PAC194X/5X UEFI BIOS Integration and Microsoft® Windows® 10 and Windows 11 Energy Meter Interface Device Driver User's Guide

© 2021-2022 Microchip Technology Inc. and its subsidiaries

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 product is 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.

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 Information regarding device applications is provided only for your convenience and may be superseded by updates.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 https://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.

For information regarding Microchip’s Quality Management Systems, please visit www.microchip.com/quality.

Trademarks

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

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

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, GridTime, IdealBridge, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, Inter-Chip Connectivity, JitterBlocker, Knob-on-Display, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, MultiTRAK, NetDetach, NVM Express, NVMe, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SmartHLS, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, TSHARC, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

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

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

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

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

© 2021-2022, Microchip Technology Incorporated and its subsidiaries.

All Rights Reserved.

ISBN: 978-1-5224-9459-1

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Table of Contents

Preface **5**
Introduction 5
Document Layout 5
Conventions Used in this Guide 7
Recommended Reading 8
The Microchip Web Site 8
Product Change Notification Service 8
Customer Support 8
Document Revision History 9

Chapter 1. Product Overview

Chapter 2. Hardware Integration
2.1 I2C Bus Controller Windows Support 13
2.2 PAC194X/5X VDD, $\overline{\text{PWRDN}}$, SLOW/$\overline{\text{ALERT1}}$ and GPIO/$\overline{\text{ALERT2}}$ Pin Connections 13
2.3 Channel Sense Resistor (RSENSE) 15
2.4 Channel Polarity and Scale Range 15
2.5 PAC194X/5X-2 Connection Considerations 16

Chapter 3. ACPI BIOS Integration
3.1 PAC194X/5X ACPI Device 17

Chapter 4. Driver Installation
4.1 Prerequisites 23
4.2 Installation Instructions 24

Chapter 5. Device Instantiation and Initialization
5.1 Getting Started 25
5.2 Device Configuration 26

Chapter 6. PAC194X/5X Device Driver Interfaces
6.1 Introduction 29
6.2 Energy Meter Interface 29
6.3 Device Control Interface 31

Chapter 7. PAC194X/5X Driver Characteristics
7.1 Introduction 45
7.2 Multiple I/O Requests 45
7.3 Energy Meter Interfaces (EMI) vs. Device Control Interfaces 45
7.4 Software Accumulators 46
7.5 Low-Power Mode and Power States 46

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Appendix A. Sample ACPI ASL PAC194X/5X Device Definition
Appendix B. Testing the PAC194X/5X ACPI Integration
Appendix C. PAC194X/5X Device Control Interface
Appendix D. Glossary of Terms and Acronyms
Worldwide Sales and Service 83

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a "DS" number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is "DSXXXXXXXXA", where "XXXXXXXX" is the document number and "A" is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE online help. Select the Help menu, and then Topics, to open a list of available online help files.

INTRODUCTION

This chapter contains general information that will be useful to know before using the PAC194X/5X Microsoft® Windows® 10 and Windows 11 Device Drivers. Items discussed in this chapter include:

- Document Layout
- Conventions Used in this Guide
- Recommended Reading
- The Microchip Web Site
- Customer Support
- Document Revision History

DOCUMENT LAYOUT

This document describes how to integrate the PAC194X/5X devices in a Windows 10 or Windows 11 system, how to install the device driver and how to use the driver software interfaces for configuring the device and reporting the data samples. The manual layout is as follows:

- **Chapter 1. "Product Overview"** – This chapter presents the purpose of the PAC194X/5X Device Driver and a high level picture about the associated software ecosystem.
- **Chapter 2. "Hardware Integration"** – This chapter presents the key device driver assumptions about the PAC194X/5X Device Driver hardware integration into a Windows 10 or Windows 11 system.
- **Chapter 3. "ACPI BIOS Integration"** – This chapter presents the host system ACPI BIOS implementation details required for the PAC194X/5X Device Driver instantiation and configuration.
- **Chapter 4. "Driver Installation"** – This chapter presents the software dependencies and the steps to follow in order to install the PAC194X/5X Device Driver in the host Windows 10 or Windows 11 system.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

- **Chapter 5. "Device Instantiation and Initialization"** – This chapter presents the initial configuration steps performed by the PAC194X/5X Device Driver for each device, based on the BIOS configuration options and the initial system power state.
- **Chapter 6. "PAC194X/5X Device Driver Interfaces"** – This chapter provides the driver's software interfaces details needed by the user applications or other software modules to control the PAC194X/5X Device Driver configuration and to access the data samples.
- **Chapter 7. "PAC194X/5X Driver Characteristics"** – This chapter presents details about the driver features and operation, useful for the interoperation with other applications or software modules.
- **Appendix A. "Sample ACPI ASL PAC194X/5X Device Definition"** – This appendix contains the reference ACPI BIOS integration ASL code for one PAC194X/5X Device Driver.
- **Appendix B. "Testing the PAC194X/5X ACPI Integration"** – This appendix presents the software tools and the steps to follow in order to create, compile and test the ACPI BIOS ASL code changes needed for the PAC194X/5X devices.
- **Appendix C. "PAC194X/5X Device Control Interface"** – This appendix provides the detailed information about the PAC194X/5X Device Driver control interface, data structures, IOCTL codes and command parameters.
- **Appendix D. "Glossary of Terms and Acronyms"**

© 2021-2022 Microchip Technology Inc. and its subsidiaries

CONVENTIONS USED IN THIS GUIDE

This manual uses the following documentation conventions:

DOCUMENTATION CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	*MPLAB® IDE User's Guide*
	Emphasized text	...is the *only* compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	*File>Save*
Bold characters	A dialog button	Click **OK**
	A tab	Click the **Power** tab
N'Rnnnn	A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	4'b0010, 2'hF1
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier New font:		
Plain Courier New	Sample source code	`#define START`
	Filenames	`autoexec.bat`
	File paths	`c:\mcc18\h`
	Keywords	`_asm, _endasm, static`
	Command-line options	`-Opa+, -Opa-`
	Bit values	`0, 1`
	Constants	`0xFF, 'A'`
Italic Courier New	A variable argument	*`file`*`.o`, where *`file`* can be any valid filename
Square brackets [ ]	Optional arguments	`mcc18 [options]` *`file`* `[options]`
Curly brackets and pipe character: { | }	Choice of mutually exclusive arguments; an OR selection	`errorlevel {0|1}`
Ellipses...	Replaces repeated text	`var_name [, var_name...]`
	Represents code supplied by user	`void main (void) { ... }`

RECOMMENDED READING

This user's guide describes how to use the PAC194X/5X Microsoft Windows 10 and Windows 11 Device Drivers. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources:

- **PAC194X Data Sheet – *"Single/Multi-Channel Power Monitor with Accumulator, 9V Full-Scale Range"* (DS20006543)**
- **PAC195X Data Sheet – *"Single/Multi-Channel Power Monitor with Accumulator, 32V Full-Scale Range"* (DS220006539)**

THE MICROCHIP WEB SITE

Microchip provides online support via our web site at 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 (FAQs), 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

PRODUCT CHANGE NOTIFICATION SERVICE

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notifications 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 www.microchip.com, click on **Product 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.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

DOCUMENT REVISION HISTORY

Revision B (January 2022)

- Added the PAC194X/5X Device Driver support for Windows 11.

Revision A (May 2021)

- Initial Release of this Document.

NOTES:

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Chapter 1. Product Overview

This document presents the PAC194X/5X Microsoft Windows 10 or Windows 11 Driver technical aspects that users need to know in order to access the energy measurements reported by PAC194X/5X DC power monitor devices. Every time the document is mentioning PAC194X/5X it refers to all device family models, including the PAC194X/5X-2 low side models. When the document is mentioning PAC194X/5X-2 it specifically refers to the low-side device models only.

The document also describes the basic steps for integrating the PAC194X/5X device in a Microsoft Windows 10 or Windows 11 host system to enable the support of the PAC194X/5X device driver. As the PAC194X/5X device can be used in multiple ways and in different system configurations, there are some specific hardware and BIOS configuration details that need to be addressed before loading the Windows device driver.

Starting with Microsoft Windows 10 operating system, the energy consumption of various components from PC laptops or other portable devices can be monitored in order to identify the optimal configuration options that can maximize the battery life. The Energy Estimation Engine (E3) service in Microsoft Windows 10 or Windows 11 uses estimation algorithms or the real-time energy measured by metering ICs connected into various power rails of the system.

The device drivers of the metering ICs must report the energy measurements to Windows E3 module through a standard software interface, the Energy Meter Interface (EMI). Windows E3 service or the user applications can access the energy data by calling a standard set of EMI IOCTL operations.

The PAC194X/5X device driver creates in addition to the EMI interface a customized "Device Control Interface" and implements an extended set of IOCTL operations which provide access to much more data and device configuration options.

The application PAC194X/5X Energy Metering and Driver Interface Utility – EMI Utility uses all the IOCTL operations implemented by the PAC194X/5X device driver to demonstrate the device capabilities. The operator can also use it to change the device initial configuration. Some of the device initial configuration parameters are provisioned to the device driver by ACPI BIOS Device Specific Methods (DSM).

Figure 1-1 depicts the main software blocks and their interaction with the PAC194X/5X device driver.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

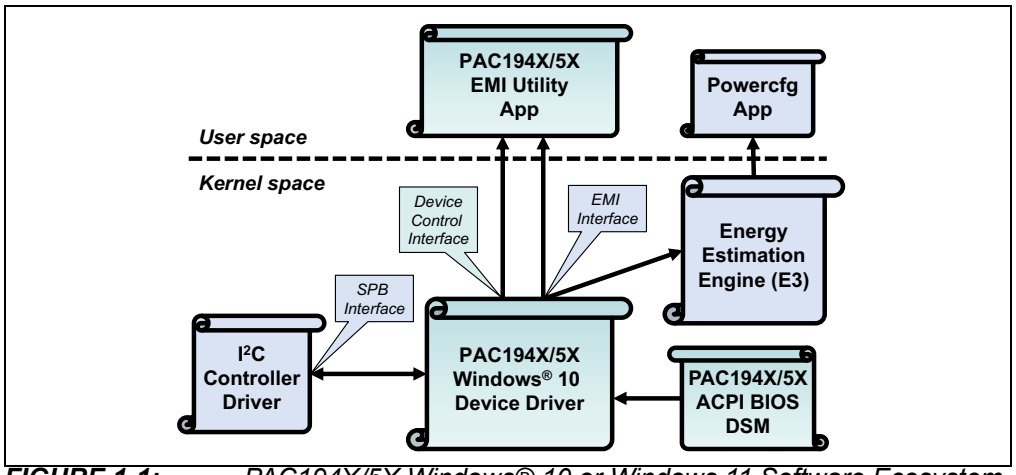

FIGURE 1-1: *PAC194X/5X Windows® 10 or Windows 11 Software Ecosystem.*

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Chapter 2. Hardware Integration

The hardware integration must first address all the electrical details specified in the device datasheet. (For example, the device SDA pin is open drain and requires a pull-up resistor to the I^2C bus Controller V_{DD}.)

The following notes refers only to the hardware details that need specific configuration to make them compatible with the Windows 10 or Windows 11 device driver:

- I2C Bus Controller Windows Support
- PAC194X/5X VDD, PWRDN, SLOW/ALERT1 and GPIO/ALERT2 Pin Connections
- Channel Sense Resistor (RSENSE)
- Channel Polarity and Scale Range
- PAC194X/5X-2 Connection Considerations

2.1 I^2C BUS CONTROLLER WINDOWS SUPPORT

The PAC194X/5X device I^2C/SMBus interface is, by default, configured in the I^2C mode. Keep in mind that the SMBus protocol is not currently supported by Windows; therefore, the system integrators must connect the PAC devices to the I^2C bus controllers for which a Windows device driver is available and the driver is compatible with the Windows Simple Peripheral Bus Framework Extension (SpbCx). This is a mandatory condition because the PAC194X/5X Windows device driver uses the SpbCx standard software interface in order to communicate with I^2C peripherals.

The I^2C bus controller may be located in the system CPU (SoC architecture), in the system Embedded Controller (EC) or may be another device connected to one of the system internal buses.

2.2 PAC194X/5X V_{DD}, $\overline{\text{PWRDN}}$, SLOW/$\overline{\text{ALERT1}}$ AND GPIO/$\overline{\text{ALERT2}}$ PIN CONNECTIONS

V_{DD} and $\overline{\text{PWRDN}}$ pins – connected to "always on" power rail

The usual PAC194X/5X system integration scenario assumes that the system power rails are also monitored during the "standby" or "hibernate" operating system power states. Therefore, the PAC device V_{DD} and $\overline{\text{PWRDN}}$ pins should be connected to a system power rail that is not shut down by the system standby or hibernate power sequence. This way, the PAC device continues to be capable of accumulating the data about the system energy consumption during the standby/hibernate state, if desired. The PAC194X/5X Windows device driver can retrieve and report these data to the Windows Energy Estimation Engine (E3) or to the user applications.

SLOW/$\overline{\text{ALERT1}}$ pin – configured as SLOW and controlled by hardware

The PAC194X/5X Windows device driver ensures that SLOW/$\overline{\text{ALERT1}}$ pin is always configured as "SLOW" signal and the pin transitions trigger the accumulator registers "limited refresh", latching the accumulated data (these are the PAC device default settings.)

© 2021-2022 Microchip Technology Inc. and its subsidiaries

The data sample rate should be reduced to a minimum in standby/hibernate states, in order to reduce the chances for accumulator register overflow and to facilitate a longer data accumulation period in these states. The reduced sample rate also reduces the energy used by PAC194X/5X device. In the normal system operation, the sample rate control is done by the PAC194X/5X device driver that reconfigures the SAMPLE_MODE bits from the PAC194X/5X CTRL register just before the operating system entering the standby, hibernate or power down state.

In the case of an operating system crash, eventually the system is forcibly shutdown by the user or by some automated safety mechanism, but the PAC194X/5X device driver is not functional. If the power is not removed from the PAC V_{DD} pin, the device can still be reconfigured in slow sampling mode if the SLOW/$\overline{\text{ALERT1}}$ pin is driven high by hardware. For example, the SLOW/$\overline{\text{ALERT1}}$ pin can be driven high by a hardware block that detects the state of the CPU or the main memory power rails. Alternatively, the SLOW/$\overline{\text{ALERT1}}$ pin may be driven by the system Embedded Controller (EC). The EC is usually an always on device, which performs various hardware low-level house-keeping and controls the system power rails and the power-on sequence.

When the system is turned back on, the PAC194X/5X Windows device driver can detect that SLOW/$\overline{\text{ALERT1}}$ pin was asserted and collects the data accumulated and latched by the pin de-assertion event.

Figure 2-1 is a simplified view of the recommended PAC194X/5X hardware integration with the host system. The GPIO/$\overline{\text{ALERT2}}$ signal is not included since it is used optionally.

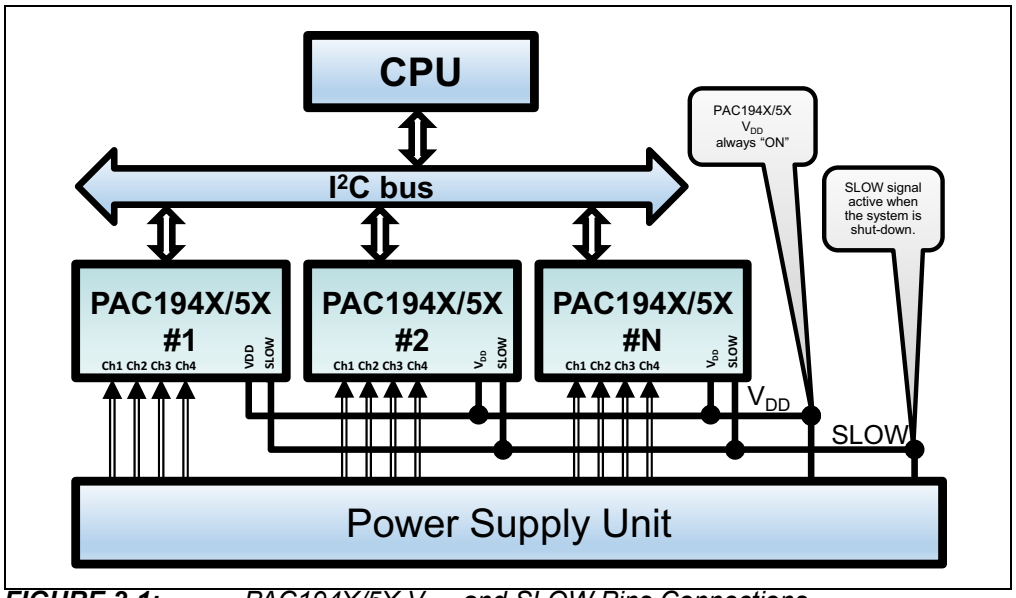

FIGURE 2-1: *PAC194X/5X V_{DD} and SLOW Pins Connections.*

GPIO/$\overline{\text{ALERT2}}$ pin – optional use

There is no design assumption implemented in the PAC194X/5X device driver about the use of this pin. Therefore, the system designer may need to consider using a software application for pin configuration and control or may choose to not use it at all.

