit

MPLAB ICD3 Programmer/Debugger (optional)

1.3 Software Prerequisites

MPLAB Integrated Development Environment (MPLAB X IDE) tool (version 5.35)

MPLAB XC32 compiler (version 2.40)

Terminal emulator utility program (Tera Term)

2. Functional Overview

- Peripheral libraries
  - Refer to 3. Peripheral Libraries for the list of peripheral libraries supported by PIC32MZ W1 Family.
- Core examples
  - Refer to 4. Core Examples for the list of core examples supported by PIC32MZ W1 Family.
- Third party libraries
  - Refer to 5. Third Party Libraries for the list of third party libraries supported by PIC32MZ W1 Family.
- MPLAB Harmony Configurator
  - Refer https://github.com/Microchip-MPLAB-Harmony/mhc/wiki for more details.
- WLAN Functional APIs
  - A set of functions providing abstracted control plane functionality to the application. This includes the following functionality:
    - Scanning and network discovery
    - Connection/disconnection to an AP (STA mode)
    - AP enable/disable and configuration (AP mode)
    - Support for implemented security configurations
    - Power control
    - Channel/region configuration

2.1 Hardware Setup

This section describes the hardware setup using the PIC32WFI32E Curiosity Board Evaluation Kit.

2.1.1 Power Supply

The PIC32WFI32E Curiosity Board can be powered using any of the following sources:

- Type A male-to-micro-B USB Cable (from a host PC)
- External +5V Supply

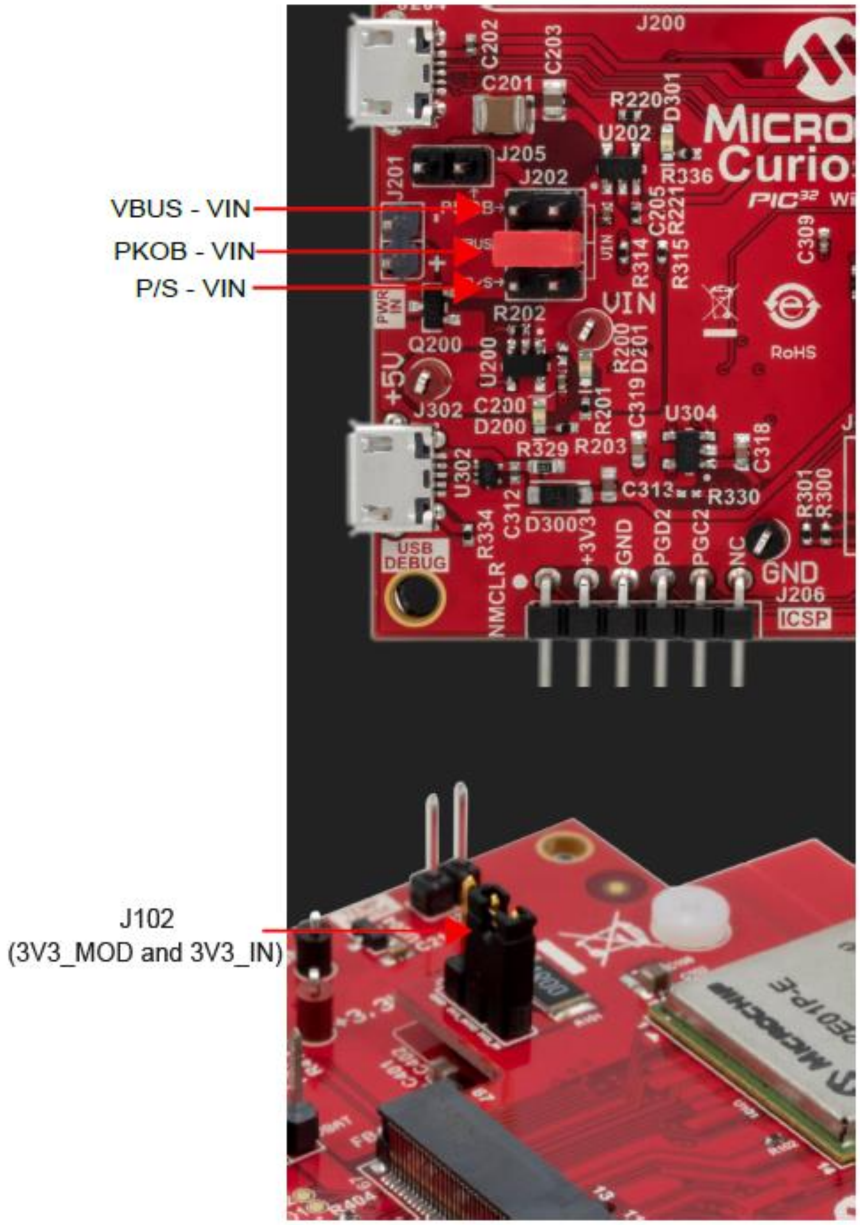
J202 jumper is used to select the voltage source for the curiosity board. The following table provides the power supply source details and its jumper positions. The MCP1727 voltage regulator generates +3.3V power supply for the MCU.

Table 2-1. Power Supply Sources

Power Input	Connector Type	Jumper Position (J8)
External 5V (J201)	Connect the development board to an external 5V power supply.	P/S-VIN (2-1)
USB micro-B (J204)	Using a Type A male-to-micro-B USB cable	VBUS-VIN (6-5)
USB micro-B (J302)	Using a Type A male-to-micro-B USB cable	PKOB-VIN (4-3)

The following figure shows the jumper positions for powering the curiosity board.

Figure 2-1. Jumper Configuration for Power Output



Note: Ensure that the 3V3_MOD and 3V3_IN of J102 is connected.

When PKOB debugger is connected to a host PC, the power supply (+3.3V and +5V) to the curiosity board is turned on via a power switch (MIC2005A) which drives the ENABLE signal high. The following figure shows the power tree diagram for the curiosity board.

2.1.2 Programming and Debugging

The PIC32WFI32E Curiosity Board has the PKOB3 debugger based on a PIC24FJ256GB106 Microcontroller. The PKOB3 debugger enables the user to program/debug through micro-AB USB connector (J302) on PIC32WFI32E Curiosity Board.

By default, the on-board debugger is connected to the programming pins (PGEC2 and PGED2) of the carrier board.
To use external debugger, remove shunts on J301 to disconnect the on-board debugger from driving the programming pins. The following table provides the details of J301 jumper position for debugger selection.

Table 2-2. J301 Jumper Positions for Debugger Selection

On-board Debugger	External Debugger
Pins 1-2 Shorted	Pins 1-2 Open
Pins 3-4 Shorted	Pins 3-4 Open

2.1.3 ICSP Header

A MPLAB programmer or debugger can be connected to PIC32WFI32E Curiosity Board using the standard ICSP header. The ICSP header (J206) provided in this curiosity board is a standard 6-pin ICSP connector (AC164110). This allows for in-circuit emulation and debugging using Microchip's in-circuit emulator tools, and direct programming of the WFI32E01PC module. ICSP header supports external debuggers, such as MPLAB REAL ICE and MPLAB ICD 3/4 or MPLAB PICKIT 3/4. The following table provides the pin details and descriptions of ICSP header.

Table 2-3. ICSP Header Description

Pin Number	Pin on ICSP Header	Pin Description of ICSP Header	Pin on WFI32E01PC Module(1)
1	$\overline{\text{MCLR}}$	Reset pin	$\overline{\text{MCLR}}$
2	3V3	3.3V power supply	+3V3
3	GND	Ground	GND
4	PGD	ICSP™ programming data	PGD2/AN5/CVD5/CVDR5/CVDT2/RTCC/RPB5
5	PGC	ICSP™ programming clock	PGC2/AN4/CVD4/CVDR4/CVDT3/RPB4/RB4
6	NC	Not connected	NC

Note:
1. For more details on the WFI32E01PC pins, refer to *PIC32MZ1025W104 MCU and WFI32E01 Module with Wi-Fi® and Hardware-based Security Accelerator Data Sheet* (DS70005425A).

2.2 Harmony Setup

The recommended configuration bits and clock configuration are automatically set to compile/build the project.

Note: Go to https://github.com/Microchip-MPLAB-Harmony/mhc/wiki for information on how to install the MPLAB Harmony 3 Configurator and how to use it to configure a project.

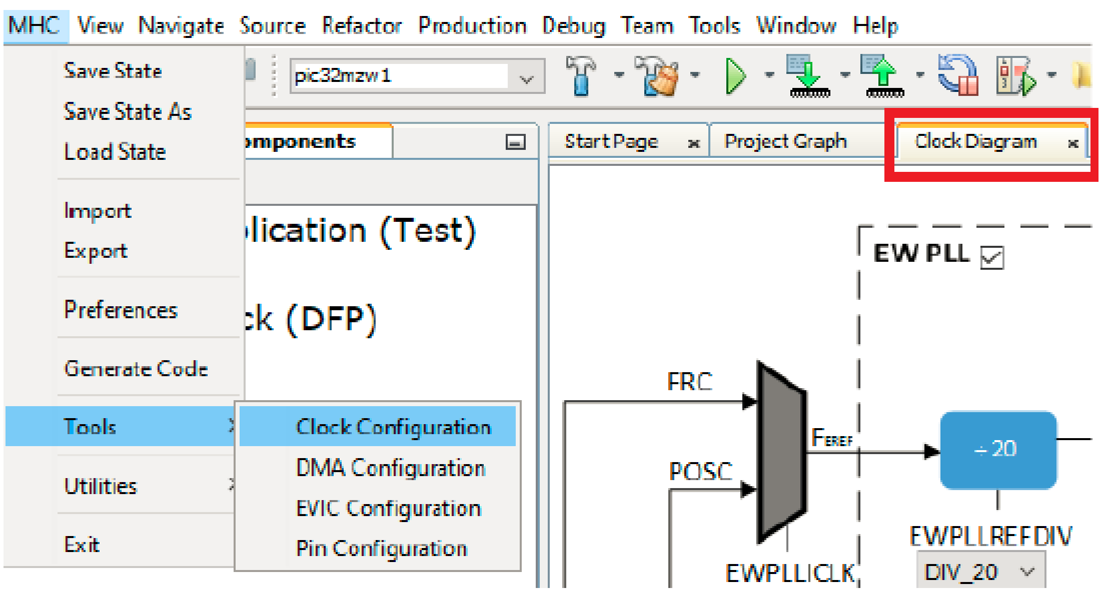
2.2.1 Configuration Bits

It is recommended to use the default configuration bits, the corresponding values are placed in the `initialization.c` file.

Note: For all the device configurations listed as part of `initialization.c`, see 6. Appendix A: Configuration Bits.

