

Class B Library Porting Guide

Table of Contents

1. Introduction.....	3
2. Conventions Followed in this Document.....	4
3. Development Environment.....	5
4. Key Points.....	6
5. Software Revision.....	7
6. System Overview.....	8
7. Porting ADC Module.....	9
8. Porting CLOCK Module.....	11
9. Porting CRC Module.....	12
10. Porting CPU Module.....	13
11. Porting FLASH Module.....	14
12. Porting GPIO Module.....	15
13. Porting INTERRUPT Module.....	16
14. Porting PC Module.....	17
15. Porting SRAM Module.....	18
16. Porting TIMER Module.....	19
17. Source Code Reference.....	20
Microchip Information.....	22
Trademarks.....	22
Legal Notice.....	22
Microchip Devices Code Protection Feature.....	22

1. Introduction

The intention of this document is to provide guidance to the system integrators who wish to port the Class B library to variants within a device family. Although most of the things remain the same, there are some configuration entities which need to be modified in certain modules in accordance with the variants in the same family.

2. Conventions Followed in this Document

- Source code references in terms of API names, macros and variable names are in a different font.

Eg: `DIAG_ADC_LinearityMonotonicityTest()`

`#define DIAG_INTERRUPT_IVTCIF 0U`

- Header/Source file references are also in the same font as that of source code references Eg:

`diag_interrupt_shared.h`

3. Development Environment