The default configuration of the GPIO/$\overline{\text{ALERT2}}$ pin is digital input. Alternatively, the system designer may consider using this pin as digital signal output. The PAC194X/5X device driver software interface provides functions for the following pin configuration options:

- GPIO input – the user application uses the driver software interface to read the digital input value
- GPIO output – the user application uses the driver software interface to control

© 2021-2022 Microchip Technology Inc. and its subsidiaries.

the digital output value

- ALERT signal – the user application uses the driver software interface to select the alert types (e.g. over/under current/voltage, over power) and the threshold values.

The *PAC194X/5X Windows Energy Metering and Driver Interface Utility* created by Microchip is one example of a software application that uses the PAC194X/5X device driver functions to provide user control over GPIO/$\overline{\text{ALERT2}}$ pin, in a graphical, intuitive interface.

2.3 CHANNEL SENSE RESISTOR (R_{SENSE})

PAC194X/5X Windows device driver supports sense resistor values that are integer values when expressed in microohm units. For example, a value of 1500 microohms is supported.

The best measurement precision can be achieved by using four-terminal current sense resistors with built-in Kelvin connections for the sense lines and resistance tolerance of 1% or better.

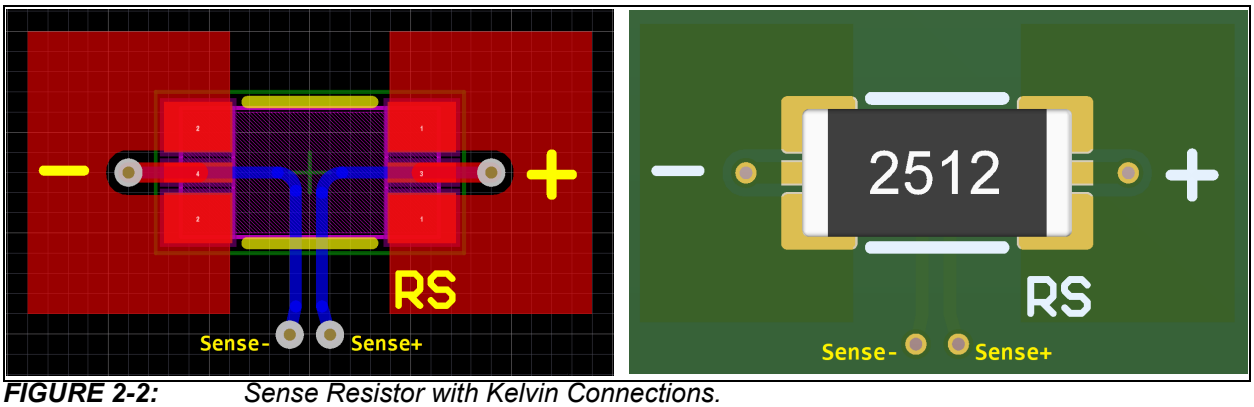

FIGURE 2-2: *Sense Resistor with Kelvin Connections.*

2.4 CHANNEL POLARITY AND SCALE RANGE

After the Power-on Reset (POR), the PAC device default channel configuration is set for unipolar voltage measurements (both V_{SENSE} and V_{BUS}) and the full-scale range FSR values are set to the maximum voltage values supported by device (V_{SENSE} full-scale is 100 mV and V_{BUS} full-scale is 32V on PAC195X or 9V on PAC194X). The alternative configuration option (FSR/2) for the full-scale values reduces them by half. The Windows device driver can change this configuration during the device initialization phase, as indicated by the ACPI BIOS Device Specific Method (DSM) parameters. In addition, the user applications may call dedicated driver functions to change the channel polarity and scale settings.

In bipolar, configuration the PAC194X/5X device can report also the negative values measured for V_{SENSE} and/or V_{BUS}. However, it is recommended to connect the PAC device SENSE+/- pins to the power rails such that the measured voltage values are reported as positive values during the normal system operation. There are two reasons for this:

- Windows E3 service supports only positive energy values and is not able by design to control the channel polarity,
- Windows E3 service reads the energy data from the standard Energy Meter Interfaces (EMI) created by the PAC194X/5X device driver for the monitored rails. By definition, the EMI interface allows only positive integer values.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

The PAC194X/5X device driver can also report the "signed" energy values, through a different Microchip-defined software interface. User applications may use the alternative interface to collect the "signed" energy values.

Please note that the bipolar configuration reduces by half the least significant bit value (the precision) of the measured voltage parameter (V_{SENSE} or V_{BUS}). To avoid the precision loss, the user may consider using the alternative FSR/2 configuration option. This option is valid only if the electrical design of monitored power rail cannot induce V_{SENSE} or V_{BUS} maximum voltage values higher than the alternative FSR/2 range value.

2.5 PAC194X/5X-2 CONNECTION CONSIDERATIONS

PAC194X/5X-2 devices have two additional pins for each channel, V_{BUS+} and V_{BUS-}.

The device driver implementation assumptions used for computing the values for energy, power, rail voltage, current and other reported values, are:

- The SENSE+/- pins are connected to the sense resistor terminals, in the Kelvin-connection configuration.
- V_{BUS-} pin is connected to the ground.
- V_{BUS+} pin connection options are:
  - V_{BUS+} connected to the monitored power rail if the rail voltage cannot exceed the channel V_{BUS} full-scale range.
  - V_{BUS+} connected to a power rail voltage divider (for example a resistive voltage divider) if the rail voltage is higher than channel V_{BUS} full-scale range.
- The ratio between the power rail voltage and the voltage measured as V_{BUS} is a constant factor used by the device driver to compute the real power rail voltage, energy, and power. During the device initialization phase the device driver acquires the V_{BUS} multiplication factor from a dedicated ACPI BIOS DSM function.

© 2021-2022 Microchip Technology Inc. and its subsidiaries.

Chapter 3. ACPI BIOS Integration

Over and above any integration guidance provided in this document, OEMs and system integrators are responsible for creating a valid ACPI BIOS that accurately reflects their hardware platform. This includes the details of how the PAC194X/5X devices are configured and connected in the system.

The guidance provided in this chapter is limited to describing the details that are necessary to integrate the PAC194X/5X devices into a system running the Windows 10 or Windows 11 operating systems and the PAC194X/5X device driver.

Appendix A. "Sample ACPI ASL PAC194X/5X Device Definition" of this document contains an example ACPI device definition, written in ACPI Source Language (ASL). The example illustrates how a PAC194X/5X needs to be integrated into the ACPI BIOS. Please refer to Appendix A example when reading this section.

Appendix B. "Testing the PAC194X/5X ACPI Integration" of this document provides an example of easily available tools and the process to follow in order to quickly test the PAC194X/5X integration into the system, without replacing the BIOS firmware in the system Flash.

3.1 PAC194X/5X ACPI DEVICE

There are several per-device and per-channel initial configuration options included in the system's ACPI BIOS for each PAC194X/5X device that needs to be accessible to the PAC194X/5X Windows device driver:

- The device I^2C address.
- The name of the power rail to which a device channel is connected.
- The value of the sense resistor connected to each device channel (R_{SENSE}).
- EMI enable/disable flags, indicating for which channels the device driver must create the Energy Meter Interface (EMI). The channels without EMI interface may be referred as "private channels".
- Polarity and scale range settings for each channel.
- The device Active mode (full speed) and Idle mode (low power) data sampling frequencies.
- The device driver "Refresh watchdog" timer interval duration.
- The V_{BUS} multiplication factor for each channel.

For each PAC194X/5X device, the system's ACPI BIOS provides a DSM function that device drivers can call to retrieve the configuration option values. The PAC194X/5X device driver uses the DSM functions for the device initial configuration. Some of the configuration options can also be changed during the system runtime by the user applications, using special driver functions. But, a system reboot reverts the PAC device configuration to the ACPI BIOS initialization options.

The OEM/system integrator is responsible for creating a proper ACPI BIOS device specific configuration for each PAC194x/5x device integrated in the system.

Appendix A. "Sample ACPI ASL PAC194X/5X Device Definition" contains an example of ACPI ASL code for one PAC194X/5X device. This example may be copied, multiplied and adapted for the customer system.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

As part of the PAC194X/5X device definition, a DSM control method that corresponds to the Universal Unique Identifier (UUID), **721F1534-5D27-4B60-9DF4-41A3C4B7-DA3A** must be included, as shown in the example. This UUID is reserved to Microchip PAC194X/5X devices and must not be changed. The PAC194X/5X device driver relies on the DSM control method identified by this UUID.

There are 8 functions (0 to 7) implemented by the current version of the DSM (Revision 0).

DSM Function 0

As required by ACPI, the DSM Function 0 returns an integer value representing a bit-mask which indicates the functions supported by this DSM. For example, a value of 1 on the second least significant bit, bit 1, indicates that the Function 1 is supported. Hence, the returned bit-mask value is `0xFF` for the DSM revision 0.

DSM Function 1

DSM Function 1 returns a set of four strings that provides for each device channel the name of the power rail to which that channel is connected. The first string is associated to channel 1. The PAC194X/5X accepts any arbitrary name strings. However, for the channels that report energy data to the Windows E3 module (through EMI interfaces) the supplied power rail names cannot be arbitrary. E3 needs to associate the power rails data with the system's main hardware components. Therefore, the power rail names must be formed using predefined string tokens, according to the Windows-defined EMI power rail taxonomy rules. Please refer to Microsoft supplied documentation for the detailed description of the power rail naming rules.

The PAC194X/5X driver also accepts empty strings for the power rail names. The device driver also performs all the data acquisition and processing for the unnamed channels but does not create the standard EMI interfaces for them, so the energy data is not accessible to the Windows E3 module. Such channels are referred to as "private channels". The private channels data can be queried by the user applications using the PAC194X/5X driver's device control interface IOCTL functions.

DSM Function 2

DSM Function 2 returns a buffer of four positive integer values representing the channel sense resistor (R_{SENSE}) values expressed in microohm units. The first value is associated to channel 1.

Zero value can be assigned to an R_{SENSE} channel to indicate that the channel is not used (not connected to a power rail) or can be ignored. The PAC194X/5X device driver turns the channel OFF in the device control register and performs no data processing for the unused channel (this reduces the PAC194X/5X device own power consumption and the system CPU processing load).

OEMs and systems integrators must take great care in properly configuring the names and the R_{SENSE} values for the power rails because both the Windows E3 and the PAC194X/5X device driver rely on these values being correct. There is no way for the device driver to know if a given power rail name is valid, or, more importantly, to know if the indicated channel is connected to the specified power rail.

DSM Function 3

DSM Function 3 returns an integer value representing a bit-mask which indicates for what channels the PAC194X/5X device driver may create EMI interfaces. The Most Significant bit (MSb) of the bit-mask is representing channel 1. The device driver combines this information with the information returned by DSM Function 1 to determine if the EMI interface can be created for a certain channel:

- EMI interface requires a non-empty string assigned for the power rail name, otherwise the channel is kept private.
- If a mask bit value is cleared, the associated channel is kept "private", the EMI

© 2021-2022 Microchip Technology Inc. and its subsidiaries

interface is not created, even if there is a non-empty string name assigned to the channel.

DSM Function 4

DSM Function 4 returns a buffer of eight integer values indicating for each channel the V_{SENSE} and V_{BUS} polarity and the full-scale range configuration:

- The first four values are the configuration codes for V_{SENSE}, starting with channel 1.
- The last four values are the configuration codes for V_{BUS}, starting with channel 1.
- The only accepted values for the configuration codes are:
  - `0x0` = unipolar
  - `0x1` = bipolar
  - `0x2` = bipolar, half-scale range

WARNING

Any invalid configuration code value triggers a device driver configuration error. The driver stops the device configuration and reports the error to the Windows® device manager module. Consequently, the operating system will consider the device as non-functional and inaccessible.

DSM Function 5

DSM Function 5 returns a buffer of two integer values indicating the PAC194X/5X sample frequencies to be configured when the device enters the Active mode, respectively when the device enters the Idle mode. The only accepted values are 1024, 256, 64, 8 sps.

WARNING

An invalid sample frequency value triggers the device driver configuration error. The driver stops the device configuration and reports the error to the Windows® device manager module. Consequently, the operating system will consider the device as non-functional and inaccessible.

DSM Function 6

DSM Function 6 returns an integer value expressed in seconds, representing the maximum time interval that the device driver may allow between consecutive PAC194X/5X device REFRESH commands while the system is running, to avoid the device accumulator saturation. If the provided value does not belong to the 60~60000 interval, the device driver ignores it and uses an internally hardcoded value (240).

DSM Function 7

DSM Function 7 returns a buffer of four positive integer values representing the V_{BUS} multiplication factors (the ratio between the power rail voltage and the voltage measured as V_{BUS}). The first value is associated to the channel 1.

The multiplication factor (K) must be expressed as shown in Equation 3-1.

EQUATION 3-1:

$$K = 1000 \times \frac{V_{RAIL}}{V_{BUS}}$$

Please note that the device driver uses the DSM Function 7 only if the device is identified as a low-side type (PAC194X/5X-2), based on the device PRODUCT ID register value.

TABLE 3-1: PAC194X/5X ACPI CONFIGURATION PARAMETERS SUMMARY

Parameter	Type	Valid Values	Comments
UID	Integer	≥1	Each PAC device in the system must have a unique UID.
I^2C 7-bit Device Address	Integer	See the "Address Select Resistor" table in the device datasheet.	• There are 16 possible values, selected by ADDRSEL pin voltage level. • The table provides the supported I^2C address values along with the value of the resistor to be connected to ADDRSEL pin.
I^2C Connection Speed	Integer	Typical: 400000 Maximum: 3400000	• The connection speed in Hz. • The speed must match the I^2C bus controller supported speed.
I^2C Controller Name	String	The ACPI name of the I^2C bus controller.	The I^2C bus controller driver must have Windows® SpbCx compliant driver.
Channel Names (DSM Function 1)	String (set of 4)	"" Or non-empty string	• Configuration set layout: {STR_CH1, STR_CH2, STR_CH3, STR_CH4} • STR_CHn: name string for channel 'n'. It should comply with Microsoft® power rail / EMI naming taxonomy. • STR_CHn = "": channel 'n' has no EMI interface (private channel).
R_{SENSE} Values (DSM Function 2)	Integer (set of 4)	≥0	• Configuration set layout: {R_{SENSE1}, R_{SENSE2}, R_{SENSE3}, R_{SENSE4}} • R_{SENSEn}: sense resistor value for channel 'n', expressed in microohms • R_{SENSEn} = 0: channel 'n' is not used
EMI Enable/Disable Bit-Mask (DSM Function 3)	Bit-mask ($b_3b_2b_1b_0$)	0x0 ~ 0xF	• Bit-mask layout: b_{CH1}:b_{CH2}:b_{CH3}:b_{CH4} • Bit values: - b_{CHn} = 1: EMI enabled for channel 'n' - b_{CHn} = 0: EMI disabled for channel 'n' (private channel)
V_{BUS}/V_{SENSE} Polarities and Full Scale Ranges (DSM Function 4)	Integer (set of 8)	0x0, 0x1, 0x2	• Configuration set layout: {CFG_VS1, CFG_VS2, CFG_VS3, CFG_VS4, CFG_VB1, CFG_VB2, CFG_VB3, CFG_VB4} • CFG_VSn: V_{SENSE} configuration for channel 'n' • CFG_VBn: V_{BUS} configuration for channel 'n' • Configuration values (CFG_VSn/CFG_VBn): - `0x0` = unipolar - `0x1` = bipolar - `0x2` = half scale bipolar

© 2021-2022 Microchip Technology Inc. and its subsidiaries

TABLE 3-1: PAC194X/5X ACPI CONFIGURATION PARAMETERS SUMMARY

Parameter	Type	Valid Values	Comments
Sampling Frequencies for Active and Idle modes (DSM Function 5)	Integer (set of 2)	{1024, 256, 64, 8}	• Configuration set layout: {SPS_{ACTIVE}, SPS_{IDLE}} • SPS_{ACTIVE}: The sampling frequency the PAC device must use when the system runs in a high-power “active” state. • SPS_{IDLE}: The sampling frequency the PAC device must use when the system runs in a low-power Idle mode (e.g. sleep, hibernate or modern-standby)
REFRESH Watchdog Timer Interval (DSM Function 6)	Integer	60 ~ 60000	• The maximum time interval that the device driver may allow between consecutive PAC194X/5X device REFRESH commands while the system is “active”. • Expressed in seconds. • The driver ignores the invalid values and uses an internally hardcoded value (240).
V_{BUS} Multiplication Factors (DSM Function 7)	Integer (set of 4)	≥1000	• Configuration set layout: {K_{CH1}, K_{CH2}, K_{CH3}, K_{CH4}} • K_{CHn}: the ratio between the power rail voltage and the voltage measured as V_{BUS} for channel 'n', multiplied by 1000 - $K_{CHn} = 1000 * (V_{RAILn}/V_{BUSn})$ • The driver ignores the configuration set if the device is not a low-side one (PAC194X/5X-2).