2.2.2 Clock Configuration

MHC provides a Clock Diagram tab (see the following image) to showcase all the clocks in the PIC32MZ1025W104 SoC and suggests configuration options. The permitted range of inputs is set to generate clock configurations for a pre-determined output range (via drop-down menus). The clock diagram can be used to configure all clock PLLs in the PIC32MZ W1 clock sub-systems and to setup peripheral clock frequencies which are typically derived from the main clock source.



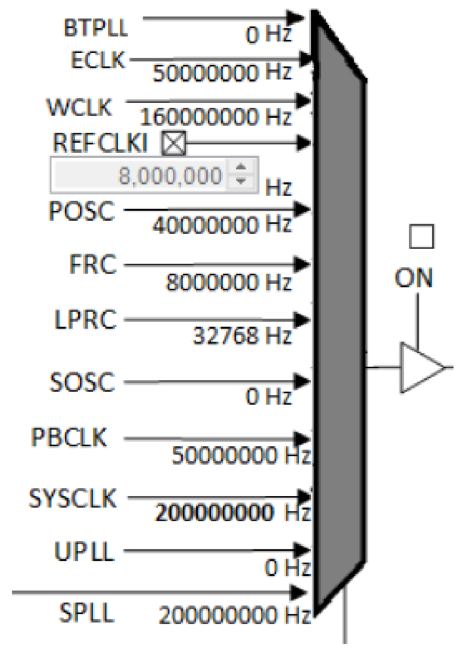
The following table shows the clock sources for all the peripherals. Once these changes are done, initiate a build through the menu or build button in the MPLAB X IDE. When compiled, it is important to verify that the Build Successful message appears in output window of MPLAB X IDE.

Table 2-4. Clock Sources for PIC32MZ W1 Peripherals

Main PLLs	Clock	PIC32MZ W1 Peripherals
SPLL	PBCLK3	USART1, USART2, SPI1, SPI2, I2C2
	PBCLK2	I2C1, Ports A, B, C, K
	PBCLK1	Timer1-7, WDT, USART3, PPS, CACHE, NVM
EWPLL	ECLK/WCLK	ETH/Wi-Fi®
UPLL	USBCLK	USB
BTPLL	BTCLK	BT

Each PLL is routed to inputs on the clock MUX and can be configured as a system clock source as shown in the following figure. The recommended practice is to use SPLL to generate the system clock (through ROSEL1). The values of the clocks shown in the following image are the actual values used in this release.

Figure 2-4. PLL Clock Source Selection



By default, the system is setup to operate at a maximum frequency of 200 MHz. All of the software demos are tested at this frequency. ETHCLK for the Ethernet module is 50 MHz while the WLAN module EWPLL clock is set to 160 MHz (internally divided later). The UPLL is set to 96 MHz. The FRC (8 MHz) is a low frequency clock available at boot up as the system clock and is mostly used to evaluate a new part during the initial development stage or until it switches to clock sources.

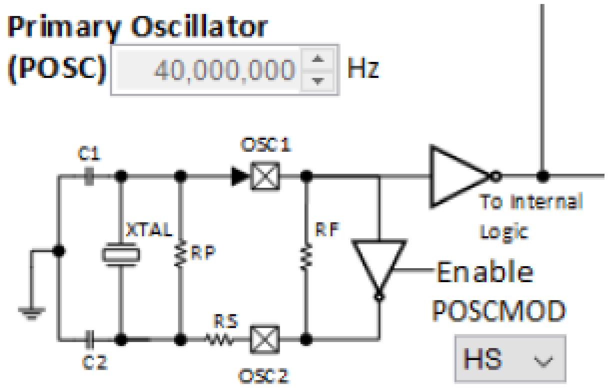
2.2.2.1 Clock Configuration Procedure

Perform the following steps to configure PLLs and peripheral clocks for recommended values:

1. Launch the MHC configuration menu for the required main project.
2. Open Clock Diagram from the menu (MHC > Tools > Clock Configuration) as shown in 2.2.2 Clock Configuration.

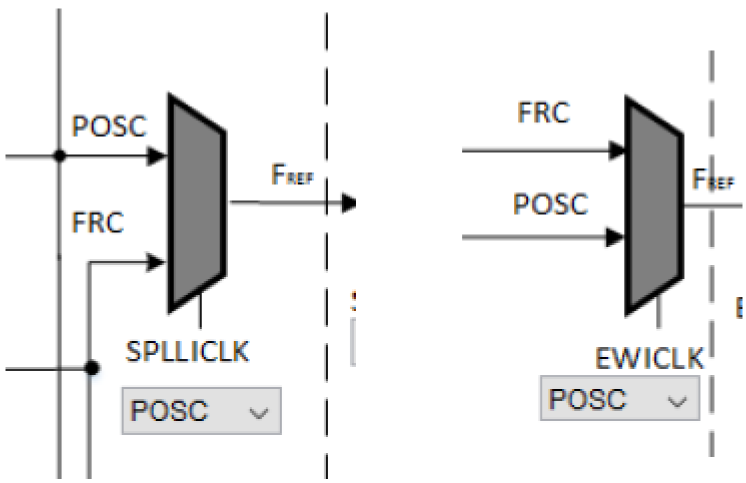
Enable POSCMOD to HS (if not set).

Figure 2-5. Enable POSCMOD



Select the input clock as POSC from the respective clock mux for SPLLICLK and EWICLK as shown in the following image.

Figure 2-6. Set Input Voltage

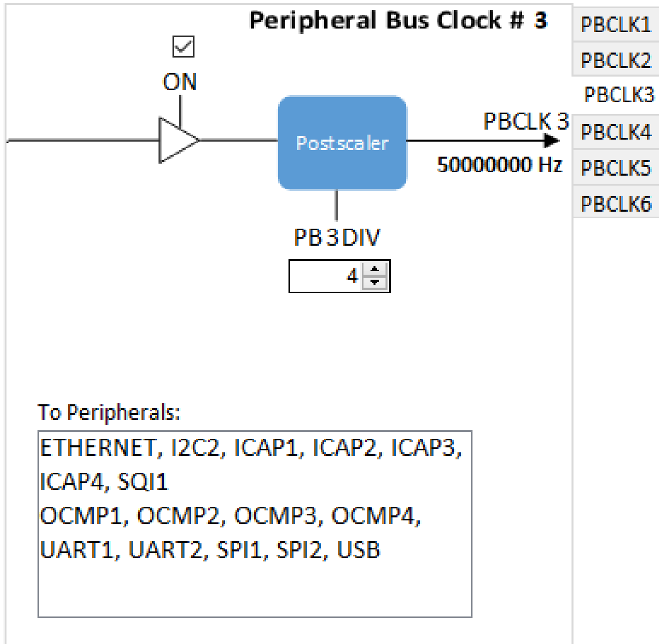


Next click on **Auto-Calculate** button for each PLL block to set the proper divisor values
to achieve the required output frequency. In all the example codes, the system clock (SYSCLK) is set to 200 MHz.

Generate peripheral clocks PKCLK1, according to the requirements

1. Ensure from the Clock Diagram that the required frequencies are derived for peripheral clocks
2. through the clock settings. Usually all peripheral clocks are SYSCLCK/2.
3. Some applications may need to use 4 as the PBCLK3 divisor value (PB3DIV) to provide a 50Mhz clock for the associated/required peripherals.

Figure 2.7. PBCLK3 Divisor Value



4. Save the setting and click on the  button to generate the code.
   a. Once the code generation is successful, initiate a build through the menu or build button through the MPLAB X IDE. (Compile/Re-compile the project).
   b. When compiled, make sure the 'Build Successful' message is seen in the output window of MPLAB® X IDE.

Note:

1. This procedure applies to all the existing example/demo projects and for the development of new example/application projects.

2.2.3 PIN Manager and Ports/PPS Configuration

The MHC tool in the MPLAB IDE is provided to enable users to configure the pins of Microchip PIC32 devices in an easy and time-efficient manner. The tool consists of a graphical representation of the state of the component and a table provides the means to configure the pins of the device.

Perform the following steps to configure a device:

- Launch the tool (if not already running)
- Add modules by enabling desired functionality in the configuration tree (for example, USART or SPI)
- Use the pin table to Lock cells representing function and pin pairings
- Use the pin flag management dialog to change pin register values
- Generate code via the **Generate** button. Once generation is complete, the resultant code for configuring the device pins will be automatically added to the user's project.

Figure 2-8. Launching Pin Configuration

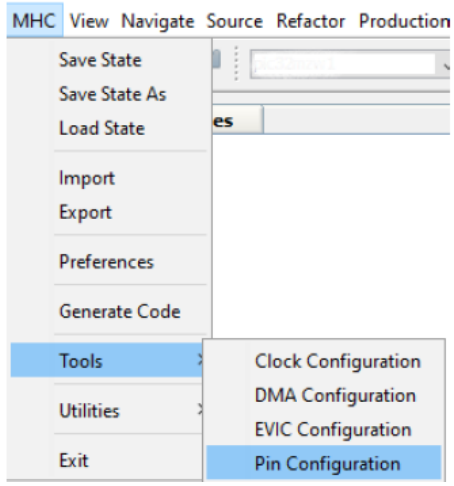
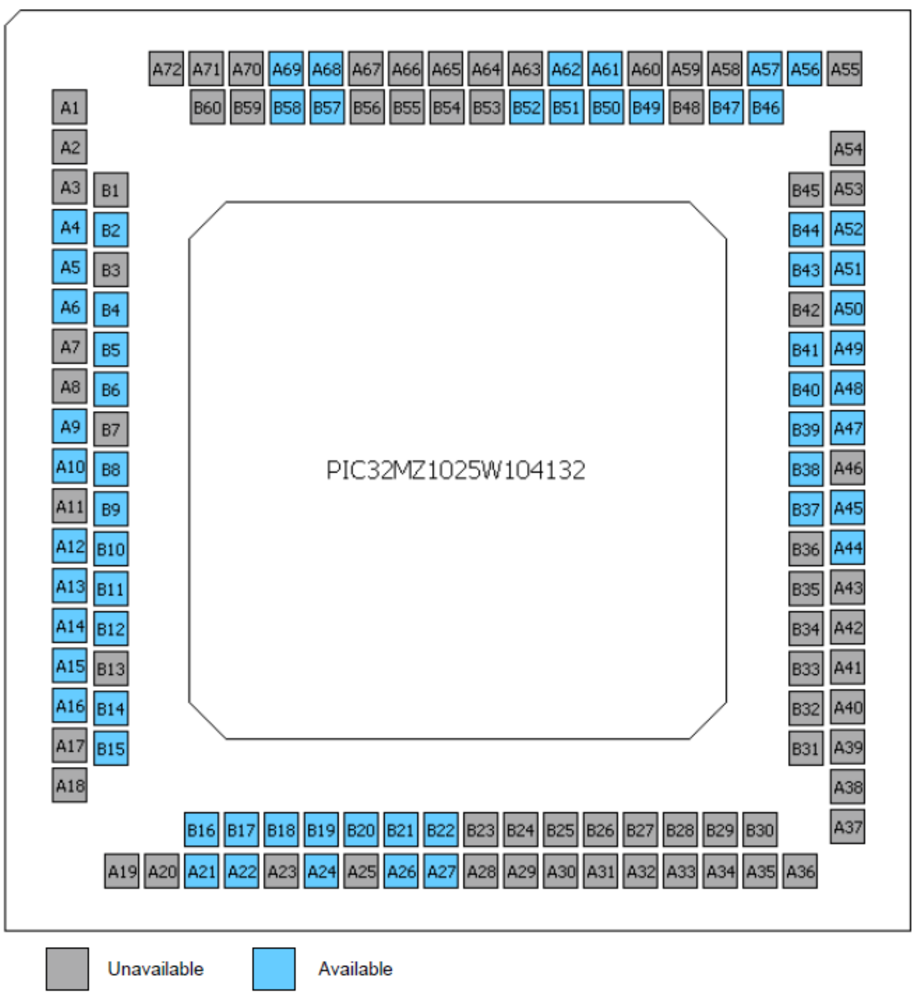


Figure 2-9. DQFN Option in PIC32MZ1025W104 SoC



Note: For more details, refer the PIN Manager section from *MPLAB Harmony Configurator User's Guide* available at https://github.com/Microchip-MPLAB-Harmony/mhc/wiki.