The diagnostics referred to in this document are almost entirely written in C, with some implementations in assembly language. The diagnostics development is done on MPLAB X IDE v6.25 or above and the compiler used is XC-DSC v3.31 or above.

4. Key Points

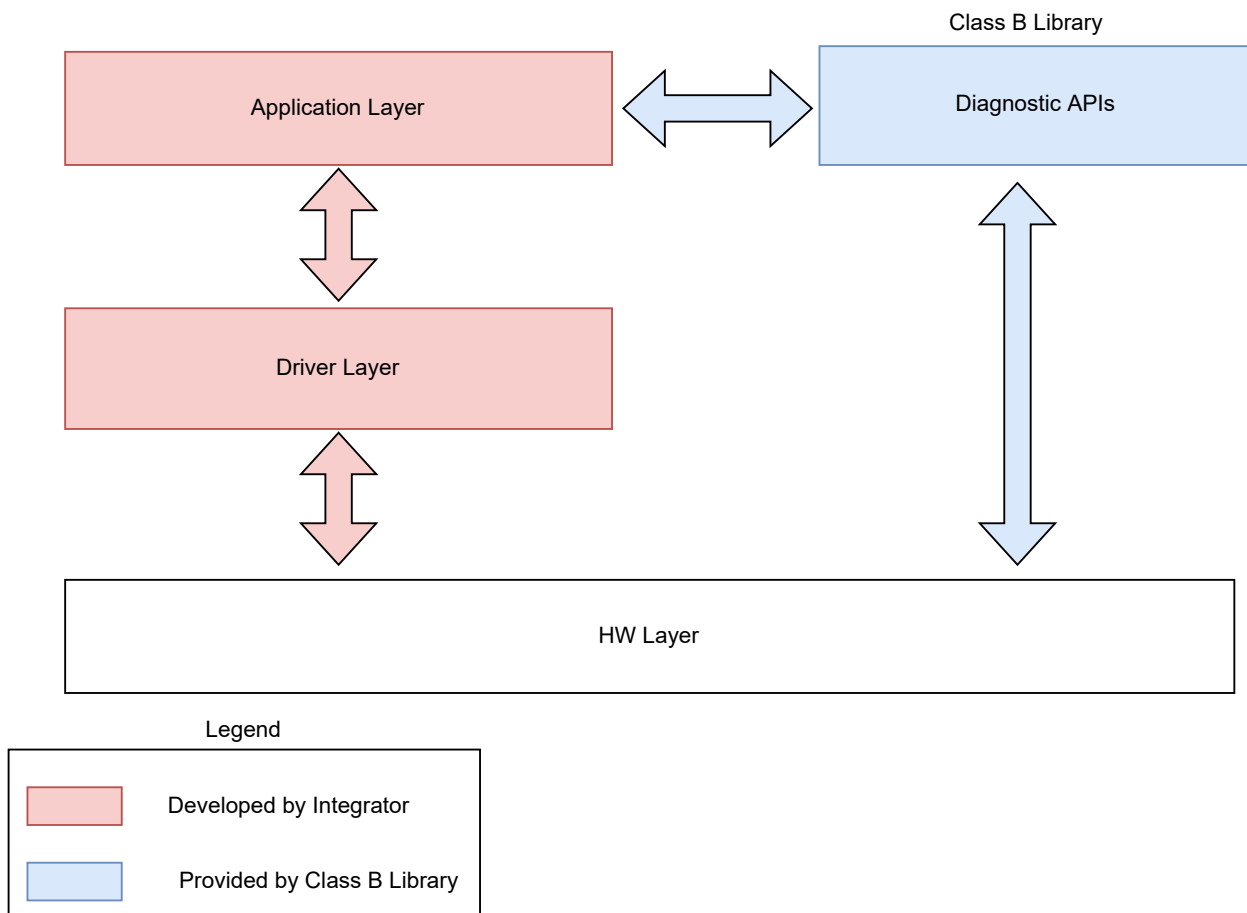
1. The Class B library for some of the modules are based on an instance of the module available in the variant of the family. Hence it is imperative that the system integrator is aware of the device datasheet and its capabilities before the migration activity.
2. Class B library APIs in some modules are dependent on the frequency at which the CPU is running. All the modules are tested to work correctly at the maximum frequency at which the CPU runs. If the system integrator wishes to run the CPU at a lesser frequency for whatever reason, some configuration entities may need to be tweaked.
3. Compiler Optimization level plays a key role in both size of the code and the speed of its execution. All the diagnostics are tested to work correctly with -O2 option.
4. One of the major differences between variants of the same device family is the pin outs available for each variant. More so, for the lower variants of the family with lesser pin outs available, one feature that can be explored is the usage of virtual pins while using some diagnostics. Refer the device datasheet for information about virtual pins and the methodologies to use them.