NOTES:

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Chapter 4. Driver Installation

4.1 PREREQUISITES

The following driver dependencies must be resolved before installing the driver:

- The PAC194X/5X devices are integrated in the host system.
- The I^2C bus controller device driver is SPB Framework Extension (SpbCx) compliant.

4.1.1 PAC194X/5X Integration in the Host System

4.1.1.1 ACPI ASL

Refer to the indications listed in **Chapter 3. "ACPI BIOS Integration"** and make the necessary ACPI ASL changes to ensure that PAC194X/5X devices are enumerated by the Microsoft Windows Device Manager.

4.1.1.2 V_{DD} AND SLOW PINS

The device V_{DD} pin should be connected to a system power rail that is not shut down by the system standby or power-down sequence.

The SLOW pin should be connected to a signal that is driven high when the system power is turned off.

CAUTION
Do not use the SLOW pin to change the device sampling rate during the normal system operation. The PAC194X/5X device driver does not monitor the SLOW pin transitions while the system is running. Therefore, if the user application reconfigures the device into a non-adaptive sampling mode, any subsequent SLOW pin state activation would cause incorrect data processing in the driver.

The PAC194X/5X device driver supports the following SLOW pin use cases:

- A fail-safe mechanism that ensures that PAC devices are switched into the Slow Sampling mode if the system crashes and is forcibly turned off by the operator or by other fail-safe mechanism
- When the system power is turned back on, the SLOW pin transition high-to-low saves into the device readable registers the data accumulated while the system was off. The device driver can report the saved data to the user application for further processing and analysis.

4.1.2 SpbCx Compliant I^2C Bus Controller Device Driver

To access the I^2C bus and communicate with the PAC devices, the PAC194X/5X device driver uses the standard interface and protocol implemented by the Microsoft Windows SpbCx framework. This implies that the I^2C bus controller must also have a Microsoft Windows device driver compliant with the SpbCx framework. Usually, such driver is created by the I^2C controller manufacturer and provided through Windows installation kit or through Windows update.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

4.2 INSTALLATION INSTRUCTIONS

There are several options to install the PAC194X/5X Microsoft Windows device driver:

- Install via Microsoft Windows update.
- Install using the Device Manager and the driver package downloaded from the Microchip website.
- Use the DevCon utility and the driver package downloaded from the Microchip website.

4.2.1 Install Using Microsoft Windows Update or Device Manager

For details about the driver installation through Microsoft Windows update or Device Manager, refer to the following Microsoft article: "How to: Install and Update drivers in Windows 10".

4.2.2 Driver install using DevCon Utility

The DevCon utility is provided with the Windows Driver Kit (WDK).

1. Depending on the target system OS (32-bit or 64-bit), extract the following files from the driver package:

- `PAC194x_5x.sys`
- `PAC194x_5x.inf`
- `PAC194x_5x.cat`

2. Copy the driver files and the DevCon utility on the target system.
3. Open in an elevated command prompt, change directory to the folder where the driver files have been copied and execute the command:

```
devcon.exe update pac194x_5x.inf ACPI\MCHP1940
```

4.2.3 Verify the Installation

To verify that the PAC194X/5X device driver installation was successfully completed, open the Device Manager and check if the PAC194X/5X devices are listed under the "Microchip Energy Metering Devices" category and they are reported as working properly. See Figure 4-1 for more details.

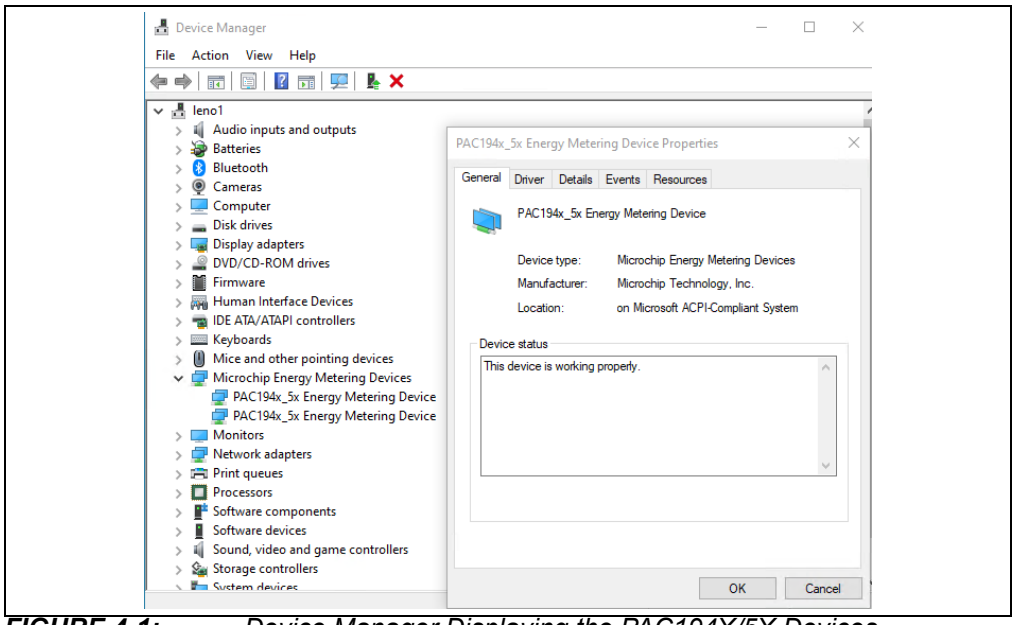

FIGURE 4-1: *Device Manager Displaying the PAC194X/5X Devices.*

© 2021-2022 Microchip Technology Inc. and its subsidiaries.

Chapter 5. Device Instantiation and Initialization

5.1 GETTING STARTED

During the Windows operating system startup, the PAC194X/5X devices connected to the system's I^2C buses, are enumerated by the Windows ACPI driver (`acpi.sys`) which plays the role of the bus driver. A PAC194X/5X device instance path is `ACPI\MCHP1940\<UID>`, where UID is the device's unique ID value assigned in the ACPI ASL code and `MCHP1940` is the vendor hardware ID assigned for the PAC194X/5X device family. The ACPI ASL code details specific for the PAC194X/5X are described in **Chapter 3. "ACPI BIOS Integration"**.

The PAC194X/5X device driver is a Kernel-Mode Driver Framework (KMDF) function driver which is instantiated by the Windows PnP Manager whenever the ACPI driver enumerates the devices with the Hardware ID `ACPI\MCHP1940`.

In the device initialization stage, the PAC194X/5X device driver invokes the device's ACPI BIOS Device Specific Method (DSM) which is identified by the Universal Unique ID (UUID) `721F1534-5D27-4B60-9DF4-41A3C4B7DA3A`. The information returned by device's DSM is used to setup the initial configuration for a subset of device features which affect the data processing algorithms implemented by the driver:

- Channel name – the name of the power rail monitored by the channel:
  - The driver creates EMI device interfaces for all the named channels. For details, refer to the following Microsoft article: Energy Meter Interface. The Windows E3 service can use the data exported by the EMI interfaces if their names are compatible with Windows EMI power rail taxonomy.
  - If the channel name is an empty string, the driver does not create an EMI. The data accumulated in the device for that channel is still collected, processed and reported through the PAC194X/5X driver's device control interface created for each PAC194X/5X device (the channel is referred to as private channel).
- R_{SENSE} values – the value of the sense resistor for each channel, expressed in microohms.
  - The resistor value is used for the energy calculations and must be an integer, positive value.
  - The R_{SENSE} = 0 indicates a channel which is not connected to a power rail. No EMI interface is created, and the driver reports no data for this channel.
- EMI enable/disable bit mask – indicates for which named channels the driver will create an EMI interface.
- V_{SENSE} polarity, V_{BUS} polarity and full-scale range configuration for each channel.
  - If any of the configuration values is invalid the driver reports the `STATUS_ACPI_INVALID_DATA` error to the kernel and stops the device initialization process.
- The sampling frequencies to be configured when the device enters the Active mode, respectively when the device enters the Idle mode.
  - If any of the configuration values are invalid the driver reports the `STATUS_ACPI_INVALID_DATA` error to the kernel and stops the device initialization process.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

- REFRESH watchdog timer interval – the maximum time interval the driver allows between consecutive device REFRESH commands when the system power state is Working (S0).
  - The driver automatically sends a REFRESH command to the device, resets the timer, and collects all the device data when the timer expires. The timer is also reset when the driver executes a REFRESH command to resolve a data request or a device configuration change request received from Windows E3 service or from user applications.
  - If the configuration value is invalid, the driver ignores it and uses the internally hardcoded value of 240 seconds.
- V_{BUS} multiplication factors for each channel of the PAC194X/5X-2 devices.
  - The factor value is multiplied with the measured V_{BUS} value to compute the rail voltage only if the device is a PAC194X/5X-2 type, otherwise the value is ignored.
  - The device driver computes the rail voltage assuming that the multiplication factor value is predetermined by the system designer as $1000 * (V_{RAIL}/V_{BUS})$.

The driver uses the information returned by device's ACPI BIOS DSM in the following situations:

- The operating system was powered-up from a full shutdown state: Soft Off – S5 or Mechanical Off – G3 (Microsoft article System Power States describes the system power states from the Windows perspective).
- The PAC194X/5X device lost the power while the operating system transitioned through a low-power state (the PAC device POR has been detected).
- The operating system was Restarted.

There are also PAC194X/5X features for which the initial configuration is not affected by device's ACPI BIOS DSM:

- The configuration option is hardcoded in the driver, or
- The device driver preserves the feature configuration as is (it may be the default configuration option set by the last device POR or it may be the configuration selected by the user in the previous operating system running session).

Section 5.2 "Device Configuration" provides details about the initial configuration and about the runtime change options for the device features.

5.2 DEVICE CONFIGURATION

Summarized below is the list of PAC194X/5X device features controlled by the Windows device driver. The list indicates the initial configuration and the configuration options for the user.

- I^2C/SMBus™ interface is configured as I^2C
  - Hardcoded configuration. SMBus option is not supported currently by Windows SPB driver framework.
  - No user runtime control supported; the driver has exclusive control.
- SAMPLE_MODE – the data sampling mode and frequency
  - Initial configuration: adaptive accumulation mode with sample frequency values provisioned by ACPI BIOS DSM for both the Active and the Idle device running states.
  - The user has runtime control through device driver's control interface.
- SLOW/$\overline{\text{ALERT1}}$ pin is configured as SLOW
  - Hardcoded configuration. **Chapter 2. "Hardware Integration"** presents the SLOW pin connection options.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

  - No user runtime control supported; the driver has exclusive control.
- Limited REFRESH_V command triggered on Slow signal transition (latch the data into the readable registers).
  - Hardcoded configuration.
  - No user runtime control supported; the driver has exclusive control.
- GPIO/$\overline{\text{ALERT2}}$ pin configuration
  - Initial configuration: general-purpose input pin, hardcoded in the driver.
  - The user has runtime control through device driver's control interface:
    - Read the pin input value.
    - Change the pin configuration to general-purpose output and control the output value.
    - Change the pin configuration to ALERT pin and select which alert type can activate the pin.
- Channel ON/OFF configuration
  - Initial configuration: all the channels are enabled (ON) except for the channels with 0Ω sense resistors, as indicated by the sense resistor values provisioned by ACPI BIOS DSM.
  - The user has runtime control through device driver's control interface.
    - Only the channel with no EMI interface can be disabled.
    - The driver creates the EMI interface for the channel based on the ACPI BIOS DSM provisioned information: the channel name and the EMI Enable/Disable bitmask.
- Channel V_{SENSE} polarity, V_{BUS} polarity and full-scale range (FSR) configuration
  - Initial configuration: according to the values provisioned by the ACPI BIOS DSM.
  - The user has runtime control through device driver's control interface.
- Accumulators configuration
  - Initial configuration: Power accumulation, hardcoded in the driver.
  - The user has runtime control through device driver's control interface. There are two alternative options: V_{SENSE} accumulation and V_{BUS} accumulation.
- Limit registers threshold values
  - Initial configuration: the device driver preserves the configuration as is. It may be the default configuration from the last device POR, or the last user selected configuration change.
    - POR threshold for accumulator fullness: 15/16 full.
    - POR threshold for Over Current (OC), Under Current (UC), Over Power (OP), Over Voltage (OV), Under Voltage (UV): 0.
    - POR threshold for the number of consecutive, limit exceeding samples required to trigger the limit alert: 1.
  - The user has runtime control through device driver's control interface.
- ALERT source enable/disable configuration
  - Initial configuration: all device alert sources are disabled. The configuration is hardcoded in the driver.
  - The user has runtime control through device driver's control interface. The user can enable or disable any alert source: conversion cycle complete, accumulator fullness, OC, UC, OP, OV, UV.

NOTES:

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Chapter 6. PAC194X/5X Device Driver Interfaces

6.1 INTRODUCTION

A device interface contains the device's symbolic link name which the user applications can use to direct the Input/Output (I/O) requests into the device. When the driver creates the interface, it also associates an interface class GUID (the interface type identifier).

The user application can use the `SetupDiEnumDeviceInterfaces` function call to discover the interfaces belonging to a certain GUID class, and then use the `SetupDiGetDeviceInterfaceDetail` function call to obtain the symbolic link names of the devices. (The two `SetupDi…` functions mentioned in this paragraph are implemented by `setupapi.lib` library).

Next, the user application can pass the device symbolic link names to the `CreateFile` kernel function to open the device for I/O and receive the assigned file handle. To request the device to perform an I/O operation, the user application can invoke the `DeviceIoControl` kernel function, specifying the device file handle, the operation IOCTL code (the input/output control code which identifies the requested operation) and other operation specific parameters. Then, the kernel executes the device driver function which translates the IOCTL request into device specific commands, handles the device communication, processes the reported data and returns the results back to the operation requester.

This chapter presents the class GUID and the IOCTL codes implemented by the PAC194X/5X device driver for each type of interface it creates:

- Energy Meter Interface
- Device Control Interface

6.2 ENERGY METER INTERFACE

The driver creates an EMI interface for each device channel provided that:

- the power rail monitored by the channel has a non-empty name,
- the EMI enable/disable bit-mask allows it.

For details regarding the EMI interface, refer to the Microsoft article Energy Meter Interface and the file `emi.h` distributed with the Microsoft Windows Driver Kit.

The GUID for the EMI device interface class is:

- `45BD8344-7ED6-49cf-A440-C276C933B053`

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Table 6-1 lists the IOCTL control codes for the device I/O operations implemented by EMI.