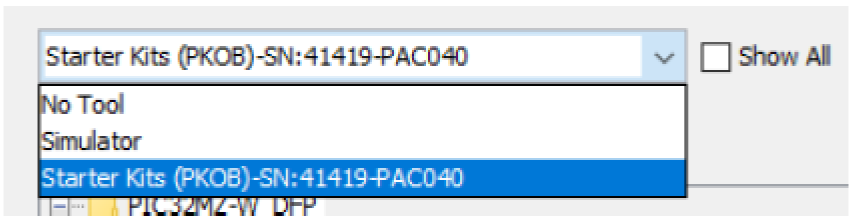
2.3 Programming MPLAB Projects

Note: To create a new MPLAB® X IDE project, refer https://github.com/Microchip-MPLAB-Harmony/mhc/wiki.

Perform the following steps for programming the project:

- Open the MHC plugin tool. If the project has already been created, launch the MPLAB® Harmony 3 Configurator by selecting Tools > Embedded > MPLAB® Harmony 3 Configurator from the MPLAB® X IDE's menu bar.
- Open the MPLAB project and select the MCU device as PIC32MZ1025W104132 from Project Properties.
- Select the MPLAB XC32 v2.40 as the complier for compiling the application or example projects.
- Select the Debugger/Programmer from Projects Properties window as shown below. The following image shows the selected Starter Kit.

Figure 2-10. Select On-board PKOB



Use the following steps (i and ii) or (iii) to build the project:

- Production > Build Main Project or press F11 to build a project
- Production > Set Main Project > Choose the project to program
- Go to Debug > Debug Main Project

Note: For more details on programming and debugging of the curiosity board, refer to *PIC32WFI32E Curiosity Board User's Guide* (DSxxxxxxxx).

3 Peripheral Libraries

This section describes the peripheral libraries supported by PIC32MZ W1 Family.

ADCHS

adchs_polled - In this application, an analog input is converted by a hardware trigger generated by the TMR peripheral. Converted digital value is displayed on the serial terminal.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

CAN

can_normal_operation_blocking - This application transmits and receives messages to/from the CAN Bus.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

CORETIMER

coretimer_periodic_timeout - This example application configures the CoreTimer Peripheral Library to generate periodic interrupts. The application registers a periodic timeout callback. It toggles an LED every time the callback is triggered.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

CLOCK

clock_config - The Clock system generates and distributes the clock for the processor and peripherals. This example application shows how to use the clock manager to configure the device to run at the maximum possible speed. The pre-scaled clock signal is routed to GPIO pin enabling the developer to measure the frequency and accuracy of the internal device clock.

Topics

Name	Description
Building The Application	Information on how to build application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

DMAC

dmac_memory_transfer - This application uses a software trigger to initiate a memory-memory transfer from a source buffer to a destination buffer with 16-bit size and 32-bit size.

PIC32MZ: Please note only the transfer status is reported on console, not the data that is transferred.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

DMT

dmt_timeout - This example application shows how the DMT PLIB resets the device if the application software fails to clear the Deadman timer counter frequently enough. This application sets up the DMT to reset the device after the DMT counter overflows. This application also sets up the CoreTimer to blink an LED and to clear the DMT counter to avoid the DMT reset. Later, when the user presses the switch, it forces the device to wait in an infinite loop thereby emulating a deadlock condition. As a result, the Deadman timer overflows and triggers a device.

Topics

Name	Description
Building The Application	Information on how to build the application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using the MPLAB X IDE.

GPIO

gpio_led_on_off_interrupt - This application uses the GPIO Peripheral library to generate callbacks when the user presses or releases a switch. It drives the GPIO pin connected to the LED to indicate the switch status: ON when the switch is pressed, and OFF when the switch is released.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

I2C

i2c_eeprom - This example uses the I2C peripheral library to write an array of values to an I2C Serial EEPROM device (not supplied). It verifies the values written by reading the values back and comparing to the values written.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

ICAP

icap_capture_mode - In this application, a pulse signal is generated using the OCMP peripheral and is fed to the ICAP input. The ICAP peripheral is used to measure the times each pulse edge occurs, and the application calculates and displays the pulse width on the serial terminal.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

NVM

flash_read_write - This example uses the NVM peripheral library to erase a page and write an array of values to the internal Flash memory. It verifies the data written by reading the values back and comparing to those written.

Topics

Name	Description
Building the Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configuration	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

OCMP

ocmp_compare_mode - In this application, three OCMP modules are used to generate an active low, an active high and a signal which toggles when the programmed timer counter matches the compare value. This match results in an output pin level change as per the configured output compare mode.

Active Low Output: By default, output is set as high and it is set as low on the compare match

Active High Output: By default, output is set as low and it is set as high on the compare match

Toggled Output: Compare match toggles the output

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

RNG

rng_random_number - The RNG peripheral has two blocks: a True Random Number Generator (TRNG) and a Pseudo-Random Number Generator (PRNG). The TRNG generates true random numbers that can be used as seeds for the PRNG. The PRNG can generate random numbers of up to 64-bits in length. This application shows how to generate a 64-bit pseudo-random number with the help of TRNG and PRNG.

Topics

Name	Description
Building the Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running the Application	Information on running the application using MPLAB X IDE.

RTCC

rtcc_alarm - This example application shows how to setup the RTCC time and configure an alarm using the RTCC Peripheral Library. The application sets up an alarm to be generated every day at a specified time. A message is sent via the Virtual COM port when the alarm triggers.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

SPI

spi_sst26_write_read - This example uses the SPI peripheral library to write an array of values to the SST26 Flash and verify the value written by reading the values back and comparing it to the those written. The example application performs the SST26 write only once after a power on reset.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configuration	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

TMR

tmr_timer_mode - The TMR module generates interrupts at 1 second intervals . The application registers an ISR with the framework which toggles the state of the onboard RED LED enabling the developer to verify the operation of the timer.

Two TMR modules are used to form a 32-bit timer.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

TMR1

tmr1_timer_mode - The TMR1 module generates a 100 ms periodic interrupt. The RED LED onboard is toggled in the interrupt handler to indicate periodic callback.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

UART

uart_echo_interrupt - This example shows the read and write operation over a UART in a non-blocking manner. The peripheral interrupt is used to manage the transfer. The application receives 10 characters from the terminal window and echoes them back.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

WDT

wdt_timeout - This example application shows how the WatchDog PLIB resets the device if the application software fails to clear the WDT timer counter frequently enough. This application sets up the WDT to reset the device if the WDT counter overflows/reaches zero (whatever the correct definition is). This application also sets up the CoreTimer to blink an LED that clears the watchdog counter and avoid the reset. Later, when the user presses the switch, it forces the device to wait in an infinite loop thereby emulating a deadlock condition. As a result, the Watchdog timer overflows/reaches zero (whatever) and triggers a device reset.

Topics

Name	Description
Building The Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running The Application	Information on running the application using MPLAB X IDE.

4. Core Examples

4.1 FILESYSTEM

sdspi_fat - File System Operations on the SD Card:

- This application uses the SDSPI driver to communicate with an SD card over the SPI interface.
- It opens a file named FILE_TOO_LONG_NAME_EXAMPLE_123.JPG on the root of the SD card, and creates a copy of it in a directory named Dir1 (see note below).
- The source image file can be any file of the users choice but we recommend a JPEG (image) file to allow easy (i.e. visual) verification of a large amount of data.

Note:

The application creates the directory named Dir1; it is important that this directory **does not exist on the SD card**. If the directory is already present on the SD card, the application will exit with an error.

Topics

Name	Description
Building the Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configuration	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running the Application	Information on running the application using MPLAB X IDE.

nvmi_fat

File System Operations on NVM:

- The application contains a FAT disk image consisting of a Master Boot Record (MBR) sector, Logical Boot Sector, File Allocation Table, and Root Directory Area, placed in the internal Flash memory (NVM)
- The application opens an existing file named **FILE.TXT** and performs following file system related operations:
  - SYS_FS_FileStat
  - SYS_FS_FileSize
  - SYS_FS_FileSeek
  - SYS_FS_FileEOF
- Performs read on the file and checks if string **"Data"** is present. If present, it continues to next step or it fails the application.
- Finally, the string **"Hello World"** is written to this file. The string is then read and compared with the string that was written to the file. If the string compare is successful, an LED indication is provided.

File system layer uses:

- Memory driver to communicate with underlying NVM media.

Topics

Name	Description
Building the Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configuration	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running the Application	Information on running the application using MPLAB X IDE.

4.2 I2C Driver

i2c_eeprom (sync & async)

This example uses the I^{2}C driver in synchronous mode to communicate with the EEPROM and perform write and read operations in an RTOS environment.

The application communicates with the following EEPROM's based on the project configuration selected.

- External AT24CM02 EEPROM
- On-Board AT24MAC402

Topics

Name	Description
Building the Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configuration	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running the Application	Information on running the application using MPLAB X IDE.

4.3 SPI Driver

spi_self_loopback_multi_client (sync & async)

This example writes and reads back the same data (self-loop back) for two different clients connected over the same SPI bus by using the multi-client feature of a synchronous SPI driver. The example also demonstrates how to setup two different client transfers at two different baud rates.