5. Software Revision

The dsPIC33A Class B library referred to in this document is at version v3.0.0

6. System Overview

The Class B library can be seamlessly integrated to the application as they are self-contained entities and directly interact with the hardware with its internally implemented driver functions. The following diagram depicts the same



The table below shows the responsibilities of the system integrator and what is already available at their disposal provided by Class B library.

Module	Description
Application Layer	The user application code needs to be developed by the system integrator. The test application code provided along with the release provides information as to how the modules have to be initialized.
Class B Library	The component of the block diagram has Class B library APIs as well as the internal driver layer which directly interacts with the hardware layer.
Driver Layer	This can be used by the application layer to interact with the hardware layer.

7. Porting ADC Module

The following diagnostic APIs are available for the ADC module

- DIAG_ADC_StartupTest
- DIAG_ADC_LinearityMonotonicityTest

The number of ADC instances and the channels associated with those instances in any variant in the family varies and this information shall be updated in `diag_adc.h`, `diag_adc_shared.h` and `diag_adc_shared.c`

For eg:

The following macros, enums, variables are available for the dsPIC33AK512MPS512 and these shall be updated based on the device in use.

```
#define DIAG_ADC_MAX_INSTANCE 5U
#define DIAG_ADC_NEXT_CHANNEL_ADDRESS 0x0020UL
uint16_t nextADCInstanceAddress [DIAG_ADC_MAX_INSTANCE]
uint16_t totalChannelsADCInstance [DIAG_ADC_MAX_INSTANCE]
```

```
typedef enum
{
    DIAG_ADC1_CH0 = 0x0000U,
    DIAG_ADC1_CH1 = 0x0001U,
    DIAG_ADC1_CH2 = 0x0002U,
    DIAG_ADC1_CH3 = 0x0003U,
    DIAG_ADC1_CH4 = 0x0004U,
    DIAG_ADC1_CH5 = 0x0005U,
    DIAG_ADC1_CH6 = 0x0006U,
    DIAG_ADC1_CH7 = 0x0007U,
    DIAG_ADC2_CH0 = 0x0100U,
    DIAG_ADC2_CH1 = 0x0101U,
    DIAG_ADC2_CH2 = 0x0102U,
    DIAG_ADC2_CH3 = 0x0103U,
    DIAG_ADC2_CH4 = 0x0104U,
    DIAG_ADC2_CH5 = 0x0105U,
    DIAG_ADC2_CH6 = 0x0106U,
    DIAG_ADC2_CH7 = 0x0107U,
    DIAG_ADC3_CH0 = 0x0200U,
    DIAG_ADC3_CH1 = 0x0201U,
    DIAG_ADC3_CH2 = 0x0202U,
    DIAG_ADC3_CH3 = 0x0203U,
    DIAG_ADC3_CH4 = 0x0204U,
    DIAG_ADC3_CH5 = 0x0205U,
    DIAG_ADC3_CH6 = 0x0206U,
    DIAG_ADC3_CH7 = 0x0207U,
    DIAG_ADC4_CH0 = 0x0300U,
    DIAG_ADC4_CH1 = 0x0301U,
    DIAG_ADC4_CH2 = 0x0302U,
    DIAG_ADC4_CH3 = 0x0303U,
    DIAG_ADC4_CH4 = 0x0304U,
    DIAG_ADC4_CH5 = 0x0305U,
    DIAG_ADC4_CH6 = 0x0306U,
    DIAG_ADC4_CH7 = 0x0307U,
    DIAG_ADC5_CH0 = 0x0400U,
    DIAG_ADC5_CH1 = 0x0401U,
    DIAG_ADC5_CH2 = 0x0402U,
    DIAG_ADC5_CH3 = 0x0403U,
    DIAG_ADC5_CH4 = 0x0404U,
    DIAG_ADC5_CH5 = 0x0405U,
    DIAG_ADC5_CH6 = 0x0406U,
    DIAG_ADC5_CH7 = 0x0407U,
    DIAG_ADC5_CH8 = 0x0408U,
    DIAG_ADC5_CH9 = 0x0409U,
    DIAG_ADC5_CH10 = 0x040AU,
    DIAG_ADC5_CH11 = 0x040BU,
    DIAG_ADC5_CH12 = 0x040CU,
    DIAG_ADC5_CH13 = 0x040DU,
```

```
    DIAG_ADC5_CH14 = 0x040EU,  
    DIAG_ADC5_CH15 = 0x040FU  
}DIAG_ADC_CHANNELS;
```

Porting Instructions

Update the macros and enums for the specific device variant as needed. No additional changes are required for the diagnostics.

8. Porting CLOCK Module

The following diagnostic APIs are available for the CLOCK module

- `DIAG_CLOCK_FscmTest`

Porting Instructions

No modifications are required for other variants of the same device family

9. Porting CRC Module

The following diagnostic APIs are available for the CRC module.

- DIAG_CRC_FunctionalTest

Porting Instructions

No modifications are required for other variants of the same device family

10. Porting CPU Module

The following diagnostic APIs are available for the CPU module

- `DIAG_CPU_SelfTest`
- `DIAG_CPU_ControlRegisterTest`
- `DIAG_CPU_RegisterResetStateCheck`

Porting Instructions:

No modifications are required for other variants in the same device family

11. Porting FLASH Module

The following diagnostic APIs are available for the FLASH module

- `DIAG_FLASH_CRCCalculate`
- `DIAG_FLASH_CRCPractice`
- `DIAG_FLASH_WriteVerifyPractice`
- `DIAG_FLASH_IntegrityReadPractice`
- `DIAG_FLASH_SingleDoubleErrorDetectionTest`

Porting Instructions:

No modifications are required for other variants of the same device family

12. Porting GPIO Module

The following diagnostic APIs are available for the GPIO Module

- `DIAG_GPIO_InputPractice`
- `DIAG_GPIO_OutputTest`
- `DIAG_GPIO_ActivityCheck`
- `DIAG_GPIO_InterruptGenTest`
- `DIAG_GPIO_PpsOutputConnectionTest`
- `DIAG_GPIO_IntegrityMonitorTest`

Different variants of a device family vary in terms of the number of port registers available. For eg in dsPIC33AK512MPS512 there are 8 port registers available and these are mapped to the corresponding macros defined in `diag_gpio.h` as shown below

```
#define PORTA_PTR &PORTA
#define PORTB_PTR &PORTB
#define PORTC_PTR &PORTC
#define PORTD_PTR &PORTD
#define PORTE_PTR &PORTE
#define PORTF_PTR &PORTF
#define PORTG_PTR &PORTG
#define PORTH_PTR &PORTH
```

If the device variant has additional port registers, those must be defined accordingly in `diag_ports.h`

Porting Instructions

Define additional macros or remove existing macros so that the port registers that are available in the device variant are defined correctly. No other changes are required for the diagnostics.

These devices are equipped with Input Output Integrity Module which is also tested as a part of the diagnostic, using the `DIAG_GPIO_IntegrityMonitorTest` API. The number of integrity module instances for a device is defined using the macro `#define NUM_IOIM_INSTANCES`. For eg for the dsPIC33AK512MPS512 device, it is 16. Based on the variant this macro has to be modified. Also the diagnostic works on the registers of the I/O integrity module, and hence the base address for each instance is required. These base addresses have to be modified according to the device variant. This is defined as an array `static uint32_t ioimBaseAddr[NUM_IOIM_INSTANCES]` in the `diag_gpio_iomonitortest.c`

The base address for the Integrity monitor instance 1 is 0x1E90 and for instance 2 is 0x1E9C and so on. This has to be modified according to the device variant.



Attention: The GPIO module diagnostics uses the delay routines defined in `diag_delay.c`. The users have to pass the system clock frequency value to the API `DIAG_SetClockFreq()` and call this API in their main application. If the API is not called, then this may result in improper functioning of the GPIO module diagnostics.

13. Porting INTERRUPT Module

The following diagnostics are available in the Interrupt module

- `DIAG_INTERRUPT_ServicingTest`
- `DIAG_INTERRUPT_FrequencyCheck`
- `DIAG_INTERRUPT_ExternalInputTest`
- `DIAG_INTERRUPT_HardTrapTest`
- `DIAG_INTERRUPT_IsrClearedCheck`

Porting Instructions

No modifications are required for other variants of the same device family.

14. Porting PC Module

The following diagnostic APIs are available for the PC module

- `DIAG_PC_ProgramCounterTest`

Porting Instructions:

No modifications are required for other variants of the same device family

15. Porting SRAM Module

The following diagnostic APIs are available for the SRAM module

- `DIAG_SRAM_eccEventInterruptTest`
- `DIAG_SRAM_IsBackedUpDataValid`
- `DIAG_SRAM_ReplicationWrite`

Porting Instructions

No modification is required for other variants of the same device family

16. Porting TIMER Module

The following diagnostic APIs are available for the timer module:

- DIAG_TIMER_FunctionalTest
- DIAG_TIMER_LinearityTest

Porting DIAG_TIMER_FunctionalTest API

The number of TIMER instances in any variant in the family varies and this information shall be updated in `diag_utility_timer.h`

The register data for these timer instances shall be updated in the lookup table in `diag_utility_timer.c`

For Eg:

The following macros, enums, variables are available for the dsPIC33AK512MPS512 and these shall be updated based on the device in use.

```
#define DIAG_UTILITY_TIMER_TOTAL 12U

const DIAG_UTILITY_TIMER_DATA_LOOKUP
timerDataLookup[DIAG_UTILITY_TIMER_TOTAL]
```

```
typedef enum
{
    DIAG_UTILITY_TIMER_TMR1,
    DIAG_UTILITY_TIMER_TMR2,
    DIAG_UTILITY_TIMER_TMR3,
    DIAG_UTILITY_TIMER_SCCP1,
    DIAG_UTILITY_TIMER_SCCP2,
    DIAG_UTILITY_TIMER_SCCP3,
    DIAG_UTILITY_TIMER_SCCP4,
    DIAG_UTILITY_TIMER_SCCP5,
    DIAG_UTILITY_TIMER_SCCP6,
    DIAG_UTILITY_TIMER_SCCP7,
    DIAG_UTILITY_TIMER_SCCP8,
    DIAG_UTILITY_TIMER_MCCP9
}DIAG_UTILITY_TIMER_INSTANCE;
```