TABLE 6-1: EMI IOCTLS

IOCTL name	Description
IOCTL_EMI_GET_MEASUREMENT	Returns the current energy measurement and the time at which the measurement was taken. - Input: None - Output: EMI_MEASUREMENT_DATA
IOCTL_EMI_GET_METADATA	Returns the EMI metadata from a device. - Input: None - Output: EMI_METADATA
IOCTL_EMI_GET_METADATA_SIZE	Returns the size of the EMI metadata object. - Input: None - Output: EMI_METADATA_SIZE
IOCTL_EMI_GET_VERSION	Retrieves the EMI version supported by the device driver. - Input: None - Output: EMI_VERSION

Table 6-2 describes the EMI I/O data structures and enumerations.

TABLE 6-2: EMI I/O DATA STRUCTURES AND ENUMERATIONS

Topic	Type	Description
EMI_MEASUREMENT_DATA	Output data structure	Reported by IOCTL_EMI_GET_MEASUREMENT, provides the energy measurement and the time when the measurement was taken. - Energy unit: 1 pWh - Timestamp unit: 100 ns - Energy and timestamp are 64-bit unsigned integer values.
EMI_MEASUREMENT_UNIT	Enumeration	Represents the available set of measurement units for the energy reported by IOCTL_EMI_GET_MEASUREMENT - EMI_VERSION_V1 implements only the pWh unit.
EMI_METADATA	Output data structure	Reported by IOCTL_EMI_GET_METADATA, provides information about: - The metering device hardware model and revision. - The metered power rail name.
EMI_METADATA_SIZE	Output data structure	Reported by IOCTL_EMI_GET_METADATA_SIZE, provides the size of the EMI_METADATA structure.
EMI_VERSION	Output data structure	Reported by IOCTL_EMI_GET_VERSION, provides the EMI version supported by this device driver. - PAC194X/5X device driver implements EMI_VERSION_V1 specification.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

6.3 DEVICE CONTROL INTERFACE

The PAC194X/5X device driver creates one control interface for each enumerated PAC194X/5X device. Compared to the EMI interface, the device control interface accepts the `IOCTL_PAC194X5X_*` requests, and provides users with access to much more data and device configuration options:

- All the measured values (V_{BUS},V_{SENSE} and many more in addition to the accumulated energy)
- Data for all the channels, including the channels with no EMI interface
- Device-level and channel-level configuration options (sampling mode, channel polarity, channel ON/OFF, alert limits)

NOTICE

Some device or channel configuration options are blocked by the driver either temporarily or permanently.

- The I^2C/SMBus communication options, the SLOW/$\overline{\text{ALERT1}}$ pin configuration options and the limited REFRESH configuration options are not accessible for the user.
- The channel ON/OFF control and the channel accumulator configuration options are allowed only if the channel has no EMI interface.

The GUID for the Device Control interface class is:

- `2DF42D48-4C09-453A-A05F-8A48F8FBB415`

The detailed C definition of the Device Control interface (IOCTL codes, data structures and data types) is listed in **Appendix C. "PAC194X/5X Device Control Interface"**. Table 6-3 and Table 6-4 provide a summary description of the IOCTL operations and their input/output data structures

Table 6-3 lists the IOCTL control codes for the device I/O operations implemented by Device Control interface.

TABLE 6-3: DEVICE CONTROL IOCTLs

IOCTL Name	Description
`IOCTL_PAC194X5X_GET_INTERFACE_REVISION`	Returns the device control interface revision number. New driver versions may implement new I/O operations or change the behavior of the existing ones. This IOCTL can help the user application determine which control interface IOCTLs are implemented by the PAC194X/5X driver currently installed in the host system. - Input buffer: none - Output buffer: `PAC194X5X_GET_INTF_REV` **Note:** - The current interface revision number is `0x01`
`IOCTL_PAC194X5X_GET_DEVICE_INFO_SIZE`	Returns the size allocation requirements for the buffer that must hold the device info data structure returned by `IOCTL_PAC194X5X_GET_DEVICE_INFO` operation. - Input buffer: none - Output buffer: `PAC194X5X_GET_DEVICE_INFO_SIZE`

TABLE 6-3: DEVICE CONTROL IOCTLs (CONTINUED)

IOCTL Name	Description
IOCTL_PAC194X5X_GET_DEVICE_INFO	Returns information about the device version and about the channel connections (rail names, sense resistors, connection status). - Input buffer: none - Output buffer: PAC194X5X_GET_DEVICE_INFO
IOCTL_PAC194X5X_GET_RSENSE_MICRO	Returns the sense resistor values for all channels as microohm units. A value of zero is returned for the unconnected sense resistors and for the channels which are not available on the devices with less than four channels. - Input buffer: none - Output buffer: PAC194X5X_GET_RSENSE_MICRO
IOCTL_PAC194X5X_DATA_REFRESH	Performs the update of the device driver software accumulators using the REFRESH_V device command. - Input buffer: none - Output buffer: none (zero bytes returned)
IOCTL_PAC194X5X_DATA_REFRESH_RESET	Performs the update of the device driver software accumulators using the REFRESH device command. Depending on the value of the input flag parameter, this request can also clear (reset to 0) the software accumulators for the channels without EMI interfaces. - Input buffer: PAC194X5X_REFRESH_FLAGS - Output buffer: none (zero bytes returned)
IOCTL_PAC194X5X_DATA_REFRESH_RESET_-GLOBAL	Performs a global update of the device driver software accumulators for all the PAC194X/5X devices in the system. Depending on the value of the input flag, this request can also clear (reset to 0) the software accumulators for the channels without EMI interfaces. - Input buffer: PAC194X5X_REFRESH_FLAGS - Output buffer: none (zero bytes returned) **Note:** Windows® SpbCx framework does not support the I^2C general call address (0x00), therefore the driver cannot implement this operation request using the REFRESH_G device commands. Instead, the driver implements this request sending sequential REFRESH commands to all the devices in the system.
IOCTL_PAC194X5X_GET_DATA	Returns the measured values (e.g. voltage, power, accumulated power, sample count) and the timestamp when the measurement was taken. - Input buffer: none - Output buffer: PAC194X5X_DATA
IOCTL_PAC194X5X_GET_LAST_SLOW_DATA	Returns the measured values (voltage, power, accumulated power, sample count) latched in the device's readable registers by the last SLOW signal High-to-Low transition (SLOW limited REFRESH) caused by the system power state transition to Working (S0) from other non-powered state. - Input buffer: none - Output buffer: PAC194X5X_LAST_SLOW_DATA

© 2021-2022 Microchip Technology Inc. and its subsidiaries

TABLE 6-3: DEVICE CONTROL IOCTLs (CONTINUED)

IOCTL Name	Description
IOCTL_PAC194X5X_GET_OVERFLOW	Returns the status of the channel accumulators fullness limit overflow ACC_OVF and the accumulator count fullness limit overflow ACC_COUNT bits from ALERT_STATUS register. Depending on the value of the input flag, this request can also clear (reset to 0) the ACC_OVF and ACC_COUNT flags. - Input buffer: PAC194X5X_GET_OVERFLOW - Output buffer: PAC194X5X_GET_OVERFLOW
IOCTL_PAC194X5X_GET_CTRL	Returns the content of the device registers CTRL, CTRL_ACT, CTRL_LAT - Input buffer: none - Output buffer: PAC194X5X_GET_CTRL
IOCTL_PAC194X5X_SET_CTRL	Provides limited user control over some device configuration bits from CTRL register: SAMPLE_MODE, CHx_OFF, GPIO_ALERT2. The driver rejects as invalid parameters the reserved or the unsupported SAMPLE_MODE values and ignores the CHx_OFF configuration requests which may affect channels with open EMI interface. The driver activates the new device configuration using a REFRESH command. - Input buffer: PAC194X5X_SET_CTRL - Output buffer: none (zero bytes returned) **Notes:** - Change requests for the CHx_OFF bits of the channels with EMI interfaces are ignored. - Change requests for SLOW_ALERT1 are ignored. - The Reserved or the unsupported values for SAMPLE_MODE bits are rejected, with invalid parameters error code STATUS_INVALID_PARAMETER. - SAMPLE_MODE setting to non-adaptive accumulation mode is not supported together with GPIO_ALERT2 setting as SLOW pin. The change requests are rejected (STATUS_INVALID_PARAMETER error). - SAMPLE_MODE setting to Single Shot, Single Shot 8x and SLEEP is rejected for the device with EMI interfaces (STATUS_INVALID_PARAMETER error).
IOCTL_PAC194X5X_GET_CHANNEL_OFF	Returns the content of the channel ON/OFF configuration bits from CTRL, CTRL_ACT and CTRL_LAT registers - Input buffer: none - Output buffer: PAC194X5X_GET_CHANNEL_OFF
IOCTL_PAC194X5X_SET_CHANNEL_OFF	Provides limited user control over CHn_OFF bits from CTRL register. The driver activates the new device configuration using a REFRESH command. - Input buffer: PAC194X5X_CHANNEL_OFF - Output buffer: none (zero bytes returned) **Notes:** - The only accepted values are 0 (ON) and 1 (OFF) for each CHx_OFF input parameter structure field, otherwise the request is rejected (STATUS_INVALID_PARAMETER error). - The driver ignores the change requests to OFF for the channels with EMI interfaces.

TABLE 6-3: DEVICE CONTROL IOCTLs (CONTINUED)

IOCTL Name	Description
IOCTL_PAC194X5X_SET_SAMPLE_MODE	Provides limited user control over device configuration SAMPLE_MODE bits from CTRL register. The driver activates the new device configuration using a REFRESH command. - Input buffer: PAC194X5X_SET_SAMPLE_MODE - Output buffer: none (zero bytes returned) **Notes:** - The unsupported and the Reserved values for SAMPLE_MODE bits are rejected, with invalid parameters error code STATUS_INVALID_PARAMETER. - Any of the non-adaptive SAMPLE_MODE setting requests are rejected if GPIO_ALERT2 is already set as SLOW pin (STATUS_INVALID_PARAMETER error). - SINGLE SHOT, SINGLE SHOT8X and SLEEP modes are rejected if there are EMI interfaces open on this device (STATUS_INVALID_PARAMETER error).
IOCTL_PAC194X5X_SET_GPIO2_MODE	Provides user control over device configuration bits GPIO_ALERT2 from CTRL register. The driver activates the new device configuration using a REFRESH_V command. - Input buffer: PAC194X5X_SET_GPIO2_MODE - Output buffer: none (zero bytes returned) **Notes:** - The SLOW pin configuration request is rejected as invalid parameter if the device sample mode is not adaptive accumulation (STATUS_INVALID_PARAMETER error). - The unsupported input configuration values are rejected (STATUS_INVALID_PARAMETER error).
IOCTL_PAC194X5X_GET_SMBUS_SETTINGS	Returns the value of the SMBUS_SETTINGS device register. - Input buffer: none - Output buffer: PAC194X5X_GET_SMBUS_SETTINGS **Note:** The configuration of the SMBUS_SETTINGS register is controlled exclusively by the driver, except the GPIO_DATA2 bit.
IOCTL_PAC194X5X_GET_ANY_ALERT	Returns the value of the ANY_ALERT bit from the SMBUS_SETTINGS device register. - Input buffer: none - Output buffer: PAC194X5X_GET_ANY_ALERT
IOCTL_PAC194X5X_GET_GPIO_DATA2	Returns the value of the GPIO_DATA2 bit from the SMBUS_SETTINGS device register. - Input buffer: none - Output buffer: PAC194X5X_GET_GPIO_DATA2
IOCTL_PAC194X5X_SET_GPIO_DATA2	Provides user control over the value of the GPIO_DATA2 bit in the SMBUS_SETTINGS device register. - Input buffer: PAC194X5X_SET_GPIO_DATA2 - Output buffer: none

© 2021-2022 Microchip Technology Inc. and its subsidiaries

TABLE 6-3: DEVICE CONTROL IOCTLs (CONTINUED)

IOCTL Name	Description
IOCTL_PAC194X5X_GET_NEG_POWER	Returns the content of the channel polarity and full-scale range (FSR) configuration registers NEG_PWR_FSR, NEG_PWR_FSR_ACT and NEG_PWR_FSR_LAT - Input buffer: none - Output buffer: PAC194X5X_GET_NEG_POWER
IOCTL_PAC194X5X_SET_NEG_POWER	Provides user control over channel polarity and full-scale range (FSR) configuration register NEG_PWR_FSR. Channel polarity and FSR can be changed by the user at any moment, regardless the EMI interface status. The driver activates the new device configuration using a Refresh command. - Input buffer: PAC194X5X_SET_NEG_POWER - Output buffer: none (zero bytes returned)
IOCTL_PAC194X5X_GET_SLOW	Returns the value of the SLOW device register. - Input buffer: none - Output buffer: PAC194X5X_GET_SLOW **Note:** The configuration of the SLOW register is controlled exclusively by the PAC194X/5X device driver.
IOCTL_PAC194X5X_GET_ACCUM_CONFIG	Returns the value of the ACCUM_CONFIG register. - Input buffer: none - Output buffer: PAC194X5X_GET_ACCUM_CONFIG
IOCTL_PAC194X5X_SET_ACCUM_CONFIG	Provides limited user control over the value of the ACCUM_CONFIG register. The driver activates the new device configuration using a REFRESH command. - Input buffer: PAC194X5X_SET_ACCUM_CONFIG - Output buffer: none (zero bytes return) **Note:** - The Reserved values for the ACCUM_CONFIG device register are rejected, with invalid parameters error code STATUS_INVALID_PARAMETER. - The accumulator configuration change requests for the channels which have EMI interfaces are ignored.
IOCTL_PAC194X5X_GET_ALERT_STATUS	Returns the value of the ALERT_STATUS device register: indicates which alert functions met their limit trigger (over/under current, over/under voltage, overpower, accumulator fullness). - Input buffer: none - Output buffer: PAC194X5X_GET_ALERT_STATUS
IOCTL_PAC194X5X_GET_ALERT_ENABLE	Returns the value of the ALERT_ENABLE device register: indicates which alert functions are enabled or disabled (over/under current, over/under voltage, overpower, accumulator fullness, conversion complete). - Input buffer: none - Output buffer: PAC194X5X_GET_ALERT_ENABLE
IOCTL_PAC194X5X_SET_ALERT_ENABLE	Provides user control over the device configuration bits from ALERT_ENABLE register: enable/disable the various alert functions (over/under current, over/under voltage, overpower, accumulator fullness, conversion complete). - Input buffer: PAC194X5X_SET_ALERT_ENABLE - Output buffer: none

TABLE 6-3: DEVICE CONTROL IOCTLs (CONTINUED)

IOCTL Name	Description
IOCTL_PAC194X5X_GET_GPIO_ALERT2	Returns the value of the GPIO_ALERT2 register: indicates which alert signals are assigned to the GPIO/ALERT2 pin. - Input buffer: none - Output buffer: PAC194X5X_GET_GPIO_ALERT2
IOCTL_PAC194X5X_SET_GPIO_ALERT2	Provides user control over the device configuration bits from GPIO_ALERT2 register: assign alert signals to the GPIO/ALERT2 pin. - Input buffer: PAC194X5X_SET_GPIO_ALERT2 - Output buffer: none **Note:** - The GPIO/ALERT2 pin must be configured as alert pin in order to output the alert signals. The GPIO/ALERT2 configuration can be controlled using the IOCTL_PAC194X5X_SET_GPIO2_MODE operation or the IOCTL_PAC194X5X_SET_CTRL operation.
IOCTL_PAC194X5X_GET_OC_LIMIT	Returns the alert triggering limit values from the OC limit registers: OC_limit_Ch1, OC_limit_Ch2, OC_limit_Ch3, OC_limit_Ch4. - Input buffer: none - Output buffer: PAC194X5X_CURRENT_LIMIT
IOCTL_PAC194X5X_SET_OC_LIMIT	Provides user control over the alert triggering limit values from the OC limit registers: OC_limit_Ch1, OC_limit_Ch2, OC_limit_Ch3, OC_limit_Ch4. - Input buffer: PAC194X5X_CURRENT_LIMIT - Output buffer: none
IOCTL_PAC194X5X_GET_UC_LIMIT	Returns the alert triggering limit values from the UC limit registers: UC_limit_Ch1, UC_limit_Ch2, UC_limit_Ch3, UC_limit_Ch4. - Input buffer: none - Output buffer: PAC194X5X_CURRENT_LIMIT
IOCTL_PAC194X5X_SET_UC_LIMIT	Provides user control over the alert triggering limit values from the UC limit registers: UC_limit_Ch1, UC_limit_Ch2, UC_limit_Ch3, UC_limit_Ch4. - Input buffer: PAC194X5X_CURRENT_LIMIT - Output buffer: none
IOCTL_PAC194X5X_GET_OP_LIMIT	Returns the alert triggering limit values from the OP limit registers: OP_limit_Ch1, OP_limit_Ch2, OP_limit_Ch3, OP_limit_Ch4. - Input buffer: none - Output buffer: PAC194X5X_POWER_LIMIT
IOCTL_PAC194X5X_SET_OP_LIMIT	Provides user control over the alert triggering limit values from the OP limit registers: OP_limit_Ch1, OP_limit_Ch2, OP_limit_Ch3, OP_limit_Ch4. - Input buffer: PAC194X5X_POWER_LIMIT - Output buffer: none
IOCTL_PAC194X5X_GET_OV_LIMIT	Returns the alert triggering limit values from the OV limit registers: OV_limit_Ch1, OV_limit_Ch2, OV_limit_Ch3, OV_limit_Ch4. - Input buffer: none - Output buffer: PAC194X5X_VOLTAGE_LIMIT