The example has three RTOS threads for the purpose:

1. **APP_CLIENT1_Tasks**: This thread opens the SPI driver instance and performs a continuous loop-back transfer. If the loop back is successful, the loop back is repeated every 100 ms. In case of an error, the thread closes the driver and suspends itself.
2. **APP_CLIENT2_Tasks**: This thread opens the SPI driver instance and performs a continuous loop back transfer. If the loop back is successful, the loop back is repeated every 100 ms. In case of an error, the thread closes the driver and suspends itself.
3. **APP_MONITOR_Tasks:** This thread checks the status of loop back done by the two client tasks and turns on the LED if the loop back transfer status reported by both the clients is successful.

Topics

Name	Description
Building the Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configuration	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running the Application	Information on running the application using MPLAB X IDE.

4.4 USART Driver

usart_echo (sync & async)

This example uses the USART driver in synchronous mode in an RTOS environment to communicate over the console. It receives and echoes back the characters entered by the user.

Topics

Name	Description
Building the Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running the Application	Information on running the application using MPLAB X IDE.

4.5 System Services

sys_time_multiclient

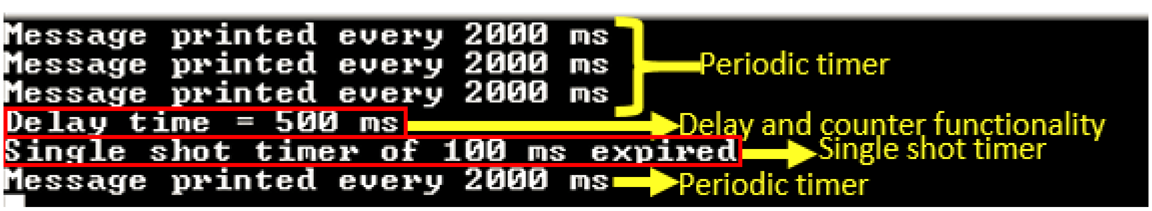
This application demonstrates the use of two free-running periodic timers with a single shot timer.

At startup, the application creates two persistent periodic timers as follows:

- the first causes a text message (as illustrated below) to be displayed on the console every two seconds,
- the second causes an LED to toggle ON/OFF every second.

The application also monitors SW1 button on the Curiosity board and installs an additional single-shot timer whenever the button is pressed. This is a single-shot timer configured to expire 500ms into the future. On expiry the application calculates the time elapsed between installing the timer and the current time and displays a message on the console,."*delay time = x ms*"

Finally, in the callback for the above-described single-shot timer, another single shot timer is configured to expire 100ms into the future. On expiry, the application displays the message "Single shot timer of 100ms expired" on the console.



Topics

Name	Description
Building the Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configuration	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running the Application	Information on running the application using MPLAB X IDE.

5. THIRD PARTY LIBRARIES

RTOS

basic_freertos - This application demonstrates inter-process communication between FreeRTOS tasks using a message queue.

There are three tasks in execution:

- Task 1: This is the master task that controls the behavior of Tasks 2 and 3. It sends messages to these tasks with information (delay duration) via a queue causing them to perform a dedicated (task specific) operation. It then yields to allow these other tasks to execute. The master task will resend these messages every 200ms.
- Task 2: This task sits idle until receiving a message from the master task. Upon receiving a message, it checks to see if the delay in the message matches the value it expects. If it matches, the task then toggles the state of the Red LED and then blocks for the duration indicated in the message from the master task (yielding the CPU).
- Task 3: This task behaves identically to Task 2.

.

Topics

Name	Description
Building the Application	Information on how to build an application using MPLAB X IDE.
MPLAB Harmony Configurations	Information on the MHC configurations.
Hardware Setup	Information on configuring the supported hardware.
Running the Application	Information on running the application using MPLAB X IDE.

Note: For information on other third-party libraries supported by the PIC32MZ W1 Family, refer:
• https://github.com/Microchip-MPLAB-Harmony/wolfMQTT for wolfMQTT library
• https://github.com/Microchip-MPLAB-Harmony/wolfssl for wolfSSL library

6 WLAN API Guide

System Interface

The PIC32MZW1 Wi-Fi module consists of two instances, a control instance and a data instance. The control instance is driven as a conventional Harmony system interface by the system kernel or scheduler to initialize and run the module, the data instance implements a Harmony TCP/IP stack MAC Driver module instance driven by the Harmony TCP/IP stack.

```
SYS_MODULE_OBJ WDRV_PIC32MZW_Initialize
(
        const SYS_MODULE_INDEX index,
        const SYS_MODULE_INIT *const init
)
```

Description:

This function initializes the PIC32MZW1 Wi-Fi module instance.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.

Parameters:

<table>
<tr><td><code>index</code></td><td>Zero-based index of the module instance to be initialized. This value is either:<table><tr><td>Zero</td><td>Module Control Instance</td></tr><tr><td>Non-Zero</td><td>Module Data Instance</td></tr></table></td></tr>
<tr><td><code>Init</code></td><td>Pointer to the data structure containing any data necessary to initialize the module. This pointer may be null if no data is required.</td></tr>
</table>

Returns:

A handle to the instance of the module that was initialized.
NOTE: This handle is a necessary parameter to all the other system level routines for that module.

Remarks:

For the control instance this function will normally only be called once during system initialization.

```
void WDRV_PIC32MZW_Deinitialize(SYS_MODULE_OBJ object)
```

Description:

This function deinitializes a PIC32MZW1 Wi-Fi module instance.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.

Parameters:

`object`	Handle to the module instance.

Remarks:

If the module instance must be used again, the module's "initialize" function must first be called.

```
SYS_STATUS WDRV_PIC32MZW_Status(SYS_MODULE_OBJ object)
```

Description:

This function gets the current status of the PIC32MZW1 Wi-Fi module instance.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.

Parameters:

`Object`	Handle to the module instance.

Returns:

One of the possible status codes from `SYS_STATUS`

Remarks:

A module's status function can be used to determine when any of the other system level operations has completed as well as to obtain general status of the module.

If the status function returns `SYS_STATUS_BUSY`, a previous operation has not yet completed.

Once the status function returns `SYS_STATUS_READY`, any previous operations have completed.

The value of `SYS_STATUS_ERROR` is negative (-1).

```
void WDRV_PIC32MZW_Tasks(SYS_MODULE_OBJ object)
```

Description:

This function performs the tasks necessary to maintain the state machine in the PIC32MZW1 Wi-Fi module instance.

Preconditions:

1. The low-level board initialization must have been completed and the module's initialization function must have been called before the system can call the tasks routine for any module.

Parameters:

`object`	Handle to the module instance.

Wi-Fi Driver Data Types

`WDRV_PIC32MZW_ASSOC_HANDLE`

Description:

An ‘opaque’ type (implemented as a pointer). Application code should never set this directly and should instead only store values returned from the Wi-Fi driver for later use in subsequent calls (to the Wi-Fi driver).

`WDRV_PIC32MZW_AUTH_TYPE`

Description:

Defines various authentication types for use in connecting-to or hosting Wi-Fi networks.

Definitions:

Please note: the following definitions are all prefixed with ‘`WDRV_PIC32MZW_AUTH_TYPE_`’:

`DEFAULT`	Recommended for general use when connecting to a network – will automatically use the most secure type available
`OPEN`	Open system (no security)
`WPAWPA2_PERSONAL`	WPA2 mixed mode / compatibility mode with pre-shared key
`WPA2_PERSONAL`	WPA2-only
`WPA2WPA3_PERSONAL`	WPA3 (Simultaneous Authentication of Equals) transition mode – accepts either WPA2 and WPA3 security
`WPA3_PERSONAL`	WPA3-only

`WDRV_PIC32MZW_AUTH_MOD_MASK`

Description:

Defines modifiers (optional settings) for a configured `WDRV_PIC32MZW_AUTH_TYPE`. Not all modifiers are relevant to all authentication types.

Definitions:

Please note: the following definitions are all prefixed with ‘`WDRV_PIC32MZW_AUTH_MOD_`’:

`NONE`	Recommended – no modifiers, default behavior for the configured `AUTH_TYPE`
`MFP_REQ`	Management Frame Protection (802.11w) is required. Relevant only if the authentication type is …`WPA2_PERSONAL` or …`WPA2WPA3_PERSONAL`.
`MFP_OFF`	Management Frame Protection (802.11w) is disabled. Relevant only if the authentication type is …`WPAWPA2_PERSONAL` or …`WPA2_PERSONAL`.
`SHARED_KEY`	Enables shared key authentication, this is generally not recommended and should only be used if the network refuses to accept the default open system authentication. Relevant only if the authentication type is `WEP`.


```
WDRV_PIC32MZW_CHANNEL_ID
```

Description:

Defines the supported Wi-Fi channels on the PIC322MZW1.

Definitions:

Please note: the following definitions are all prefixed with ‘`WDRV_PIC32MZW_CID_`’:

`ANY`	Use any valid channel
`2_4G_CH1`	Use 2.4GHz band, channel 1 (2412 MHz)
`2_4G_CH2`	Use 2.4GHz band, channel 2 (2417 MHz)
`2_4G_CH3`	Use 2.4GHz band, channel 3 (2422 MHz)
`2_4G_CH4`	Use 2.4GHz band, channel 4 (2427 MHz)
`2_4G_CH5`	Use 2.4GHz band, channel 5 (2432 MHz)
`2_4G_CH6`	Use 2.4GHz band, channel 6 (2437 MHz)
`2_4G_CH7`	Use 2.4GHz band, channel 7 (2442 MHz)
`2_4G_CH8`	Use 2.4GHz band, channel 8 (2447 MHz)
`2_4G_CH9`	Use 2.4GHz band, channel 9 (2452 MHz)
`2_4G_CH10`	Use 2.4GHz band, channel 10 (2457 MHz)
`2_4G_CH11`	Use 2.4GHz band, channel 11 (2462 MHz)
`2_4G_CH12`	Use 2.4GHz band, channel 12 (2467 MHz)
`2_4G_CH13`	Use 2.4GHz band, channel 13 (2472 MHz)

```
WDRV_PIC32MZW_CHANNEL24_MASK
```

Description:

A bitmap identifying the legal channels for the current regulatory region. This type also contains definitions for specific regulatory regions such as North America, Europe, Asia for convenience when using the regulatory APIs.