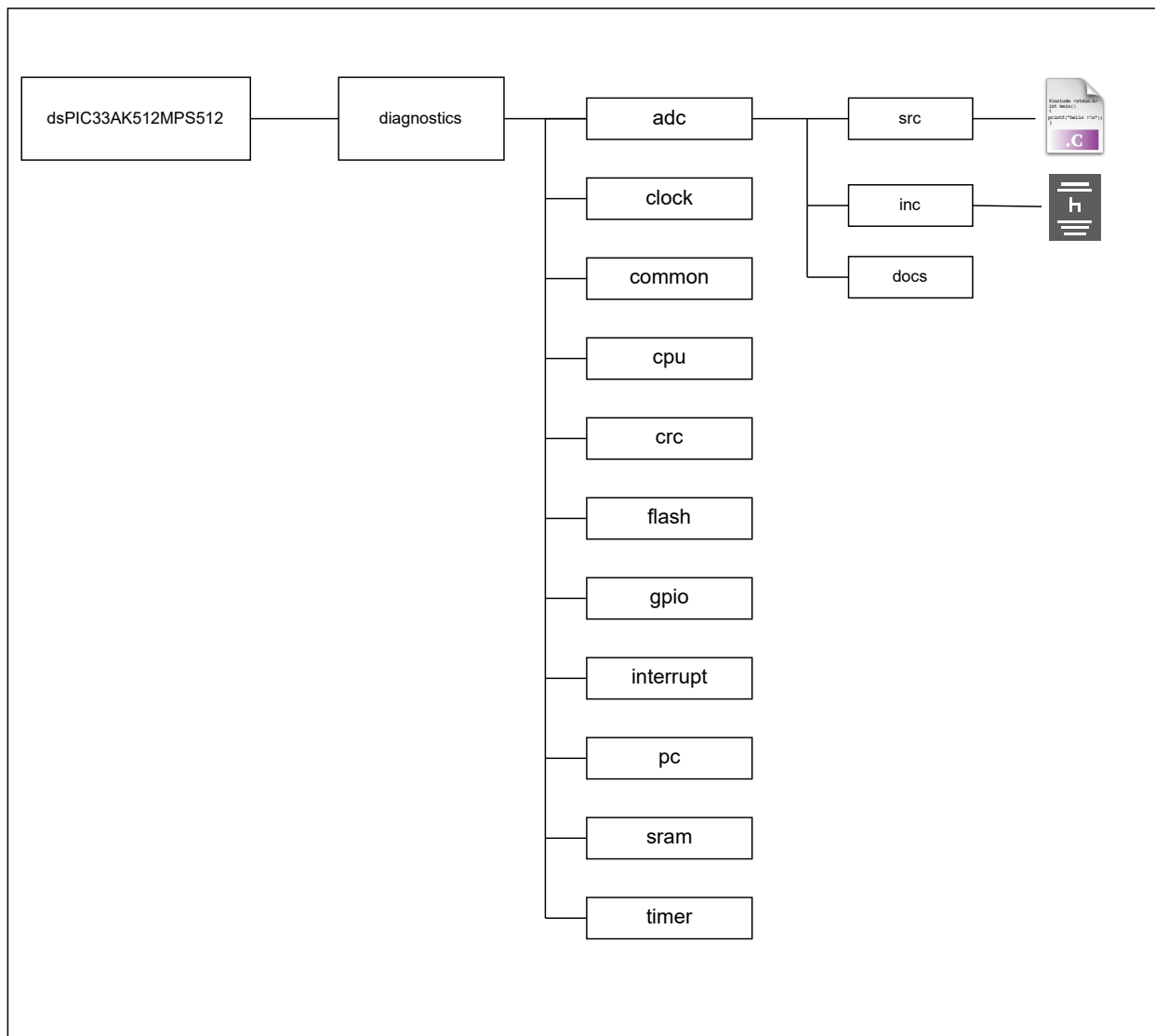
Porting Instructions:

No changes are required in the diagnostic API for the proper functionality

17. Source Code Reference

The directory structure for the Class B library for each variant consists of diagnostics. As an example, the directory structure for one of the devices dsPIC33AK512MPS512 is as shown below:

Folder Structure



The following table provides a list of APIs in the source code:

MODULE	DIAGNOSTIC API	SOURCE CODE REFERENCE
INTERRUPT	DIAG_INTERRUPT_ServicingTest	diag_interrupt_servicing_test.c
	DIAG_INTERRUPT_FrequencyCheck	diag_interrupt_frequency_check.c
	DIAG_INTERRUPT_ExternalInputTest	diag_interrupt_external_input.c
	DIAG_INTERRUPT_HardTrapTest	diag_interrupt_hard_trap.c
	DIAG_INTERRUPT_IsrClearedCheck	diag_interrupt_isrcleared_check.c
TIMER	DIAG_TIMER_FunctionalTest	diag_timer.c
	DIAG_TIMER_LinearityTest	

CPU	DIAG_CPU_ControlRegisterTest	diag_cpu_control_register_test.c
	DIAG_CPU_SelfTest	diag_cpu_selftest.c
	DIAG_CPU_RegResetStateTest	diag_cpu_regresetstate_test.c
FLASH	DIAG_FLASH_SingleDoubleErrorDetectionTest	diag_flash_biterror_test.c
	DIAG_FLASH_IntegrityReadPractice	diag_flash_integrity_practice.c
	DIAG_FLASH_WriteVerifyPractice	diag_flash_writeverify_test.c
	DIAG_FLASH_CRCCalculate	diag_flash_crc_test.c
	DIAG_FLASH_CRCPractice	
SRAM	DIAG_SRAM_ReplicationWrite	diag_sram_replication.c
	DIAG_SRAM_IsBackedUpDataValid	
	DIAG_SRAM_eccEventInterruptTest	diag_sram_ecc_event.c
ADC	DIAG_ADC_StartupTest	diag_adc_startuptest.c
	DIAG_ADC_BoundaryCheck	diag_adc_boundarycheck.c
	DIAG_ADC_LinearityMonotonicityTest	diag_adc_linearitymonotonicitytest.c
GPIO	DIAG_GPIO_InputPractice	diag_gpio_inputpractice.c
	DIAG_GPIO_OutputTest	diag_gpio_outputtest.c
	DIAG_GPIO_ActivityCheck	diag_gpio_activitycheck.c
	DIAG_GPIO_InterruptGenTest	diag_gpio_interruptgentest.c
	DIAG_GPIO_PpsOutputConnectionTest	diag_gpio_ppsoutputtest.c
	DIAG_GPIO_IntegrityMonitorTest	diag_gpio_inmonitortest.c
CRC	DIAG_CRC_FunctionalTest	diag_crc_functional_test.c
PC	DIAG_PC_ProgramCounterTest	diag_pc_programcountertest.c
CLOCK	DIAG_CLOCK_FscmTest	diag_clock_fscmtest.c

Microchip Information

Trademarks

The “Microchip” name and logo, the “M” logo, and other names, logos, and brands are registered and unregistered trademarks of Microchip Technology Incorporated or its affiliates and/or subsidiaries in the United States and/or other countries (“Microchip Trademarks”). Information regarding Microchip Trademarks can be found at <https://www.microchip.com/en-us/about/legal-information/microchip-trademarks>.

ISBN:

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP “AS IS”. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP’S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Microchip Devices Code Protection Feature

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

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip products are strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is “unbreakable”. Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.