© 2021-2022 Microchip Technology Inc. and its subsidiaries

TABLE 6-3: DEVICE CONTROL IOCTLs (CONTINUED)

IOCTL Name	Description
IOCTL_PAC194X5X_SET_OV_LIMIT	Provides user control over the alert triggering limit values from the OV limit registers: OV_limit_Ch1, OV_limit_Ch2, OV_limit_Ch3, OV_limit_Ch4. - Input buffer: PAC194X5X_VOLTAGE_LIMIT - Output buffer: none
IOCTL_PAC194X5X_GET_UV_LIMIT	Returns the alert triggering limit values from the UV limit registers: UV_limit_Ch1, UV_limit_Ch2, UV_limit_Ch3, UV_limit_Ch4. - Input buffer: none - Output buffer: PAC194X5X_VOLTAGE_LIMIT
IOCTL_PAC194X5X_SET_UV_LIMIT	Provides user control over the alert triggering limit values from the UV limit registers: UV_limit_Ch1, UV_limit_Ch2, UV_limit_Ch3, UV_limit_Ch4. - Input buffer: PAC194X5X_VOLTAGE_LIMIT - Output buffer: none
IOCTL_PAC194X5X_GET_LIMIT_NSAMPLES	Returns the alert triggering limit values of the limit Nsamples registers: OC_limit_Nsamples, UC_limit_Nsamples, OP_limit_Nsamples, OV_limit_Nsamples, UV_limit_Nsamples. - Input buffer: none - Output buffer: PAC194X5X_GET_LIMIT_NSAMPLES
IOCTL_PAC194X5X_SET_LIMIT_NSAMPLES	Provides user control over the alert triggering limit values from the limit Nsamples registers: OC_limit_Nsamples, UC_limit_Nsamples, OP_limit_Nsamples, OV_limit_Nsamples, UV_limit_Nsamples. - Input buffer: PAC194X5X_SET_LIMIT_NSAMPLES - Output buffer: none
IOCTL_PAC194X5X_GET_ACCUM_LIMITS	Returns the alert triggering limit values of the ACC_Fullness_limits register: - Input buffer: none - Output buffer: PAC194X5X_GET_ACCUM_LIMITS
IOCTL_PAC194X5X_SET_ACCUM_LIMITS	Provides user control over the alert triggering limit values from the ACC_Fullness_limits register: - Input buffer: PAC194X5X_SET_ACCUM_LIMITS - Output buffer: none

Table 6-4 lists the data structures that the user application must provide as input or output data buffers for the IOCTL operations implemented by the driver device control interface.

TABLE 6-4: DEVICE CONTROL STRUCTURES

Structure name	Type	Description
`PAC194X5X_GET_INTF_REV`	Output	Returned by `IOCTL_PAC194X5X_GET_INTERFACE_REVISION` operation. Data field: - Control interface revision value. (Current value: `0x01`)
`PAC194X5X_GET_DEVICE_INFO_SIZE`	Output	Returned by `IOCTL_PAC194X5X_GET_DEVICE_INFO_SIZE` operation. Data fields: - The size allocation requirement for the data buffer returned by `IOCTL_PAC194X5X_GET_DEVICE_INFO` operation.
`PAC194X5X_GET_DEVICE_INFO`	Output	Returned by `IOCTL_PAC194X5X_GET_DEVICE_INFO` operation. Data fields: • PAC194X/5X device product ID, manufacturer ID, revision ID. • The number of channels connected to a power rail (R_{SENSE} > 0). • Channel info for each channel supported by device model (1, 2, 3 or 4 channels): - Channel status (enabled/disabled) as indicated by CHn_OFF flags from CTRL_ACT device register. - EMI interface status (created or not). - Sense resistor (R_{SENSE}) value in microohm units. R_{SENSE} = 0 indicates non-connected channel. - V_{BUS} multiplication factor. - Power rail name. **Note:** - The values reported for R_{SENSE}, V_{BUS} multiplication factors and power rail names are reported by the device's ACPI BIOS DSM. **Caution:** - This structure is variable in size, depending on how many channels the device has and the size of the channel name. `IOCTL_PAC194X5X_GET_DEVICE_INFO_SIZE` operation provides the size of the `PAC194X5X_GET_DEVICE_INFO` structure.
`PAC194X5X_GET_RSENSE_MICRO`	Output	Returned by `IOCTL_PAC194X5X_GET_RSENSE_MICRO` operation. Data fields: four sense resistor values, in micro-ohm units, one for each channel. **Note:** - R_{SENSE} = 0 is reported for the channels which are not available on the devices with less than four channels (PAC1943, PAC1952-2) - R_{SENSE} = 0 on supported channel indicates that the channel is not connected to a monitored power rail and is not used.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

TABLE 6-4: DEVICE CONTROL STRUCTURES (CONTINUED)

Structure name	Type	Description
PAC194X5X_REFRESH_FLAGS	Input	Input flag for the IOCTL_PAC194X5X_DATA_REFRESH_RESET and IOCTL_PAC194X5X_DATA_REFRESH_RESET_GLOBAL operations. If the flag value is 1, the device driver is requested to also clear the software accumulators in addition to the device accumulator registers. **Caution:** The software energy accumulators are cleared only for the channels with no EMI interface (private channels).
PAC194X5X_DATA	Output	Returned by IOCTL_PAC194X5X_GET_DATA operation. Data fields: • Hardware registers digital values latched by the latest REFRESH command: ACC_COUNT, VACCn, VBUSn, VBUSn_AVG, VSENSEn, VSENSEn_AVG, VPOWERn, NEG_PWR_FSR_LAT, CTRL_LAT, ACCUM_CONFIG_LAT • Hardware registers Least Significant bit (LSb) values, in real measurement units, for each device channel: - V_{BUS} LSb value, in picovolts. Multiplied with VBUSn or VACCn (in V_{BUS} count mode) digital value provides the real measured voltage value. - V_{SENSE} LSb value in picovolts. Multiplied with VSENSEn or VACCn (in V_{SENSE} count mode) digital value provides the real measured voltage value. - V_{POWER} LSb value in picowatts. Multiplied with VPOWERn or VACCn (in V_{POWER} count mode) digital value provides the real measured watts value. • Measured values, computed in real measurement units: - V_{RAIL} – rail voltage, in picovolts - $V_{RAILAVG}$ – average rail voltage, in picovolts - V_{SENSE} – sense resistor voltage drop, in picovolts - $V_{SENSEAVG}$ – average sense resistor voltage drop, in picovolts - I_{SENSE} – rail amperage, in picoamps - $I_{SENSEAVG}$ – average rail amperage in picoamps - Power – rail power in picowatts • Driver software accumulators: - SampleCountTotal – total sample counter - EnergyEMI – energy accumulation, as defined by EMI. Only the positive energy increments are accumulated. Reported in picowatt-hours (pWh). - EnergyTotal – signed energy accumulator. Both positive and negative energy values are accumulated. Reported in picowatt-hours (pWh). - CoulombCount – Coulomb accumulator, reported in pico-Amp-seconds (pAs). - VRailAccumulator – V_{RAIL} accumulator, reported in picovolts (pV). • Timestamp – the timestamp when the latest measurements were taken. Reported as a count of 100 nanosecond units since January 1st, 1601. **Caution:** If the bidirectional voltage and/or current is enabled for certain channels the binary format of the reported values is in two's complement (the most significant bit is a sign bit). Therefore, care should be taken to cast the reported values to the necessary data types for the subsequent processing.

TABLE 6-4: DEVICE CONTROL STRUCTURES (CONTINUED)

Structure name	Type	Description
PAC194X5X_LAST_SLOW_DATA	Output	Returned by IOCTL_PAC194X5X_GET_LAST_SLOW_DATA operation. Data fields: • Hardware registers digital values latched by the latest limited REFRESH command triggered by SLOW signal transition: ACC_COUNT, VACCn, VBUSn, VBUSn_AVG, VSENSEn, VSENSEn_AVG, VPOWERn • Hardware registers Least Significant bit (LSb) values, in real measurement units, for each device channel: - V_{BUS} LSb value, in picovolts. Multiplied with VBUSn or VACCn (in V_{BUS} count mode) digital value provides the real measured Voltage value. - V_{SENSE} LSb value in picovolts. Multiplied with VSENSEn or VACCn (in V_{SENSE} count mode) digital value provides the real measured Voltage value. - V_{POWER} LSb value in picowatts. Multiplied with VPOWERn or VACCn (in V_{POWER} count mode) digital value provides the real measured Watts value. • The measured values, computed as real measurement units: - V_{RAIL} – rail voltage, in picovolts - $V_{RAILAVG}$ – average rail voltage, in picovolts - V_{SENSE} – sense resistor voltage drop, in picovolts - $V_{SENSEAVG}$ – average sense resistor voltage drop, in picovolts - I_{SENSE} – rail amperage, in picoamps - $I_{SENSEAVG}$ – average rail amperage in picoamps - Power – rail power, in picowatts • Device settings at the time of the SLOW signal transition: - NegPowerLat – channels' polarity and full scale range settings, in the digital format of the NEG_PWR_FSR register - CtrlLat – sampling mode, GPIO configuration and channels' ON/OFF status, in the digital format of the CTRL register - AccumConfigLat – channels' accumulator configuration, in the digital format of the ACCUM_CONFIG register. **Caution:** If the bidirectional voltage and/or current is enabled for certain channels the binary format of the reported values is in two's complement (the most significant bit is a sign bit). Therefore, care should be taken to cast the reported values to the necessary data types for the subsequent processing.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

TABLE 6-4: DEVICE CONTROL STRUCTURES (CONTINUED)

Structure name	Type	Description
PAC194X5X_GET_OVERFLOW	Input and Output	Input and output data structure for the IOCTL_PAC194X5X_GET_OVERFLOW operation. The operation returns the status of the accumulator limit overflow flags and can clear the flags if requested to. • Input data field: - OvfClear flag – a non-zero value indicates the driver to clear the overflow flags after reading them. • Output data fields: - AccOvf – the value of the ACC_OVF bit from ALERT_STATUS device register. - CountOvf – the value of the ACC_COUNT bit from ALERT_STATUS device register.
PAC194X5X_GET_CTRL	Output	Returned by IOCTL_PAC194X5X_GET_CTRL operation. The data fields contain the values of the device registers: CTRL, CTRL_ACT and CTRL_LAT.
PAC194X5X_SET_CTRL	Input	Input data structure for IOCTL_PAC194X5X_SET_CTRL operation. The data field must contain the value to be written into CTRL register.
PAC194X5X_GET_CHANNEL_OFF	Output	Returned by IOCTL_PAC194X5X_GET_CHANNEL_OFF operation. The data fields contain the channel ON/OFF configuration bit values from CTRL, CTRL_ACT and CTRL_LAT registers.
PAC194X5X_SET_CHANNEL_OFF	Input	Input data structure for IOCTL_PAC194X5X_SET_CHANNEL_OFF operation. The data fields must contain the bit values to be written into each CHn_OFF bit of the CTRL device register. **Note:** - The only accepted values are 0 (ON) and 1 (OFF).
PAC194X5X_SET_SAMPLE_MODE	Input	Input data structure for IOCTL_PAC194X5X_SET_SAMPLE_MODE operation. The data field must contain the value to be written into SAMPLE_MODE bits of the CTRL device register.
PAC194X5X_SET_GPIO2_MODE	Input	Input data structure for IOCTL_PAC194X5X_SET_GPIO2_MODE operation. The data field must contain the value to be written into GPIO_ALERT2 bits of the CTRL device register.
PAC194X5X_GET_SMBUS_SETTINGS	Output	Returned by IOCTL_PAC194X5X_GET_SMBUS_SETTINGS operation. The data field contains the value of the device register SMBUS_SETTINGS.
PAC194X5X_GET_ANY_ALERT	Output	Returned by IOCTL_PAC194X5X_GET_ANY_ALERT operation. The data field contains the value of the ANY_ALERT bit from device register SMBUS_SETTINGS.
PAC194X5X_GET_GPIO_DATA2	Output	Returned by IOCTL_PAC194X5X_GET_GPIO_DATA2 operation. The data field contains the value of the GPIO_DATA2 bit from device register SMBUS_SETTINGS.
PAC194X5X_SET_GPIO_DATA2	Input	Input data structure for IOCTL_PAC194X5X_SET_GPIO_DATA2 operation. The least significant bit of the data field must have the value to be written into GPIO_DATA2 bit of the SMBUS_SETTINGS device register.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

TABLE 6-4: DEVICE CONTROL STRUCTURES (CONTINUED)

Structure name	Type	Description
PAC194X5X_GET_NEG_POWER	Output	Returned by IOCTL_PAC194X5X_GET_NEG_POWER operation. The data fields contain the values of the device registers: NEG_PWR_FSR, NEG_PWR_FSR_ACT and NEG_PWR_FSR_LAT.
PAC194X5X_SET_NEG_POWER	Input	Input data structure for IOCTL_PAC194X5X_SET_NEG_POWER operation. The data field must contain the value to be written into NEG_PWR_FSR register.
PAC194X5X_GET_SLOW	Output	Returned by IOCTL_PAC194X5X_GET_SLOW operation. The data field contains the value of the device register SLOW.
PAC194X5X_GET_ACCUM_CONFIG	Output	Returned by IOCTL_PAC194X5X_GET_ACCUM_CONFIG operation. The data fields contain the values of the device registers: ACCUM_CONFIG, ACCUM_CONFIG_ACT and ACCUM_CONFIG_LAT.
PAC194X5X_SET_ACCUM_CONFIG	Input	Input data structure for IOCTL_PAC194X5X_SET_ACCUM_CONFIG operation. The data field must contain the value to be written into ACCUM_CONFIG register.
PAC194X5X_GET_ALERT_STATUS	Output	Returned by IOCTL_PAC194X5X_GET_ALERT_STATUS operation. The data field contains the value of the device register ALERT_STATUS.
PAC194X5X_GET_ALERT_ENABLE	Output	Returned by IOCTL_PAC194X5X_GET_ALERT_ENABLE operation. The data field contains the value of the device register ALERT_ENABLE.
PAC194X5X_SET_ALERT_ENABLE	Input	Input data structure for IOCTL_PAC194X5X_SET_ALERT_ENABLE operation. The data field must contain the value to be written into ALERT_ENABLE register.
PAC194X5X_GET_GPIO_ALERT2	Output	Returned by IOCTL_PAC194X5X_GET_GPIO_ALERT2 operation. The data field contains the value of the device register GPIO_ALERT2.
PAC194X5X_SET_GPIO_ALERT2	Input	Input data structure for IOCTL_PAC194X5X_SET_GPIO_ALERT2 operation. The data field must contain the value to be written into GPIO_ALERT2 register.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

TABLE 6-4: DEVICE CONTROL STRUCTURES (CONTINUED)

Structure name	Type	Description
PAC194X5X_CURRENT_LIMIT	Output or Input	Output: returned by IOCTL_PAC194X5X_GET_OC_LIMIT or IOCTL_PAC194X5X_GET_UC_LIMIT operations. The data fields contain all the channels current threshold values for over-current or under-current limit alerts. - The digital values of the OC_Limit or UC_Limit registers are converted and reported by the device driver into picoamp units. Input: data structure for IOCTL_PAC194X5X_SET_OC_LIMIT or IOCTL_PAC194X5X_SET_UC_LIMIT operations. The data fields for all the channels must contain the current threshold values for over-current or under-current limit alerts, expressed in picoamp units. - The user input threshold values expressed in picoamp units are converted by the device driver into the digital values to be written into OC_Limit or UC_Limit registers. **Caution:** If the input threshold value exceeds the channel full scale range, the driver writes into the limit register the scale maximum (or minimum) supported value.
PAC194X5X_POWER_LIMIT	Output or Input	Output: returned by IOCTL_PAC194X5X_GET_OP_LIMIT operation. The data fields contain all the channels' power threshold values for over-power limit alerts. - The digital values of the OP_Limit registers are converted and reported by the device driver into picowatt units. Input: data structure for IOCTL_PAC194X5X_SET_OP_LIMIT operations. The data fields for all the channels must contain the power threshold values for over-power limit alerts, expressed in picowatt units. - The user input threshold values expressed in picowatt units are converted by the device driver into the digital values to be written into OP_Limit registers. **Caution:** If the input threshold value exceeds the channel full scale range, the driver writes into the limit register the scale maximum (or minimum) supported value.