Definitions:

Please note: the following definitions are all prefixed with 'WDRV_PIC32MZW_CM_2_4G_':

CH1	2.4GHz band, channel 1 (2412 MHz) is valid/enabled
:	:
CH13	2.4GHz band, channel 13 (2472 MHz) is valid/enabled
DEFAULT	Channels 1 through 11 are valid/enabled – typically used with region 'GEN'
NORTH_AMERICA	Channels 1 through 11 are valid/enabled – typically used with region 'USA'
EUROPE	Channels 1 through 13 are valid/enabled – typically used with region 'EMEA'
ASIA	Channels 1 through 14 are valid/enabled – typically used with region 'JPN'

```
WDRV_PIC32MZW_CONN_STATE
```

Description:

Defines the possible Wi-Fi connection states of the driver.

Definitions:

Please note: the following definitions are all prefixed with 'WDRV_PIC32MZW_CONN_STATE_':

DISCONNECTED	Not connected / idle
CONNECTING	In the process of connecting to a network
CONNECTED	Associated (connected)
FAILED	Association attempt failed

```
WDRV_PIC32MZW_MAC_ADDR
{
      uint8_t addr[WDRV_PIC32MZW_MAC_ADDR_LEN];
      bool valid;
}
```

Description:

Defines a Wi-Fi network name (Service Set Identifier) – this is used to identify a specific Wi-Fi network to connect to, or to specify the name of a the network being hosted when the device is in SoftAP mode.

Fields:

`addr`	A 6 byte array containing the BSSID / MAC-address of the network
`valid`	Identifies if the address is valid (`True`) or invalid (`False`)

```
WDRV_PIC32MZW_SSID
{
      uint8_t name[WDRV_PIC32MZW_MAX_SSID_LEN];
      uint8_t length;
}
```

Description:

Defines a Wi-Fi network name (Service Set Identifier) – this is used to identify a specific Wi-Fi network to connect to, or to specify the name of a the network being hosted when the device is in SoftAP mode.

Fields:

`name`	A 32-character long string – this is the maximum allowed length of an SSID
`length`	The length of the SSID (number of bytes populated in the `name` field)

```
WDRV_PIC32MZW_BSS_CONTEXT
{
      WDRV_PIC32MZW_SSID ssid;
      WDRV_PIC32MZW_MAC_ADDR bssid;
      WDRV_PIC32MZW_CHANNEL_ID channel;
      Bool cloaked;
}
```

Description:

Context structure containing information related to Basic Service Sets (BSS). The BSS context is used to identify a network to connect to, or to define the parameters for the device if in softAP mode.

Fields:

`ssid`	The network name
`bssid`	The MAC address of the access point
`channel`	The channel (1-13) that the network resides on
`cloaked`	Identifies whether the network is visible (`False`) or hidden (`True`)

```
WDRV_PIC32MZW_BSS_INFO
{
      WDRV_PIC32MZW_BSS_CONTEXT ctx;
      int8_t rssi;
      WDRV_PIC32MZW_SEC_MASK secCapabilities;
      WDRV_PIC32MZW_AUTH_TYPE authTypeRecommended;
}
```

Description:

A structure containing information related to a discovered network (Basic Service Set). This is typically used to inspect a network's capabilities prior to association/connection.

Fields:

`ctx`	The BSS (Wi-Fi network) context, refer to `WDRV_PIC32MZW_BSS_CONTEXT`
`rssi`	The received signal strength of the network
`secCapabilities`	The network security capabilities, refer to `WDRV_PIC32MZW_SEC_MASK`
`authTypeRecommended`	Recommended authentication type for this network

```
WDRV_PIC32MZW_AUTH_CONTEXT
{
      WDRV_PIC32MZW_AUTH_TYPE authType;
      WDRV_PIC32MZW_AUTH_MOD_MASK authMod;
      authInfo;
}
```

Description:

Context structure containing information related to the authentication requirements of the Wi-Fi network (typically used in conjunction with the `WDRV_PIC32MZW_BSS_CONTEXT`).

Fields:

<table>
<tr><td><code>authType</code></td><td>The security suite to use – recommend to use <code>PIC32MZW1_AUTH_TYPE_DEFAULT</code></td></tr>
<tr><td><code>authMod</code></td><td>Security type modifiers/options – recommend to use <code>PIC32MZW1_AUTH_MOD_NONE</code></td></tr>
<tr><td><code>authInfo</code></td><td>A union of two structures defining credentials used for either WEP or WPA/WPA2/WPA3 (Personal) security. The structures within this union have the following fields:

WEP
<table>
<tr><td><code>uint8_t idx</code></td><td>WEP key index (range 1 – 4)</td></tr>
<tr><td><code>uint8_t size</code></td><td>WEP Key size (10 for WEP_40, or 26 for WEP_104)</td></tr>
<tr><td><code>uint8_t key[]</code></td><td>A 27 byte array for storing the WEP key</td></tr>
</table>
WPA/WPA2/WPA3 Personal
<table>
<tr><td><code>uint8_t size</code></td><td>Password length in bytes (maximum of 64)</td></tr>
<tr><td><code>uint8_t password[]</code></td><td>A 64 byte array for storing the password</td></tr>
</table>
</td></tr>
</table>

`WDRV_PIC32MZW_REGDOMAIN_SELECT`

Description:

Defines the set of regulatory configurations to return to the application when querying via the `WDRV_PIC32MZW_RegDomainGet` API.

Definitions:

Please note: the following definitions are all prefixed with '`WDRV_PIC32MZW_REGDOMAIN_SELECT _`':

`CURRENT`	Request only the information for the currently active region
`ALL`	Request information on all the regulatory configurations stored on the module

`WDRV_PIC32MZW_SEC_MASK`

Description:

Defines the various security suites supported by a network.

Definitions:

Please note: the following definitions are all prefixed with '`WDRV_PIC32MZW_SEC_BIT_`':

`WEP`	Network supports WEP
`WPA`	Network supports WPA (uses TKIP encryption for all traffic)
`WPA2OR3`	Network supports CCMP for pairwise traffic, support CCMP for group traffic only if the `WPA` bit (above) is not set
`MFP_CAPABLE`	Network supports (does not mandate) management frame protection (802.11w)
`MFP_REQUIRED`	Network mandates use of management frame protection (802.11w)
`ENTERPRISE`	Network supports WPA/WPA2/WPA3 Enterprise
`PSK`	Network supports passphrase (WPA/WPA2 Personal) authentication
`SAE`	Network supports WPA3 (Simultaneous Authentication of Equals) authentication

Client Interface - Open/Close

```
DRV_HANDLE WDRV_PIC32MZW_Open
(
        const SYS_MODULE_INDEX index,
        const DRV_IO_INTENT intent
)
```

Description:

This function opens a driver for use by a client module and provides an open-instance handle that must be provided when calling all client-interface functions to identify the caller and the instance of the driver module.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.

Parameters:

index	A zero-based index, identifying the instance of the driver to be opened. This value matches the index passed to the driver's initialize function.
intent	This parameter is not currently used.

Returns:

If successful, the function returns a valid open-instance handle (an opaque value identifying both the caller and the driver instance).

If an error occurs, the value returned is `DRV_HANDLE_INVALID` (-1).

```
void WDRV_PIC32MZW_Close(DRV_HANDLE handle)
```

Descriptions:

This function closes an opened instance of a driver, invalidating the handle provided.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.

Parameters:

`handle`	A valid open-instance handle, returned from the driver's open function.

Returns:

None.

Client Interface – STA

The PIC32MZW1 device can connect to a BSS as a Wi-Fi station device. To connect to a BSS the function WDRV_PIC32MZW_BSSConnect can be called, this takes two contexts which provide information about the BSS to be connected to and the authentication method to be used. A callback is provided to communicate events back to the application regarding connection status.

To disconnect from a BSS the function WDRV_PIC32MZW_BSSDisconnect can be called.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSConnect
(
    DRV_HANDLE handle,
    const WDRV_PIC32MZW_BSS_CONTEXT *const pBSSCtx,
    const WDRV_PIC32MZW_AUTH_CONTEXT *const pAuthCtx,
    const WDRV_PIC32MZW_BSSCON_NOTIFY_CALLBACK pfNotifyCallback
)
```

Description:

Using the defined BSS and authentication contexts, this function requests the PIC32MZW1 connect to the BSS as an infrastructure station.

Preconditions:

`WDRV_PIC32MZW_Initialize` should have been called.
`WDRV_PIC32MZW_Open` should have been called to obtain a valid handle.
A BSS context must have been created and initialized.
An authentication context must have been created and initialized.

Parameters:

`handle`	Client handle obtained by a call to `WDRV_PIC32MZW_Open`.
`pBSSCtx`	Pointer to BSS context.
`pAuthCtx`	Pointer to authentication context.

`pfNotifyCallback`	Pointer to function of the following prototype:
	`void func`
	`(`
	`DRV_HANDLE handle,`
	`WDRV_PIC32MZW_ASSOC_HANDLE assocH,`
	`WDRV_PIC32MZW_CONN_STATE state`
	`)`

Returns:

`WDRV_PIC32MZW_STATUS_OK`	- The request has been accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN`	- The driver instance is not open.
`WDRV_PIC32MZW_STATUS_INVALID_ARG`	- The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_REQUEST_ERROR`	- The request to the PIC32MZW1 was rejected.
`WDRV_PIC32MZW_STATUS_INVALID_CONTEXT`	- The BSS context is not valid.
`WDRV_PIC32MZW_STATUS_CONNECT_FAIL`	- The connection has failed.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSDisconnect(DRV_HANDLE handle)
```

Description:

Disconnects from an existing BSS.

Preconditions:

`WDRV_PIC32MZW_Initialize` should have been called.
`WDRV_PIC32MZW_Open` should have been called to obtain a valid handle.

Parameters:

`handle`	Client handle obtained by a call to WDRV_PIC32MZW_Open.

Returns:

`WDRV_PIC32MZW_STATUS_OK`	- The request has been accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN`	- The driver instance is not open.
`WDRV_PIC32MZW_STATUS_INVALID_ARG`	- The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_DISCONNECT_FAIL`	- The disconnection has failed.
`WDRV_PIC32MZW_STATUS_REQUEST_ERROR`	- The request to the PIC32MZW1 was rejected.

Client Interface - Soft-AP

The PIC32MZW1 device can create a Soft-AP allowing a Wi-Fi station device to connect to the PIC32MZW1.

To create a Soft-AP the function WDRV_PIC32MZW_APStart is called. This function takes several context structures to config aspects of the Soft-AP operation, these contexts are:

1. BSS - Describes the BSS which will be created
2. Authentication - Describes the security level to be used

Specification of a BSS context is required. If no authentication context is provided, the Wi-Fi driver will create a Soft-AP with no security.

A callback is also provided to update the application on events.

To stop using the Soft-AP the function WDRV_PIC32MZW_APStop is called.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_APStart
(
    DRV_HANDLE handle,
    const WDRV_PIC32MZW_BSS_CONTEXT *const pBSSCtx,
    const WDRV_PIC32MZW_AUTH_CONTEXT *const pAuthCtx,
    const WDRV_PIC32MZW_BSSCON_NOTIFY_CALLBACK pfNotifyCallback
)
```

Description:

Using the defined BSS and authentication contexts this function creates and starts a Soft-AP instance.

Preconditions:

`WDRV_PIC32MZW_Initialize` should have been called.
`WDRV_PIC32MZW_Open` should have been called to obtain a valid handle.
A BSS context must have been created and initialized.
An authentication context must have been created and initialized.

Parameters:

handle	Client handle obtained by a call to WDRV_PIC32MZW_Open.
pBSSCtx	Pointer to BSS context.
pAuthCtx	Pointer to authentication context.
pfNotifyCallback	Pointer to notification callback function.

Returns:

`WDRV_PIC32MZW_STATUS_OK` - The request has been accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN` - The driver instance is not open.
`WDRV_PIC32MZW_STATUS_INVALID_ARG` - The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_REQUEST_ERROR` - The request to the PIC32MZW1 was rejected.
`WDRV_PIC32MZW_STATUS_INVALID_CONTEXT` - The BSS context is not valid.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_APStop(DRV_HANDLE handle)
```

Description:

Stops an instance of Soft-AP.

Precondition:

`WDRV_PIC32MZW_Initialize` should have been called.
`WDRV_PIC32MZW_Open` should have been called to obtain a valid handle.

Parameters:

`handle`	Client handle obtained by a call to WDRV_PIC32MZW_Open.

Returns:

`WDRV_PIC32MZW_STATUS_OK`	- The request has been accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN`	- The driver instance is not open.
`WDRV_PIC32MZW_STATUS_INVALID_ARG`	- The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_REQUEST_ERROR`	- The request to the PIC32MZW1 was rejected.

Client Interface - Authentication Context

The authentication context contains information relevant to the authentication mechanisms used in Wi-Fi. Currently supported are Open, WEP and WPA-PSK.

The structure WDRV_PIC32MZW_AUTH_CONTEXT is provided to contain the context.

To initialize a context the function WDRV_PIC32MZW_AuthCtxSetDefaults is provided, this ensures the context is in a known state.

The function WDRV_PIC32MZW_AuthCtxIsValid is provided to test if a context is valid.

Each authentication type has at least one function which can be used to configure the type of authentication algorithm used as well as the parameters which are required.

```
bool WDRV_PIC32MZW_AuthCtxIsValid(const WDRV_PIC32MZW_AUTH_CONTEXT *const pAuthCtx)
```

Description:

Tests the elements of the authentication context to judge if their values are legal.

Parameters:

`pAuthCtx`	Pointer to an authentication context.

Returns:

`True` or `False` indicating if context is valid.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_AuthCtxSetDefaults
(
        WDRV_PIC32MZW_AUTH_CONTEXT *const pAuthCtx
)
```

Description:

Ensures that each element of the structure is configured into a default state.

Parameters:

`pAuthCtx`	Pointer to an authentication context.

Returns:

`WDRV_PIC32MZW_STATUS_OK` - The context has been configured.
`WDRV_PIC32MZW_STATUS_INVALID_ARG` - The parameters were incorrect.

Remarks:

A default context is not valid until it is configured.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_AuthCtxSetOpen
(
        WDRV_PIC32MZW_AUTH_CONTEXT *const pAuthCtx
)
```

Description:

The type and state information are configured appropriately for Open authentication.

Parameters:

pAuthCtx	Pointer to an authentication context.

Returns:

WDRV_PIC32MZW_STATUS_OK - The context has been configured.
WDRV_PIC32MZW_STATUS_INVALID_ARG - The parameters were incorrect.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_AuthCtxSetWEP
(
        WDRV_PIC32MZW_AUTH_CONTEXT *const pAuthCtx,
        uint8_t idx,
        uint8_t *pKey,
        uint8_t size
)
```

Description:

The type and state information are configured appropriately for WEP authentication.

Parameters:

pAuthCtx	Pointer to an authentication context.
idx	WEP index.
pKey	Pointer to WEP key.
size	Size of WEP key.

Returns:

WDRV_PIC32MZW_STATUS_OK - The context has been configured.
WDRV_PIC32MZW_STATUS_INVALID_ARG - The parameters were incorrect.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_AuthCtxSetPersonal
(
        WDRV_PIC32MZW_AUTH_CONTEXT *const pAuthCtx,
        uint8_t *pPassword,
        uint8_t size,
        WDRV_PIC32MZW_AUTH_TYPE authType
)
```

Description:

The type and state information are configured appropriately for WPA-PSK authentication.

Parameters:

pAuthCtx	Pointer to an authentication context.
pPassword	Pointer to password (or 64-character PSK).
size	Size of password (or 64 for PSK).
authType	Authentication type (or WDRV_PIC32MZW_AUTH_TYPE_DEFAULT).

Returns:

WDRV_PIC32MZW_STATUS_OK - The context has been configured.
WDRV_PIC32MZW_STATUS_INVALID_ARG - The parameters were incorrect.

Client Interface - BSS Context

The BSS context contains information relevant to a BSS such as SSID and channel.

The structure WDRV_PIC32MZW_BSS_CONTEXT is provided to contain the context.

To initialize a context the function WDRV_PIC32MZW_BSSCtxSetDefaults is provided, this ensures the context is in a known state.

The function WDRV_PIC32MZW_BSSCtxIsValid is provided to test if a context is valid. In some applications a BSS context is valid even if an SSID is blank therefore this function can be informed whether a blank SSID is valid or not.

The SSID of the BSS can be configured using the function WDRV_PIC32MZW_BSSCtxSetSSID.

The channel of the BSS can be configure using the function WDRV_PIC32MZW_BSSCtxSetChannel.

In some applications a BSS can be cloaked therefore the state can be configured in a BSS context using the function WDRV_PIC32MZW_BSSCtxSetSSIDVisibility to indicate if an BSS is considered visible or not.

```
bool WDRV_PIC32MZW_BSSCtxIsValid
(
        const WDRV_PIC32MZW_BSS_CONTEXT *const pBSSCtx,
        bool ssidValid
)
```

Description:

Tests the elements of the BSS context to judge if their values are legal.

Parameters:

`pBSSCtx`	Pointer to a BSS context.
`ssidValid`	Flag indicating if the SSID within the context must be valid.

Returns:

true or false indicating if context is valid.

Remarks:

A valid SSID is one which has a non-zero length. The check is optional as it is legal for the SSID field to be zero length.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSCtxSetDefaults
(
        WDRV_PIC32MZW_BSS_CONTEXT *const pBSSCtx
)
```

Description:

Ensures that each element of the structure is configured into a legal state.

Parameters:

`pBSSCtx`	Pointer to a BSS context.

Returns:

`WDRV_PIC32MZW_STATUS_OK` - The context has been configured.
`WDRV_PIC32MZW_STATUS_INVALID_ARG` - The parameters were incorrect.

Remarks:

A default context is not valid until it is configured.


```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSCtxSetSSID
(
        WDRV_PIC32MZW_BSS_CONTEXT *const pBSSCtx,
        uint8_t *const pSSID,
        uint8_t ssidLength
)
```

Description:

The SSID string and length provided are copied into the BSS context.

Parameters:

pBSSCtx	Pointer to a BSS context.
pSSID	Pointer to buffer containing the new SSID.
ssidLength	The length of the SSID held in the pSSID buffer.

Returns:

WDRV_PIC32MZW_STATUS_OK	- The context has been configured.
WDRV_PIC32MZW_STATUS_INVALID_ARG	- The parameters were incorrect.
WDRV_PIC32MZW_STATUS_INVALID_CONTEXT	- The BSS context is not valid.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSCtxSetBSSID
(
        WDRV_PIC32MZW_BSS_CONTEXT *const pBSSCtx,
        uint8_t *const pBSSID
)
```

Description:

The BSSID string is copied into the BSS context.

Parameters:

pBSSCtx	Pointer to a BSS context.
pBSSID	Pointer to buffer containing the new BSSID.

Returns:

WDRV_PIC32MZW_STATUS_OK	- The context has been configured.
WDRV_PIC32MZW_STATUS_INVALID_ARG	- The parameters were incorrect.
WDRV_PIC32MZW_STATUS_INVALID_CONTEXT	- The BSS context is not valid.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSCtxSetChannel
(
        WDRV_PIC32MZW_BSS_CONTEXT *const pBSSCtx,
        WDRV_PIC32MZW_CHANNEL_ID channel
)
```

Description:

The supplied channel value is copied into the BSS context.

Parameters:

`pBSSCtx`	Pointer to a BSS context.
`channel`	Channel number.

Returns:

`WDRV_PIC32MZW_STATUS_OK` - The context has been configured.
`WDRV_PIC32MZW_STATUS_INVALID_ARG` - The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_INVALID_CONTEXT` - The BSS context is not valid.

Remarks:

channel may be `WDRV_PIC32MZW_ALL_CHANNELS` to represent no fixed channel.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSCtxSetSSIDVisibility
(
        WDRV_PIC32MZW_BSS_CONTEXT *const pBSSCtx,
        bool visible
)
```

Description:

Specific to Soft-AP mode this flag defines if the BSS context will create a visible presence on air.

Parameters:

`pBSSCtx`	Pointer to a BSS context.
`visible`	Boolean flag value indicating if the BSS will be visible or not.

Returns:

`WDRV_PIC32MZW_STATUS_OK` - The context has been configured.
`WDRV_PIC32MZW_STATUS_INVALID_ARG` - The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_INVALID_CONTEXT` - The BSS context is not valid.

Client Interface - BSS Find

The application can use this interface to request a scan for local BSSs. Once requested, the device will conduct the search using the configured search parameters and report the results back to the application one BSS at a time. This interface can be used in either a callback mode, a polled mode, or a combination of both callback and polled depending on how the application wishes to receive the BSS information.

How a scan is conducted depends on the parameters and channel lists provided by the application.

The channels which can be scanned may be set by calling WDRV_PIC32MZW_BSSFindSetEnabledChannels24.

When an application wishes to begin a scan operation it must call WDRV_PIC32MZW_BSSFindFirst. It is possible to request a scan on only a single channel or on all channels enabled by calling WDRV_PIC32MZW_BSSFindSetEnabledChannels. The scan can be performed using active mode (where probe requests are transmitted) or passive mode (where beacons are listened for).

WDRV_PIC32MZW_BSSFindFirst takes an optional callback function to use for notifying the application when the scan operation is complete, and the first result is available. If this isn't provided, the application can poll this interface using WDRV_PIC32MZW_BSSFindInProgress to determine if the device is still scanning.