TABLE 6-4: DEVICE CONTROL STRUCTURES (CONTINUED)

Structure name	Type	Description
PAC194X5X_VOLTAGE_LIMIT	Output or Input	Output: returned by IOCTL_PAC194X5X_GET_OV_LIMIT or IOCTL_PAC194X5X_GET_UV_LIMIT operations. The data fields contain all the channels' voltage threshold values for over-voltage or under-voltage limit alerts. - The digital values of the OV_Limit or UV_Limit registers are converted and reported by the device driver into picovolt units. Input: data structure for IOCTL_PAC194X5X_SET_OV_LIMIT or IOCTL_PAC194X5X_SET_UV_LIMIT operations. The data fields for all the channels must contain the voltage threshold values for over-voltage or under-voltage limit alerts, expressed in picovolt units. - The user input threshold values expressed in picovolt units are converted by the device driver into the digital values to be written into OV_Limit or UV_Limit registers. **Caution:** If the input threshold value exceeds the channel full scale range, the driver writes into the limit register the scale maximum (or minimum) supported value.
PAC194X5X_GET_LIMIT_NSAMPLES	Output	Returned by IOCTL_PAC194X5X_GET_LIMIT_NSAMPLES operation. The data fields contain the values of the device registers: OC_limit_Nsamples, UC_limit_Nsamples, OP_limit_Nsamples, OV_limit_Nsamples and UV_limit_Nsamples.
PAC194X5X_SET_LIMIT_NSAMPLES	Input	Input data structure for IOCTL_PAC194X5X_SET_LIMIT_NSAMPLES operation. The data field must contain the value to be written into device registers: OC_limit_Nsamples, UC_limit_Nsamples, OP_limit_Nsamples, OV_limit_Nsamples and UV_limit_Nsamples.
PAC194X5X_GET_ACCUM_LIMITS	Output	Returned by IOCTL_PAC194X5X_GET_ACCUM_LIMITS operation. The data field contain the value of the ACC_Fullness_limits register.
PAC194X5X_SET_ACCUM_LIMITS	Input	Input data structure for IOCTL_PAC194X5X_SET_ACCUM_LIMITS operation. The data field must contain the value to be written into ACC_Fullness_limits register.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Chapter 7. PAC194X/5X Driver Characteristics

7.1 INTRODUCTION

This section provides information about the driver features which can influence the design decisions for the user applications processing the data reported by the driver.

- Multiple I/O Requests
- Energy Meter Interfaces (EMI) vs. Device Control Interfaces
- Software Accumulators
- Low-Power Mode and Power States

7.2 MULTIPLE I/O REQUESTS

Windows E3 module and multiple other user applications (or multiple instances of the same application) may concurrently invoke input/output device operation requests (IOCTL) for data reporting or device configuration changes. The PAC194X/5X device driver is queuing all the requests and executes them one at a time, sequentially, in a run-to-completion way, ensuring the consistency of the I^2C transactions.

7.3 ENERGY METER INTERFACES (EMI) VS. DEVICE CONTROL INTERFACES

For each PAC194X/5X device, the device driver creates one control interface and zero or more EMI interfaces. Each device channel may have one EMI interface if several conditions are met (as indicated by the parameters returned by the device ACPI BIOS DSM):

- The channel is connected to a power rail ($R_{SENSE} > 0$).
- The channel power rail name is a non-empty string.
- The EMI enable/disable mask allows the interface creation.
- The EMI interface accumulates only the positive energy increments and reports the positive energy accumulation and the timestamp of the last measurement.

The control interface reports all the measurements performed by the PAC194X/5X device: energy, power, voltage, current, limit alerts. This includes the positive-only energy accumulation and the timestamp values reported by EMI interfaces but also the total energy accumulation, which takes into account also the negative energy increments that may occur on a bipolar channel.

The Windows Energy Estimation Engine (E3) collects only the energy figures reported by the EMI interfaces (EMI IOCTL operations).

CAUTION
The power rail names must comply with Microsoft® E3 power rail naming conventions to ensure that E3 is properly using the data.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

The user applications can collect the data from EMI, control or both interface types:

- EMI interface implements the Microsoft® defined control codes (IOCTL) and data structures and this makes the user application capable to collect data from any EMI compliant metering device vendor.
- Control interface provides much more information with Microchip defined IOCTL and data structures.

The control interface provides also access to the device configuration registers, with some restrictions:

- Some device configuration options are controlled exclusively by the device driver: I^2C interface configuration, register pointer skip feature, SLOW/$\overline{\text{ALERT1}}$ pin configuration.
- The change of the channel ON/OFF status or the change of the channel accumulator to alternate mode (V_{SENSE} or V_{BUS} accumulation) are accepted only if the affected channels have no EMI interface.

7.4 SOFTWARE ACCUMULATORS

The PAC194X/5X device driver implements for each device a set of software accumulators:

- Sample counter (per device)
- EMI energy accumulators (per channel)
- Total energy accumulator (per channel)
- Coulomb accumulator (per channel)
- Rail voltage accumulator (per channel)

Their purpose is to:

- Accumulate the real measurements units for energy (pWh) and Coulomb counter (pAs), as computed by the driver using the value of the channel accumulator register (VACCn), the sense resistor value (R_{SENSE}) value and the data sampling frequency.
- Extend the total accumulation time limit beyond the device capabilities thanks to the 64-bit resolution of the software accumulators.

In order to extend the accumulation time limit, every time the device EMI interface is requested to report the energy values, the driver reads the device register accumulators and clears them using a REFRESH command. Then, it updates the software accumulators and completes the data request. Windows E3 module is typically collecting the EMI energy every few minutes. But, if the device has no EMI interface (all the channels are private) the driver still can update the software accumulators and clear the device accumulators every time expires the watchdog timer created by the driver for the device. The driver configures the watchdog timer interval with the value provided by the device ACPI BIOS DSM (or uses the hardcoded 240-seconds interval if the DSM provided value exceeds the acceptable limits).

7.5 LOW-POWER MODE AND POWER STATES

The recommended integration scenario with Windows 10 or Windows 11 systems described for the PAC194X/5X devices assumes that the devices receive power regardless of the power state of the host operating system. If the power is not physically removed from the system, the PAC194X/5X devices continue to accumulate the energy data. (For details about the device integration with the host system refer the chapter **Chapter 2. "Hardware Integration"**.) Therefore, there are several aspects that must be addressed by the device driver once the operating system returns to working state (S0) or exits the working state:

© 2021-2022 Microchip Technology Inc. and its subsidiaries

- Determine if the device has gone through a Power-On Reset (POR). In this case, the device was initialized and must be applied additional configuration options. However, the device registers contain valid data which can be added to the software accumulators.
- Collect the data latched by the SLOW pin transition from high-to-low. The power was not removed from the PAC194X/5X devices, but the operating system is returning to the working state (S0) from one of the following low power states:
  - Shut down – S5,
  - Hibernate or Hybrid sleep or Hybrid shutdown – S4,
  - Sleep – S1/S2/S3
- Configure the PAC194X/5X device into a low power mode (Idle) whenever the system exits the working state (S0) in order to reduce its own power consumption.

Please refer to the Microsoft article System Power States for a detailed description of the power states.

7.5.1 Low-Power Mode

The PAC194X/5X own device power consumption is proportional with the data sampling frequency. But the sampling frequency can be reduced when the operating system exits the working state (S0), without affecting the measurements.

The device driver uses three sampling rate configuration options:

- Active sampling rate, if the system is running in working state – S0. The active sampling rate value is initialized with the parameter reported by the ACPI BIOS DSM, but the user application can change it using `IOCTL_PAC194X5X_SET_CTRL` or `IOCTL_PAC194X5X_SET_SAMPLE_MODE` control interface operations.
- Idle sampling rate if the system enters one of the low power states:
  - Sleep power states: S1, S2 or S3
  - Hibernation, Hybrid sleep or Hybrid shutdown: S4
  - Modern Standby: S0 low-power idle

The Idle sampling frequency value is initialized with the parameter reported by the ACPI BIOS DSM and the user applications cannot change it.

- SINGLE_SHOT sampling mode if the system is shut down (S5). This configuration can provide several benefits, depending on what PAC194X/5X system integration options have been implemented and assuming that the device continues to receive power while the system is shut down (S5):
  - If the SLOW pin is activated by hardware during system S5, the device accumulates energy at 8 samples per second.
  - If the SLOW pin is not activated during system S5, the device is doing no conversion cycle, effectively entering the Sleep mode since there is no REFRESH command from the system to process.
  - While the host system is shutdown the PAC194X/5X device can still service commands received from a different I^2C controller. For example, an external debug kit may be connected for assessing the system energy consumption during S5 power state.

Windows 10 and Windows 11 operating systems implement the Modern Standby system power model which can deliver superior user experience while reducing the overall system energy consumption: Instant On device availability, background processing and connectivity. This model has specific requirements for the hardware peripherals and their device drivers. On such systems the Sleep power states (S1/S2/S3) have been replaced by the S0 low power idle state.

The PAC194X/5X device driver supports the Modern Standby power model by:

- Registering itself to the Windows-Directed Power Management Framework (DFx).
- Registering the display state and the user presence Power Setting Callback functions (`GUID_CONSOLE_DISPLAY_STATE` and `GUID_GLOBAL_USER_PRESENCE`).

If the driver detects both `PowerMonitorOff` and `PowerUserInactive` events it configures the device sampling frequency to the Idle ACPI BIOS DSM parameter value.

The driver reverts the device sampling frequency to the Active value if detects that either the console display state is `PowerMonitorOn` or the user presence state is `PowerUserActive`.

Table 7-1 summarizes the operating power states and the corresponding PAC194X/5X device sampling frequency.

TABLE 7-1: PAC194X/5X DEVICE SAMPLING FREQUENCY

Windows® Power State	PAC194X/5X Sample Frequency	Comments
S0	Active	- Initialized with ACPI BIOS DSM Active parameter. - User can change it with `IOCTL_PAC194X5X_SET_CTRL` or `IOCTL_PAC194X5X_SET_SAMPLE_MODE`
S0 low-power idle	Idle	- ACPI BIOS DSM Idle parameter
S1, S2, S3	Idle or SLOW	- ACPI BIOS DSM Idle parameter or - 8 Hz if SLOW pin is activated
S4	Idle or SLOW	- ACPI BIOS DSM Idle parameter or - 8 Hz if SLOW pin is activated
S5	SINGLE_SHOT or SLOW	- No sampling or - 8 Hz if SLOW pin is activated

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Appendix A. Sample ACPI ASL PAC194X/5X Device Definition

EXAMPLE A-1: SAMPLE ACPI ASL PAC194X/5X DEVICE DEFINITION

```
Device(PA01)
{
    Name(_HID, "MCHP1940")
    Name(_UID, 1)

    // Lowest power D-State supported by the device is D3
    Name (_S0W, 3)

    // Device Status: present, enabled, and functioning properly
    Method(_STA, 0x0, NotSerialized)
    {
        Return(0xf)
    }

    // Current Resources Settings
    Method(_CRS, 0x0, NotSerialized)
    {
        Name(RBUF, ResourceTemplate()
        {
                I2CSerialBus(0x10,                  // 7-bit Peripheral Address
                             ControllerInitiated,   // Controller or peripheral?
                             400000,                // Connection Speed in Hz
                             AddressingMode7Bit,    // 7-bit or 10-bit addressing?
                             "\\_SB.PCI0.I2C0",     // I2C Controller to which PAC is connected
                             0,                     // Resource Index
                             ResourceConsumer)      // Consumer or Producer?
        })
        Return(RBUF)
    }

    // _DSM - Device Specific Method
    //
    // This method returns configuration information that PAC194x/5x driver
    // uses for the device initial configuration.
    //
    // The DSM UUID for the Microchip PAC194x/5x: {721F1534-5D27-4B60-9DF4-41A3C4B7DA3A}.
    // This must match what the Windows PAC194x/5x driver expects.
    //
    // Returns:
    //         Either: A Buffer (for supported Functions, or an error)
    //                 A Package containing PAC194x/5x resources (configuration values)
    //
    // Input Arguments (per _DSM standard):
    //     Arg0: UUID - Function Identifier
    //     Arg1: Integer - Revision
    //     Arg2: Integer - Function Index
    //     Arg3: Package - Parameters (not used in our implementation)
    //
    Function(_DSM, {BuffObj, PkgObj}, {BuffObj, IntObj, IntObj, PkgObj})
    {
```

```
        //  Is PAC194x/5x DSM UUID being invoked?
        if(LNotEqual(Arg0, ToUUID("721F1534-5D27-4B60-9DF4-41A3C4B7DA3A")))
        {
            return(Buffer() {0x0})    // incorrect UUID, return NULL for error
        }

        // Function number check
        switch(ToInteger(Arg2))
        {
            // Function 0 - returns a bit-mask of supported functions
            case(0)
            {
                switch(ToInteger(Arg1))    // revision check
                {
                    // Revision 0: function 1->7 are supported
                    case(0) {return (Buffer() {0xFF})}
                }
                break;
            }

            // Function 1 - returns the names of the monitored power rails.
            // The names should be compliant with Windows EMI power rail taxonomy.
            // If the rail name is NULL ("") -> no EMI created for the channel ("private channel").
            case(1)
            {
                Name(BUF1, Package()
                {   // Rail Names set
                    "GPU"       ,        // Channel 1 name
                    "CPU_CORES" ,        // Channel 2 name
                    "SENSORS"   ,        // Channel 3 name
                    "WIFI"               // Channel 4 name
                })
                return(BUF1)
            }

            // Function 2 - returns the sense resistor values, in micro-Ohms.
            case(2)
            {
                Name(BUF2, Package()
                {   // Sense resistors values (micro-Ohms)
                    24900000,        // Channel 1 resistor
                    49900000,        // Channel 2 resistor
                    75000000,        // Channel 3 resistor
                   100000000         // Channel 4 resistor
                })
                return(BUF2)
            }

            // Function 3 - returns the EMI enable/disable bit-mask
            case(3)
            {
                Name(BUF3, Package()
                {   // EMI bitmask - CH1:CH2:CH3:CH4
                    0xF           // CHn = 1: EMI enable for channel 'n'
                                  // CHn = 0: EMI disable for channel 'n' ("private channel")
                })
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
return(BUF3)
            }

            // Function 4 - returns Vsense and Vbus polarity and full scale range configs
            case(4)
            {
                Name(BUF4, Package()
                {   // Vsense configs
                    0x0, 0x0, 0x0, 0x0,   // CFG_VS1, CFG_VS2, CFG_VS3, CFG_VS4
                                          // CFG_VSn: Vsense configuration for channel 'n'
                                          // {0x0 = unipolar; 0x1 = bipolar; 0x2 = half scale bipolar}
                    // Vbus configs
                    0x0, 0x0, 0x0, 0x0    // CFG_VB1, CFG_VB2, CFG_VB3, CFG_VB4
                                          // CFG_VBn: Vbus configuration for channel 'n'
                                          // {0x0 = unipolar; 0x1 = bipolar; 0x2 = half scale bipolar}
                })
                return(BUF4)
            }

            // Function 5 - returns the sample frequencies for ACTIVE and IDLE modes
            case(5)
            {
                Name(BUF5, Package()
                {
                    1024,   // ACTIVE sps - accepted values = {1024, 256, 64, 8}
                       8    // IDLE sps   - accepted values = {1024, 256, 64, 8}
                })
                return(BUF5)
            }

            // Function 6 - returns the REFRESH watchdog timer interval
            case(6)
            {
                Name(BUF6, Package()
                {
                    900     // seconds (min=60, max=60000)
                })
                return(BUF6)
            }

            // Function 7 - returns the Vbus multiplication factors for PAC194x/5x-2
            // PAC194x/5x-2 power meter VBUS+ pins may be connected to voltage dividers
            // to measure rail voltage higher than device Vbus FSR.
            case(7)
            {
                Name(BUF7, Package()
                {   // K = 1000 * (Vrail / Vbus). Must be integer, positive value.
                    1000,        // Channel 1 K-factor
                    1000,        // Channel 2 K-factor
                    1000,        // Channel 3 K-factor
                    1000         // Channel 4 K-factor
                })
                return(BUF7)
            }

        } // switch(Arg2)
```

```
        // Return an error (a buffer with a value of zero)
        // if we didn't return anything else above
        return(Buffer() {0x0})

    } // _DSM

} // Device(PA01)
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

Appendix B. Testing the PAC194X/5X ACPI Integration

CAUTION

The tools and the process presented in this appendix are not the only method to integrate the PAC194x/5x ACPI ASL, compile and deploy it on a target system. This method does not replace the original system BIOS firmware from the Flash and works only if Windows 10 or Windows 11 OS on the target system is configured in "Test Mode". The method works only if the target system already has Windows 10 or Windows 11 installed on it, and the user has the possibility to install and run on it the presented tools. ASL code changes and deployment must be done with great care because of the potential risk to cause the Windows OS to crash and the loss of data.

The tools are listed below:

- `iasl` and `acpidump` tools from ACPICA Windows Binary Tools.
- `asl` compiler from Microsoft Windows Driver Kit (WDK) (use the 64-bit or the 32-bit version, depending on the target system OS type)
- `bcdedit` utility, installed by default in Windows 10 or Windows 11
- `Notepad` utility (installed by default in Windows 10 or Windows 11) or any other text editor that runs on the target system
- `shutdown` utility (installed by default in Windows 10 or Windows 11)

The process is as follows:

1. Copy the tools and the sample ASL code to the target system

- Create a working folder (for example, ACPI_MCHP1940) and copy the PAC194X/5X ASL sample code in a simple text file.
- Copy the `asl` compiler and the ACPICA tools in the working folder or, alternatively, create for them a dedicated tools folder and put it in the system path.

2. Dump the target system ACPI code

- Open an elevated command prompt (administrator command prompt) and change the working directory to the working folder just created.
- Use the `acpidump` tool to obtain the system ACPI tables in binary format:

```
C:\ACPI_MCHP1940>acpidump -b
```

Several files will be created on the working folder, one for each ACPI table type, having the name of the contained ACPI table and the `.dat` extension.

3. Disassemble the binary ACPI code

- Use the `iasl` tool to disassemble the binary ACPI code into ASL code:

```
C:\ACPI_MCHP1940>iasl -d *.dat
```

Several files will be created on the working folder, one for each ACPI table type, having the name of the contained ACPI table and the `.dsl` extension.

4. Add the PAC194X/5X ASL code into ACPI SSDT

- Open the `ssdt.dsl` file in the text editor, add the PAC194X/5X ASL sample code and modify it to match the target system architecture.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Tip: look for the I^{2}C controller ACPI definition in DSDT or SSDT and note the ACPI name. This name is needed to define the PAC194X/5X device I^{2}C connection; for more information, see the I^{2}C Serial Bus section from **Appendix A. "Sample ACPI ASL PAC194X/5X Device Definition"**.

5. Compile the modified SSDT

- Use `iasl` tool to generate the binary SSDT code:

```
C:\ACPI_MCHP1940>iasl ssdt.dsl
```

The file `ssdt.aml` is generated if there is no compilation error.

6. Load the new ACPI SSDT in the Windows registry.

Windows `acpi.sys` driver provide the ACPI code developers the option to override the original ACPI code from the system Flash with a different version saved in the Windows registry.

- Use the `asl` tool to load the ACPI table into Windows registry:

```
C:\ACPI_MCHP1940>asl /loadtable -v ssdt.aml
```

Note: this command works only from the elevated command prompt.

Tip: In case the Windows registry already contains an ACPI code override, it must be unloaded before loading the new version. Provide the option `-d` to the `asl` tool in order to achieve this:

```
C:\ACPI_MCHP1940>asl /loadtable -v -d ssdt.aml
```

7. Switch Windows in Test mode

The ACPI table override feature presented above is enabled only in Windows Test mode.

- Use the `bcdedit` utility to enable the Test mode:

```
C:\ACPI_MCHP1940>bcdedit /set testsigning ON
```

Tip: In order to disable the Test mode, replace ON with OFF in the above example.

8. Reboot the system

- Use the `shutdown` utility to make sure that a full power cycle is executed, not affected by the Windows 10 or Windows 11 Fast Start facility:

```
C:\ACPI_MCHP1940>shutdown -r -t 0
```

9. Check the Device Manager

At this stage, because there is no driver installed yet, the PAC194X/5X devices should be enumerated by the Windows Device Manager utility under Unknown Devices category.

10. Install the PAC194X/5X Windows device driver

Read and follow the steps presented by **Chapter 4. "Driver Installation"** from the PAC194X/5X Device Driver User's Guide.

CAUTION
In case that a BIOS upgrade or a BIOS configuration change is needed, it is recommended to do the following:

- First, disable the ACPI code override either by unloading the code from Windows registry (see Step 6) or by turning off the Test mode (see Step 7).
- Repeat the steps listed above, starting with Step 2.

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Appendix C. PAC194X/5X Device Control Interface

PAC194X/5X device control interface created by the PAC194X/5X Microsoft Windows 10 or Windows 11 Device Driver is defined by the example C header files listed in this appendix:

- `PAC194X/5X_HW.h` – contains PAC194X/5X device specific data type definitions
- `PAC194X/5X_INTF.h` – contains the control interface IOCTL and data type definitions

EXAMPLE C-1: PAC194X5X_HW.h

```
////////////////////////////////////////////////////////////////////////////////
//
//      (C) 2019-2021 Microchip Technology Inc. and its subsidiaries.
//      (C) Copyright 2016-2019 OSR Open Systems Resources, Inc.
//
//      Subject to your compliance with these terms, you may use Microchip software
//      and any derivatives exclusively with Microchip products. You're responsible
//      for complying with 3rd party license terms applicable to your use of 3rd party
//      software (including open source software) that may accompany Microchip software.
//      SOFTWARE IS "AS IS." NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY,
//      APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT,
//      MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. 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 SOFTWARE, 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 RELATED TO THE SOFTWARE WILL NOT EXCEED AMOUNT OF FEES, IF ANY,
//      YOU PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
//
//
//    MODULE:
//       PAC194x5x_HW.h
//
//    ABSTRACT:
//       Device-Specific Hardware Interface for the Microchip PAC194x_5x EMI Driver
//
//    AUTHOR(S):
//       OSR Open Systems Resources, Inc.
//       Microchip Technology, Inc.
//
//    REVISION:
//       v1.2.0
//
////////////////////////////////////////////////////////////////////////////////
#pragma once

////////////////////////
// DEVICE CONSTANTS   //
////////////////////////
```

```
#define PAC_MAXIMUM_CHANNELS 4

//////////////////////////////////////
// DEVICE REGISTER VALUE CONSTANTS //
//////////////////////////////////////

// PAC194x family
#define PAC_PRODUCT_ID_1941_1       0x68
#define PAC_PRODUCT_ID_1942_1       0x69
#define PAC_PRODUCT_ID_1943_1       0x6A
#define PAC_PRODUCT_ID_1944_1       0x6B
#define PAC_PRODUCT_ID_1941_2       0x6C
#define PAC_PRODUCT_ID_1942_2       0x6D
// PAC195x family
#define PAC_PRODUCT_ID_1951_1       0x78
#define PAC_PRODUCT_ID_1952_1       0x79
#define PAC_PRODUCT_ID_1953_1       0x7A
#define PAC_PRODUCT_ID_1954_1       0x7B
#define PAC_PRODUCT_ID_1951_2       0x7C
#define PAC_PRODUCT_ID_1952_2       0x7D

#define PAC_MANUFACTURER_ID         0x54
#define PAC_REVISION_0              0x00    //silicon revision starting number

#define PAC_194X_VBUS_FSR_MV           9000 //milli-Volts
#define PAC_195X_VBUS_FSR_MV          32000 //milli-Volts
#define PAC_194X_VSENSE_FSR_MV          100 //milli-Volts
#define PAC_195X_VSENSE_FSR_MV          100 //milli-Volts

//////////////////////////////
// DEVICE REGISTER ADDRESSES //
//////////////////////////////

// Main control registers
#define PAC_REGISTER_REFRESH            0x00
#define PAC_REGISTER_CTRL               0x01

// Measurement registers
#define PAC_REGISTER_ACC_COUNT          0x02
#define PAC_REGISTER_VACC1              0x03
#define PAC_REGISTER_VACC2              0x04
#define PAC_REGISTER_VACC3              0x05
#define PAC_REGISTER_VACC4              0x06

#define PAC_REGISTER_VBUS1              0x07
#define PAC_REGISTER_VBUS2              0x08
#define PAC_REGISTER_VBUS3              0x09
#define PAC_REGISTER_VBUS4              0x0A

#define PAC_REGISTER_VSENSE1            0x0B
#define PAC_REGISTER_VSENSE2            0x0C
#define PAC_REGISTER_VSENSE3            0x0D
#define PAC_REGISTER_VSENSE4            0x0E
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
#define PAC_REGISTER_VBUS1_AVG          0x0F
#define PAC_REGISTER_VBUS2_AVG          0x10
#define PAC_REGISTER_VBUS3_AVG          0x11
#define PAC_REGISTER_VBUS4_AVG          0x12

#define PAC_REGISTER_VSENSE1_AVG        0x13
#define PAC_REGISTER_VSENSE2_AVG        0x14
#define PAC_REGISTER_VSENSE3_AVG        0x15
#define PAC_REGISTER_VSENSE4_AVG        0x16

#define PAC_REGISTER_VPOWER1            0x17
#define PAC_REGISTER_VPOWER2            0x18
#define PAC_REGISTER_VPOWER3            0x19
#define PAC_REGISTER_VPOWER4            0x1A

// Control registers
#define PAC_REGISTER_SMBUS_SETTINGS     0x1C
#define PAC_REGISTER_NEG_PWR_FSR        0x1D
#define PAC_REGISTER_REFRESH_G          0x1E
#define PAC_REGISTER_REFRESH_V          0x1F
#define PAC_REGISTER_SLOW               0x20
#define PAC_REGISTER_CTRL_ACT           0x21
#define PAC_REGISTER_NEG_PWR_FSR_ACT    0x22
#define PAC_REGISTER_CTRL_LAT           0x23
#define PAC_REGISTER_NEG_PWR_FSR_LAT    0x24
#define PAC_REGISTER_ACCUM_CONFIG       0x25
#define PAC_REGISTER_ALERT_STATUS       0x26
#define PAC_REGISTER_SLOW_ALERT1        0x27
#define PAC_REGISTER_GPIO_ALERT2        0x28
#define PAC_REGISTER_ACCUM_LIMITS       0x29
#define PAC_REGISTER_OC1_LIMIT          0x30
#define PAC_REGISTER_OC2_LIMIT          0x31
#define PAC_REGISTER_OC3_LIMIT          0x32
#define PAC_REGISTER_OC4_LIMIT          0x33
#define PAC_REGISTER_UC1_LIMIT          0x34
#define PAC_REGISTER_UC2_LIMIT          0x35
#define PAC_REGISTER_UC3_LIMIT          0x36
#define PAC_REGISTER_UC4_LIMIT          0x37
#define PAC_REGISTER_OP1_LIMIT          0x38
#define PAC_REGISTER_OP2_LIMIT          0x39
#define PAC_REGISTER_OP3_LIMIT          0x3A
#define PAC_REGISTER_OP4_LIMIT          0x3B
#define PAC_REGISTER_OV1_LIMIT          0x3C
#define PAC_REGISTER_OV2_LIMIT          0x3D
#define PAC_REGISTER_OV3_LIMIT          0x3E
#define PAC_REGISTER_OV4_LIMIT          0x3F
#define PAC_REGISTER_UV1_LIMIT          0x40
#define PAC_REGISTER_UV2_LIMIT          0x41
#define PAC_REGISTER_UV3_LIMIT          0x42
#define PAC_REGISTER_UV4_LIMIT          0x43
#define PAC_REGISTER_OC_LIMIT_NSAMPLES  0x44
#define PAC_REGISTER_UC_LIMIT_NSAMPLES  0x45
#define PAC_REGISTER_OP_LIMIT_NSAMPLES  0x46
#define PAC_REGISTER_OV_LIMIT_NSAMPLES  0x47
#define PAC_REGISTER_UV_LIMIT_NSAMPLES  0x48
#define PAC_REGISTER_ALERT_ENABLE       0x49
```

```
#define PAC_REGISTER_ACCUM_CONFIG_ACT   0x4A
#define PAC_REGISTER_ACCUM_CONFIG_LAT   0x4B

// Chip IDs
#define PAC_REGISTER_PRODUCT_ID         0xFD
#define PAC_REGISTER_MANUFACTURER_ID    0xFE
#define PAC_REGISTER_REVISION_ID        0xFF

///////////////////////////////////
// DEVICE REGISTER SIZES (bytes) //
///////////////////////////////////
#define PAC_REGISTER_ACC_COUNT_SZ       4
#define PAC_REGISTER_VACCN_SZ           7
#define PAC_REGISTER_VPOWERN_SZ         4
#define PAC_REGISTER_OPLIMIT_SZ         3

///////////////////////////////////
// DEVICE REGISTER WIDTH (bits)  //
///////////////////////////////////
#define PAC_REGISTER_VPOWERN_WIDTH      30
#define PAC_REGISTER_VBUSN_WIDTH        16
#define PAC_REGISTER_VSENSEN_WIDTH      16
#define PAC_REGISTER_VLIMIT_WIDTH       16
#define PAC_REGISTER_PLIMIT_WIDTH       24

//////////////////////////////////
// DEVICE CONFIGURATION VALUES //
//////////////////////////////////

// SAMPLE_MODE
#define SAMPLE_MODE_SPS_ADAPT_1024      0x0
#define SAMPLE_MODE_SPS_ADAPT_256       0x1
#define SAMPLE_MODE_SPS_ADAPT_64        0x2
#define SAMPLE_MODE_SPS_ADAPT_8         0x3
#define SAMPLE_MODE_SPS_1024            0x4
#define SAMPLE_MODE_SPS_256             0x5
#define SAMPLE_MODE_SPS_64              0x6
#define SAMPLE_MODE_SPS_8               0x7
#define SAMPLE_MODE_SINGLE_SHOT         0x8
#define SAMPLE_MODE_SINGLE_SHOT8X       0x9
#define SAMPLE_MODE_FAST                0xA
#define SAMPLE_MODE_BURST               0xB
#define SAMPLE_MODE_RESERVED1           0xC
#define SAMPLE_MODE_RESERVED2           0xD
#define SAMPLE_MODE_RESERVED3           0xE
#define SAMPLE_MODE_SLEEP               0xF

// GPIO_ALERT2
#define GPIO2_MODE_ALERT                0x0
#define GPIO2_MODE_IN                   0x1
#define GPIO2_MODE_OUT                  0x2
#define GPIO2_MODE_SLOW                 0x3

// ACCUM_CONFIG
#define ACC_CONFIG_VPOWER               0x0
#define ACC_CONFIG_VSENSE               0x1
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
#define ACC_CONFIG_VBUS                 0x2
#define ACC_CONFIG_RESERVED             0x3