<u>Getting Results – Callback Only</u>

If a callback function was provided to WDRV_PIC32MZW_BSSFindFirst the Wi-Fi driver will call this callback when the first results are available. The callback is provided with the scan result for a single BSS as well as the index of the results within the full set of BSSs discovered.

If the callback function returns the value true to the Wi-Fi driver it will cause the driver to request the next result from the PIC32MZW device. When this result is available the Wi-Fi driver will again call the callback and provide the BSS information. It is thus possible to receive all the results via the callback.

<u>Getting Results – Callback Notification, Foreground Retrieval</u>

While the application may wish to be notified of a BSS result being available via the callback mechanism it may be preferable to retrieve the result information from a foreground task. For example, in an OS environment the callback may simply signal a semaphore triggering the main application task to retrieve the BSS information.

In this model the callback called by the Wi-Fi driver should return the value false. The Wi-Fi driver will not request the next set of BSS information from the PIC32MZW device.

The foreground task may then call WDRV_PIC32MZW_BSSFindGetInfo with a pointer to a WDRV_PIC32MZW_BSS_INFO structure to receive the BSS information. If the function is called when there is no valid BSS information present in the PIC32MZW driver the function will return WDRV_PIC32MZW_STATUS_NO_BSS_INFO.

When the application wishes to request the next set of BSS information it must call WDRV_PIC32MZW_BSSFindNext. It is possible to change the callback function at this time or even turn off callback operation if a NULL pointer is used. Assuming the callback function is again specified the Wi-Fi driver will request the next set of BSS information from the device and inform the application via the callback.

<u>Getting Results – Polled</u>

The operations of the BSS scan can be inferred by polling this interface.

WDRV_PIC32MZW_BSSFindInProgress indicates if the scan operation is currently active and results are not yet available.

WDRV_PIC32MZW_BSSFindGetNumBSSResults returns the number of sets of BSS information available as a result of a scan operation.

WDRV_PIC32MZW_BSSFindGetInfo can be called to retrieve the BSS information. If the information is not yet available this function will return WDRV_PIC32MZW_STATUS_NO_BSS_INFO.

Terminating a BSS Search

Once a scan operation has been started by calling WDRV_PIC32MZW_BSSFindFirst it must be allowed to complete, once it has the application may decide to ignore some or all of the results. To abort the BSS information retrieval and abandon any remaining results the application can call WDRV_PIC32MZW_BSSFindReset.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSFindFirst
(
        DRV_HANDLE handle,
        WDRV_PIC32MZW_CHANNEL_ID channel,
        bool active,
        const WDRV_PIC32MZW_BSSFIND_NOTIFY_CALLBACK pfNotifyCallback
)
```

Description:

A scan is requested on the specified channels. An optional callback can be provided to receive notification of the first BSS discovered.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.

Parameters:

`handle`	Client handle obtained by a call to `WDRV_PIC32MZW_Open`.
`channel`	Channel to scan, can be `WDRV_PIC32MZW_ALL_CHANNELS` in which case all enabled channels are scanned.
`active`	Use active vs passive scanning.

`pfNotifyCallback`	Callback to receive notification of first BSS found. A pointer to a function of the following prototype:
	`bool func`
	`(`
	`    DRV_HANDLE handle,`
	`    uint8_t index,`
	`    uint8_t ofTotal,`
	`    WDRV_PIC32MZW_BSS_INFO *pBssInfo`
	`)`

Returns:

`WDRV_PIC32MZW_STATUS_OK` - A scan was initiated.
`WDRV_PIC32MZW_STATUS_NOT_OPEN` - The driver instance is not open.
`WDRV_PIC32MZW_STATUS_REQUEST_ERROR` - The request to the PIC32MZW1 was rejected.
`WDRV_PIC32MZW_STATUS_INVALID_ARG` - The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_SCAN_IN_PROGRESS` - A scan is already in progress.

Remarks:

If channel is `WDRV_PIC32MZW_ALL_CHANNELS` then all enabled channels are scanned. The enabled channels can be configured using `WDRV_PIC32MZW_BSSFindSetEnabledChannels`. How the scan is performed can be configured using `WDRV_PIC32MZW_BSSFindSetScanParameters`.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSFindNext
(
        DRV_HANDLE handle,
        WDRV_PIC32MZW_BSSFIND_NOTIFY_CALLBACK pfNotifyCallback
)
```

Description:

The information structure of the next BSS is requested from the PIC32MZW1.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.
`WDRV_PIC32MZW_BSSFindFirst` must have been called.

Parameters:

`Handle`	Client handle obtained by a call to `WDRV_PIC32MZW_Open`.

`pfNotifyCallback`	Callback to receive notification of next BSS found. A pointer to a function of the following prototype:
	`bool func`
	`(`
	`DRV_HANDLE handle,`
	`uint8_t index,`
	`uint8_t ofTotal,`
	`WDRV_PIC32MZW_BSS_INFO *pBssInfo`
	`)`

Returns:

`WDRV_PIC32MZW_STATUS_OK` - The request was accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN` - The driver instance is not open.
`WDRV_PIC32MZW_STATUS_INVALID_ARG` - The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_SCAN_IN_PROGRESS` - A scan is already in progress.
`WDRV_PIC32MZW_STATUS_BSS_FIND_END` - No more results are available.


```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSFindReset
(
        DRV_HANDLE handle,
        WDRV_PIC32MZW_BSSFIND_NOTIFY_CALLBACK pfNotifyCallback
)
```

Description:

The information structure of the first BSS is requested from the PIC32MZW1.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.
`WDRV_PIC32MZW_BSSFindFirst` must have been called.

Parameters:

`handle`	Client handle obtained by a call to `WDRV_PIC32MZW_Open`.
`pfNotifyCallback`	Callback to receive notification of next BSS found. A pointer to a function of the following prototype: `bool func ( DRV_HANDLE handle, uint8_t index, uint8_t ofTotal, WDRV_PIC32MZW_BSS_INFO *pBssInfo )`

Returns:

`WDRV_PIC32MZW_STATUS_OK`	- The request was accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN`	- The driver instance is not open.
`WDRV_PIC32MZW_STATUS_INVALID_ARG`	- The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_SCAN_IN_PROGRESS`	- A scan is already in progress.
`WDRV_PIC32MZW_STATUS_BSS_FIND_END`	- No more results are available.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSFindGetInfo
(
        DRV_HANDLE handle,
        WDRV_PIC32MZW_BSS_INFO *const pBSSInfo
)
```

Description:

After each call to either `WDRV_PIC32MZW_BSSFindFirst` or `WDRV_PIC32MZW_BSSFindNext` the driver receives a single BSS information structure which it stores. This function retrieves that structure.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.
`WDRV_PIC32MZW_BSSFindFirst` must have been called.

Parameters:

`handle`	Client handle obtained by a call to WDRV_PIC32MZW_Open.
`pBSSInfo`	Pointer to structure to populate with the current BSS information.

Returns:

`WDRV_PIC32MZW_STATUS_OK` - The request was accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN` - The driver instance is not open.
`WDRV_PIC32MZW_STATUS_INVALID_ARG` - The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_NO_BSS_INFO` - There is no current BBS information available.

Remarks:

This function may be polled after calling `WDRV_PIC32MZW_BSSFindFirst` or `WDRV_PIC32MZW_BSSFindNext` until it returns `WDRV_PIC32MZW_STATUS_OK`.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSFindSetScanParameters
(
        DRV_HANDLE handle,
        uint16_t activeScanTime,
        uint16_t passiveListenTime
)
```

Description:

This function set various parameters to control the scan behavior.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.

Parameters:

`handle`	Client handle obtained by a call to WDRV_PIC32MZW_Open.
`activeScanTime`	Time spent on each active channel probing for BSS's.
`passiveListenTime`	Time spent on each passive channel listening for beacons.

Returns:

`WDRV_PIC32MZW_STATUS_OK` - The request was accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN` - The driver instance is not open.
`WDRV_PIC32MZW_STATUS_INVALID_ARG` - The parameters were incorrect.

Remarks:

If any parameter is zero then the configured value is unchanged.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_BSSFindSetEnabledChannels24
(
        DRV_HANDLE handle,
        WDRV_PIC32MZW_CHANNEL24_MASK channelMask24
)
```

Description:

To comply with regulatory domains certain channels must not be scanned. This function configures which channels are enabled to be used.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.

Parameters:

`handle`	Client handle obtained by a call to WDRV_PIC32MZW_Open.
`channelMask24`	A 2.4GHz channel mask detailing all the enabled channels.

Returns:

`WDRV_PIC32MZW_STATUS_OK` - The request was accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN` - The driver instance is not open.
`WDRV_PIC32MZW_STATUS_INVALID_ARG` - The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_REQUEST_ERROR` - The PIC32MZW1 was unable to accept this request.

```
uint8_t WDRV_PIC32MZW_BSSFindGetNumBSSResults(DRV_HANDLE handle)
```

Description:

Returns the number of BSS scan results found.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.
`WDRV_PIC32MZW_BSSFindFirst` must have been called to start a scan.

Parameters:

`handle`	Client handle obtained by a call to WDRV_PIC32MZW_Open.

Returns:

Number of BSS scan results available. Zero indicates no results or an error occurred.

```
bool WDRV_PIC32MZW_BSSFindInProgress(DRV_HANDLE handle)
```

Description:

Returns a flag indicating if a BSS scan operation is currently running.

Preconditions:

1. `WDRV_PIC32MZW_Initialize` must have been called.
2. `WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.

Parameters:

`handle`	Client handle obtained by a call to WDRV_PIC32MZW_Open.

Returns:

Flag indicating if a scan is in progress. If an error occurs the result is false.

Client Interface - Association

This interface provides information about the current association with a peer device.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_AssocPeerAddressGet
(
        WDRV_PIC32MZW_ASSOC_HANDLE assocHandle,
        WDRV_PIC32MZW_MAC_ADDR *const pPeerAddress
)
```

Description:

Attempts to retrieve the network address of the peer device in the current association.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.
A peer device needs to be connected and associated, the association handle should be obtained from the `WDRV_PIC32MZW_BSSCON_NOTIFY_CALLBACK` callback

Parameters:

`assocHandle`	Association handle.
`pPeerAddress`	Pointer to structure to receive the network address.

Returns:

`WDRV_PIC32MZW_STATUS_OK`	The request was accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN`	The driver instance is not open.
`WDRV_PIC32MZW_STATUS_INVALID_ARG`	The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_REQUEST_ERROR`	The PIC32MZW1 was unable to accept this request.
`WDRV_PIC32MZW_STATUS_RETRY_REQUEST`	The network address is not available, but it will be requested from the PIC32MZW1.
`WDRV_PIC32MZW_STATUS_NOT_CONNECTED`	Not currently connected.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_AssocRSSIGet
(
        WDRV_PIC32MZW_ASSOC_HANDLE assocHandle,
        int8_t *const pRSSI,
        WDRV_PIC32MZW_ASSOC_RSSI_CALLBACK const pfAssociationRSSICB
)
```

Description:

Attempts to retrieve the RSSI of the current association.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.
A peer device needs to be connected and associated.

Parameters:

`assocHandle`	Association handle.
`pRSSI`	Pointer to variable to receive RSSI if available.
`pfAssociationRSSICB`	Pointer to callback function to be used when RSSI value is available.

Returns:

`WDRV_PIC32MZW_STATUS_OK` - The request was accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN` - The driver instance is not open.
`WDRV_PIC32MZW_STATUS_INVALID_ARG` - The parameters were incorrect.
`WDRV_PIC32MZW_STATUS_REQUEST_ERROR` - PIC32MZW1 was unable to accept request.
`WDRV_PIC32MZW_STATUS_RETRY_REQUEST` - Network address unavailable.
`WDRV_PIC32MZW_STATUS_NOT_CONNECTED` - Not currently connected.

Client Interface - Information

This interface provides general information about the device.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_InfoDeviceMACAddressGet
(
        DRV_HANDLE handle, uint8_t *const pMACAddress
)
```

Description:

Retrieves the current working MAC address.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.

Parameters:

handle	Client handle obtained by a call to WDRV_PIC32MZW_Open.
pMACAddress	Pointer to buffer (of at least 6 bytes in length) to receive the MAC address.

Returns:

WDRV_PIC32MZW_STATUS_OK	- The information has been returned.
WDRV_PIC32MZW_STATUS_NOT_OPEN	- The driver instance is not open.
WDRV_PIC32MZW_STATUS_INVALID_ARG	- The parameters were incorrect.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_InfoOpChanGet
(
        DRV_HANDLE handle, WDRV_PIC32MZW_CHANNEL_ID *const pMACAddress
)
```

Description:

Retrieves the current operating channel.

Preconditions:

`WDRV_PIC32MZW_Initialize` must have been called.
`WDRV_PIC32MZW_Open` must have been called to obtain a valid handle.

Parameters:

handle	Client handle obtained by a call to `WDRV_PIC32MZW_Open`.
pOpChan	Pointer to variable to receive the operating channel.

Returns:

WDRV_PIC32MZW_STATUS_OK	- The information has been returned.
WDRV_PIC32MZW_STATUS_NOT_OPEN	- The driver instance is not open.
WDRV_PIC32MZW_STATUS_INVALID_ARG	- The parameters were incorrect.

Client Interface - Regulatory Domain

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_RegDomainSet
(
        DRV_HANDLE handle,
        const char *pRegDomain,
        const WDRV_PIC32MZW_REGDOMAIN_CALLBACK pfRegDomCallback
)
```

Description:

Requests that the current regulatory domain is changed to that specified.

Preconditions:

`WDRV_PIC32MZW_Initialize` should have been called.
`WDRV_PIC32MZW_Open` should have been called to obtain a valid handle.

Parameters:

<table>
<tr><td><code>handle</code></td><td>Client handle obtained by a call to WDRV_PIC32MZW_Open.</td></tr>
<tr><td><code>pRegDomain</code></td><td>Pointer to a string name of the regulatory domain.
<table>
<tr><th>String</th><th>Region</th></tr>
<tr><td>GEN</td><td>Generic/world-wide region</td></tr>
<tr><td>USA</td><td>North America</td></tr>
<tr><td>EMEA</td><td>Europe</td></tr>
<tr><td>JPN</td><td>Japan</td></tr>
<tr><td>CUST1</td><td>Provided for customer configuration (chip-down)</td></tr>
<tr><td>CUST2</td><td>Provided for customer configuration (chip-down)</td></tr>
</table>
Please note that these are actually ‘free format’ strings; the above table presents values that will commonly appear in pre-programmed modules.</td></tr>
<tr><td><code>pfRegDomCallback</code></td><td>Pointer to callback function to receive confirmation that the regulatory domain that has been set as instructed. A pointer to a function of the following prototype:
<pre>
bool func
(
        DRV_HANDLE handle,
        uint8_t index,
        uint8_t ofTotal,
        bool isCurrent,
        const char * pRegDomain,
)
</pre>
When called in response to a SET request, the callback will receive a single message indicating:
<code>index=1</code> & <code>ofTotal = 1</code> i.e. 1 of 1 messages
<code>isCurrent = True|False</code> <code>True</code> indicating success
<code>pRegDomain</code> Contains the name of the region</td></tr>
</table>

Returns:

`WDRV_PIC32MZW_STATUS_OK` - The request has been accepted.
`WDRV_PIC32MZW_STATUS_NOT_OPEN` - The driver instance is not open.

WDRV_PIC32MZW_STATUS_INVALID_ARG - The parameters were incorrect.
WDRV_PIC32MZW_STATUS_REQUEST_ERROR - The request to the PIC32MZW was rejected.

```
WDRV_PIC32MZW_STATUS WDRV_PIC32MZW_RegDomainGet
(
        DRV_HANDLE handle,
        const WDRV_PIC32MZW_REGDOMAIN_SELECT selection,
        const WDRV_PIC32MZW_REGDOMAIN_CALLBACK pfRegDomCallback
)
```

Description:

Requests either the name of the currently active regulatory domain or the names of all regulatory domains programmed into the module.

Preconditions:

WDRV_PIC32MZW_Initialize should have been called.
WDRV_PIC32MZW_Open should have been called to obtain a valid handle.

Parameters:

handle	Client handle obtained by a call to WDRV_PIC32MZW_Open.
selection	Type of regulatory domain information to retrieve: • WDRV_PIC32MZW_REGDOMAIN_SELECT_CURRENT • WDRV_PIC32MZW_REGDOMAIN_SELECT_ALL

pfRegDomCallback	Pointer to callback function to receive the requested regulatory domain information. A pointer to a function of the following prototype:
	
	`bool func`
	`(`
	`DRV_HANDLE handle,`
	`uint8_t index,`
	`uint8_t ofTotal,`
	`bool isCurrent,`
	`const char * pRegDomain,`
	`)`
	
	When called in response to a GET request, the callback will executed multiple times (once for each region) indicating:
	index=x & ofTotal = y i.e. x of y messages
	isCurrent = True|False True indicating region is active
	pRegDomain Contains the name of the region

Returns:

WDRV_PIC32MZW_STATUS_OK - The request has been accepted.
WDRV_PIC32MZW_STATUS_NOT_OPEN - The driver instance is not open.
WDRV_PIC32MZW_STATUS_INVALID_ARG - The parameters were incorrect.
WDRV_PIC32MZW_STATUS_REQUEST_ERROR - The request to the PIC32MZW was rejected.

7. APPENDIX – A: Configuration Bits

The sections provide the details (suggests) of the configuration bits used for all of the application examples. The configuration bit details are part of the `initialization.c` file in any of the Harmony Projects.

```
// ********************************************************************
// ********************************************************************
// Section: Configuration Bits
// ********************************************************************
// ********************************************************************
/*** FBCFG0 ***/
#pragma config BUHSWEN    = OFF
#pragma config PCSCMODE   = DUAL
#pragma config BOOTISA    = MIPS32


/*** DEVCFG0 ***/
#pragma config TDOEN      = ON
#pragma config TROEN      = OFF
#pragma config JTAGEN     = OFF
#pragma config FCPRI      = LRSA
#pragma config DMAPRI     = LRSA
#pragma config EXLPRI     = LRSA
#pragma config USBSSEN    = OFF
#pragma config PMULOCK    = OFF
#pragma config PGLOCK     = OFF
#pragma config PMDLOCK    = OFF
#pragma config IOLOCK     = OFF
#pragma config CFGLOCK    = OFF
#pragma config OC_ACLK    = OCMP_TMR2_TMR3
#pragma config IC_ACLK    = ICAP_TMR2_TMR3
#pragma config CANFDDIV   = 0
#pragma config PCM        = SFR
#pragma config UPLLHWMD   = OFF
#pragma config SPLLHWMD   = OFF
#pragma config BTPLLHWMD  = OFF
#pragma config ETHPLLHWMD = OFF
#pragma config FECCCON    = OFF
#pragma config ETHTPSF    = RPSF
#pragma config EPGMCLK    = FRC


/*** DEVCFG1 ***/
#pragma config DEBUG      = EMUC
#pragma config ICESEL     = ICS_PGx2
#pragma config TRCEN      = ON
#pragma config FMIIEN     = OFF
#pragma config ETHEXEREF  = OFF
#pragma config CLASSBDIS  = DISABLE
#pragma config USBIDIO    = ON
```

```
#pragma config VBUSIO     = ON
#pragma config HSSPIEN    = OFF
#pragma config SMCLR      = MCLR_NORM
#pragma config USBDMTRIM  = 0
#pragma config USBDPTRIM  = 0
#pragma config HSUARTEN   = ON
#pragma config WDTPSS     = PSS1


/*** DEVCFG2 ***/
#pragma config DMTINTV    = WIN_63_64
#pragma config POSCMOD    = HS
#pragma config WDTRMCS    = LPRC
#pragma config SOSCSEL    = CRYSTAL
#pragma config WAKE2SPD   = ON
#pragma config CKSWEN     = ON
#pragma config FSCMEN     = ON
#pragma config WDTPS      = PS1
#pragma config WDTSPGM    = STOP
#pragma config WINDIS     = NORMAL
#pragma config WDTEN      = OFF
#pragma config WDTWINSZ   = WINSZ_25
#pragma config DMTCNT     = DMT31
#pragma config DMTEN      = OFF



/*** DEVCFG4 ***/
#pragma config SOSCCFG    = 0
#pragma config VBZPBOREN  = ON
#pragma config DSZPBOREN  = ON
#pragma config DSWDTPS    = DSPS1
#pragma config DSWDTOSC   = LPRC
#pragma config DSWDTEN    = OFF
#pragma config DSEN       = OFF
#pragma config SOSCEN     = OFF
```

The Microchip Web Site

Microchip provides online support via our web site at http://www.microchip.com/. This web site is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the web site contains the following information:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQ), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Customer Change Notification Service

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, access the Microchip web site at http://www.microchip.com/. Under "Support", click on "Customer Change Notification" and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

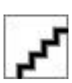
- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or Field Application Engineer (FAE) for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in the back of this document.

Technical support is available through the web site at: http://www.microchip.com/support

Product Identification System

To order or obtain information, e.g., on pricing or delivery, refer to the factory or the listed sales office.



Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN:

Quality Management System Certified by DNV

ISO/TS 16949

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: http://www.microchip.com/ support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4450-2828 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-67-3636 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-7289-7561 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820