//////////////////////////////////
// DEVICE REGISTER DEFINITIONS //
//////////////////////////////////

#pragma push(pack:1)

typedef struct _PAC194X5X_CTRL_REGISTER {
//PAC register MSB mapped into struct LSB
    UCHAR SlowAlert1 : 2;
    UCHAR GpioAlert2 : 2;
    UCHAR SampleMode : 4;
//PAC register LSB mapped into struct MSB
    UCHAR : 4;
    UCHAR CH4_OFF    : 1;
    UCHAR CH3_OFF    : 1;
    UCHAR CH2_OFF    : 1;
    UCHAR CH1_OFF    : 1;
} PAC194X5X_CTRL_REGISTER, *PPAC194X5X_CTRL_REGISTER;

typedef struct _PAC194X5X_SMBUS_SETTINGS_REGISTER {
    UCHAR I2cHiSpeed : 1;
    UCHAR NoSkip     : 1;
    UCHAR ByteCount  : 1;
    UCHAR Timeout    : 1;
    UCHAR POR        : 1;
    UCHAR AnyAlert   : 1;
    UCHAR GpioData1  : 1;
    UCHAR GpioData2  : 1;
} PAC194X5X_SMBUS_SETTINGS_REGISTER, *PPAC194X5X_SMBUS_SETTINGS_REGISTER;

typedef struct _PAC194X5X_ACCUM_CONFIG_REGISTER {
    UCHAR ACC4_CONFIG : 2;
    UCHAR ACC3_CONFIG : 2;
    UCHAR ACC2_CONFIG : 2;
    UCHAR ACC1_CONFIG : 2;
} PAC194X5X_ACCUM_CONFIG_REGISTER, *PPAC194X5X_ACCUM_CONFIG_REGISTER;


typedef struct _PAC194X5X_NEG_PWR_FSR_REGISTER {
// PAC register MSB mapped into struct LSB
    UCHAR CFG_VS4 : 2;
    UCHAR CFG_VS3 : 2;
    UCHAR CFG_VS2 : 2;
    UCHAR CFG_VS1 : 2;
// PAC register LSB mapped into struct MSB
    UCHAR CFG_VB4 : 2;
    UCHAR CFG_VB3 : 2;
    UCHAR CFG_VB2 : 2;
    UCHAR CFG_VB1 : 2;
} PAC194X5X_NEG_PWR_FSR_REGISTER, *PPAC194X5X_NEG_PWR_FSR_REGISTER;


typedef struct _PAC194X5X_SLOW_REGISTER {
```

```
    UCHAR               : 1;
    UCHAR RefreshVFall  : 1;
    UCHAR RefreshFall   : 1;
    UCHAR RefreshVRise  : 1;
    UCHAR RefreshRise   : 1;
    UCHAR SlowHighLow   : 1;
    UCHAR SlowLowHigh   : 1;
    UCHAR Slow          : 1;
} PAC194X5X_SLOW_REGISTER, *PPAC194X5X_SLOW_REGISTER;


typedef struct _PAC194X5X_ALERT_STATUS_REGISTER {
// PAC register MSB mapped into struct LSB
    UCHAR CH4_UC    : 1;
    UCHAR CH3_UC    : 1;
    UCHAR CH2_UC    : 1;
    UCHAR CH1_UC    : 1;
    UCHAR CH4_OC    : 1;
    UCHAR CH3_OC    : 1;
    UCHAR CH2_OC    : 1;
    UCHAR CH1_OC    : 1;
// PAC register middle byte mapped into struct middle byte
    UCHAR CH4_UV    : 1;
    UCHAR CH3_UV    : 1;
    UCHAR CH2_UV    : 1;
    UCHAR CH1_UV    : 1;
    UCHAR CH4_OV    : 1;
    UCHAR CH3_OV    : 1;
    UCHAR CH2_OV    : 1;
    UCHAR CH1_OV    : 1;
// PAC register LSB mapped into struct MSB
    UCHAR           : 2;
    UCHAR ACC_COUNT : 1;
    UCHAR ACC_OVF   : 1;
    UCHAR CH4_OP    : 1;
    UCHAR CH3_OP    : 1;
    UCHAR CH2_OP    : 1;
    UCHAR CH1_OP    : 1;
} PAC194X5X_ALERT_STATUS_REGISTER, *PPAC194X5X_ALERT_STATUS_REGISTER;


typedef struct _PAC194X5X_ALERT_ENABLE_REGISTER {
    // PAC register MSB mapped into struct LSB
    UCHAR CH4_UC    : 1;
    UCHAR CH3_UC    : 1;
    UCHAR CH2_UC    : 1;
    UCHAR CH1_UC    : 1;
    UCHAR CH4_OC    : 1;
    UCHAR CH3_OC    : 1;
    UCHAR CH2_OC    : 1;
    UCHAR CH1_OC    : 1;
    // PAC register middle byte mapped into struct middle byte
    UCHAR CH4_UV    : 1;
    UCHAR CH3_UV    : 1;
    UCHAR CH2_UV    : 1;
    UCHAR CH1_UV    : 1;
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
    UCHAR CH4_OV    : 1;
    UCHAR CH3_OV    : 1;
    UCHAR CH2_OV    : 1;
    UCHAR CH1_OV    : 1;
    // PAC register LSB mapped into struct MSB
    UCHAR           : 1;
    UCHAR ALERT_CC  : 1;
    UCHAR ACC_COUNT : 1;
    UCHAR ACC_OVF   : 1;
    UCHAR CH4_OP    : 1;
    UCHAR CH3_OP    : 1;
    UCHAR CH2_OP    : 1;
    UCHAR CH1_OP    : 1;
} PAC194X5X_ALERT_ENABLE_REGISTER, *PPAC194X5X_ALERT_ENABLE_REGISTER;


typedef struct _PAC194X5X_GPIO_ALERT2_REGISTER {
    // PAC register MSB mapped into struct LSB
    UCHAR CH4_UC    : 1;
    UCHAR CH3_UC    : 1;
    UCHAR CH2_UC    : 1;
    UCHAR CH1_UC    : 1;
    UCHAR CH4_OC    : 1;
    UCHAR CH3_OC    : 1;
    UCHAR CH2_OC    : 1;
    UCHAR CH1_OC    : 1;
    // PAC register middle byte mapped into struct middle byte
    UCHAR CH4_UV    : 1;
    UCHAR CH3_UV    : 1;
    UCHAR CH2_UV    : 1;
    UCHAR CH1_UV    : 1;
    UCHAR CH4_OV    : 1;
    UCHAR CH3_OV    : 1;
    UCHAR CH2_OV    : 1;
    UCHAR CH1_OV    : 1;
    // PAC register LSB mapped into struct MSB
    UCHAR           : 1;
    UCHAR ALERT_CC2 : 1;
    UCHAR ACC_COUNT : 1;
    UCHAR ACC_OVF   : 1;
    UCHAR CH4_OP    : 1;
    UCHAR CH3_OP    : 1;
    UCHAR CH2_OP    : 1;
    UCHAR CH1_OP    : 1;
} PAC194X5X_GPIO_ALERT2_REGISTER, *PPAC194X5X_GPIO_ALERT2_REGISTER;


typedef struct _PAC194X5X_LIMIT_NSAMPLES_REGISTER {
    UCHAR Nsamples_CH4 : 2;
    UCHAR Nsamples_CH3 : 2;
    UCHAR Nsamples_CH2 : 2;
    UCHAR Nsamples_CH1 : 2;
} PAC194X5X_LIMIT_NSAMPLES_REGISTER, *PPAC194X5X_LIMIT_NSAMPLES_REGISTER;


typedef struct _PAC194X5X_ACCUM_LIMITS_REGISTER {
    // PAC register MSB mapped into struct LSB
    UCHAR ACC_CH4   : 2;
    UCHAR ACC_CH3   : 2;
    UCHAR ACC_CH2   : 2;
    UCHAR ACC_CH1   : 2;
    // PAC register LSB mapped into struct MSB
    UCHAR           : 6;
    UCHAR ACC_COUNT : 2;
} PAC194X5X_ACCUM_LIMITS_REGISTER, *PPAC194X5X_ACCUM_LIMITS_REGISTER;

#pragma pop(pack)
```

EXAMPLE C-2: PAC194X5X_INTF.h

```
////////////////////////////////////////////////////////////////////////////////
//
//      (C) 2019-2021 Microchip Technology Inc. and its subsidiaries.
//      (C) Copyright 2016-2019 OSR Open Systems Resources, Inc.
//
//      Subject to your compliance with these terms, you may use Microchip software
//      and any derivatives exclusively with Microchip products. You're responsible
//      for complying with 3rd party license terms applicable to your use of 3rd party
//      software (including open source software) that may accompany Microchip software.
//      SOFTWARE IS "AS IS." NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY,
//      APPLY TO THIS SOFTWARE, INCLUDING ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT,
//      MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. 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 SOFTWARE, 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 RELATED TO THE SOFTWARE WILL NOT EXCEED AMOUNT OF FEES, IF ANY,
//      YOU PAID DIRECTLY TO MICROCHIP FOR THIS SOFTWARE.
//
//
//    MODULE:
//       PAC194x5x_INTF.h
//
//    ABSTRACT:
//       Device-Specific User Interface and IOCTL definitions for
//       the Microchip PAC194x_5x EMI Driver
//
//    AUTHOR(S):
//        OSR Open Systems Resources, Inc.
//        Microchip Technology, Inc.
//
//    REVISION:
//        v1.2.0
//
////////////////////////////////////////////////////////////////////////////////

#pragma once

#include <initguid.h>
#include <emi.h>


// {2DF42D48-4C09-453A-A05F-8A48F8FBB415}
// Identifies the control (non-EMI) portion of the interface
#define GUID_PAC194X5X_CONTROL_INTERFACE_STR L"{2DF42D48-4C09-453A-A05F-8A48F8FBB415}"
DEFINE_GUID(GUID_PAC194X5X_CONTROL_INTERFACE,
            0x2df42d48, 0x4c09, 0x453a, 0xa0, 0x5f, 0x8a, 0x48, 0xf8, 0xfb, 0xb4, 0x15);
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
//
// Register definitions
//
#include "..\Inc\PAC194x5x_HW.h"

//
// IOCTL definitions
//

#define FILE_DEVICE_PAC194X5X 0x9145 //IOCTL "Device Type" must be > 0x8000 for vendor defined devices
                                     //IOCTL "Function Code" must be > 0x800 for vendor defined functions

//info functions
#define FNUMBER_PAC194X5X_GET_INTERFACE_REVISION       0x800  //IOCTL_PAC194X5X_GET_INTERFACE_REVISION
#define FNUMBER_PAC194X5X_GET_DEVICE_INFO_SIZE         0x801  //IOCTL_PAC194X5X_GET_DEVICE_INFO_SIZE
#define FNUMBER_PAC194X5X_GET_DEVICE_INFO              0x802  //IOCTL_PAC194X5X_GET_DEVICE_INFO
#define FNUMBER_PAC194X5X_GET_RSENSE_MICRO             0x803  //IOCTL_PAC194X5X_GET_RSENSE_MICRO

//data functions
#define FNUMBER_PAC194X5X_DATA_REFRESH                 0x810  //IOCTL_PAC194X5X_DATA_REFRESH
#define FNUMBER_PAC194X5X_DATA_REFRESH_RESET           0x811  //IOCTL_PAC194X5X_DATA_REFRESH_RESET
#define FNUMBER_PAC194X5X_DATA_REFRESH_RESET_GLOBAL    0x812
//IOCTL_PAC194X5X_DATA_REFRESH_RESET_GLOBAL
#define FNUMBER_PAC194X5X_GET_DATA                     0x813  //IOCTL_PAC194X5X_GET_MEASUREMENTS
#define FNUMBER_PAC194X5X_GET_LAST_SLOW_DATA           0x814  //IOCTL_PAC194X5X_GET_LAST_SLOW_DATA

//status functions
#define FNUMBER_PAC194X5X_GET_OVERFLOW                 0x820  //IOCTL_PAC194X5X_GET_OVERFLOW

//config functions
#define FNUMBER_PAC194X5X_GET_CTRL                     0x830  //IOCTL_PAC194X5X_GET_CTRL
#define FNUMBER_PAC194X5X_SET_CTRL                     0x831  //IOCTL_PAC194X5X_SET_CTRL
#define FNUMBER_PAC194X5X_GET_CHANNEL_OFF              0x832  //IOCTL_PAC194X5X_GET_CHANNEL_CFG
#define FNUMBER_PAC194X5X_SET_CHANNEL_OFF              0x833  //IOCTL_PAC194X5X_SET_CHANNEL_CFG
#define FNUMBER_PAC194X5X_SET_SAMPLE_MODE              0x834  //IOCTL_PAC194X5X_SET_SAMPLE_MODE
#define FNUMBER_PAC194X5X_SET_GPIO2_MODE               0x835  //IOCTL_PAC194X5X_SET_GPIO2_MODE

#define FNUMBER_PAC194X5X_GET_SMBUS_SETTINGS           0x836  //IOCTL_PAC194X5X_GET_SMBUS_SETTINGS
#define FNUMBER_PAC194X5X_GET_ANY_ALERT                0x837  //IOCTL_PAC194X5X_GET_ANY_ALERT
#define FNUMBER_PAC194X5X_GET_GPIO_DATA2               0x838  //IOCTL_PAC194X5X_GET_GPIO_DATA2
#define FNUMBER_PAC194X5X_SET_GPIO_DATA2               0x839  //IOCTL_PAC194X5X_SET_GPIO_DATA2

#define FNUMBER_PAC194X5X_GET_NEG_POWER                0x83A  //IOCTL_PAC194X5X_GET_NEG_POWER
#define FNUMBER_PAC194X5X_SET_NEG_POWER                0x83B  //IOCTL_PAC194X5X_SET_NEG_POWER

#define FNUMBER_PAC194X5X_GET_SLOW                     0x83C  //IOCTL_PAC194X5X_GET_SLOW

#define FNUMBER_PAC194X5X_GET_ACCUM_CONFIG             0x83E  //IOCTL_PAC194X5X_GET_ACCUM_CONFIG
#define FNUMBER_PAC194X5X_SET_ACCUM_CONFIG             0x83F  //IOCTL_PAC194X5X_GET_ACCUM_CONFIG

#define FNUMBER_PAC194X5X_GET_ALERT_STATUS             0x840  //IOCTL_PAC194X5X_GET_ALERT_STATUS
#define FNUMBER_PAC194X5X_GET_ALERT_ENABLE             0x841  //IOCTL_PAC194X5X_GET_ALERT_ENABLE
#define FNUMBER_PAC194X5X_SET_ALERT_ENABLE             0x842  //IOCTL_PAC194X5X_SET_ALERT_ENABLE
#define FNUMBER_PAC194X5X_GET_GPIO_ALERT2              0x843  //IOCTL_PAC194X5X_GET_GPIO_ALERT2
#define FNUMBER_PAC194X5X_SET_GPIO_ALERT2              0x844  //IOCTL_PAC194X5X_SET_GPIO_ALERT2

#define FNUMBER_PAC194X5X_GET_OC_LIMIT                 0x850  //IOCTL_PAC194X5X_GET_OC_LIMIT
#define FNUMBER_PAC194X5X_SET_OC_LIMIT                 0x851  //IOCTL_PAC194X5X_SET_OC_LIMIT
#define FNUMBER_PAC194X5X_GET_UC_LIMIT                 0x852  //IOCTL_PAC194X5X_GET_UC_LIMIT
#define FNUMBER_PAC194X5X_SET_UC_LIMIT                 0x853  //IOCTL_PAC194X5X_SET_UC_LIMIT
#define FNUMBER_PAC194X5X_GET_OP_LIMIT                 0x854  //IOCTL_PAC194X5X_GET_OP_LIMIT
#define FNUMBER_PAC194X5X_SET_OP_LIMIT                 0x855  //IOCTL_PAC194X5X_SET_OP_LIMIT
#define FNUMBER_PAC194X5X_GET_OV_LIMIT                 0x856  //IOCTL_PAC194X5X_GET_OV_LIMIT
#define FNUMBER_PAC194X5X_SET_OV_LIMIT                 0x857  //IOCTL_PAC194X5X_SET_OV_LIMIT
#define FNUMBER_PAC194X5X_GET_UV_LIMIT                 0x858  //IOCTL_PAC194X5X_GET_UV_LIMIT
#define FNUMBER_PAC194X5X_SET_UV_LIMIT                 0x859  //IOCTL_PAC194X5X_SET_UV_LIMIT
#define FNUMBER_PAC194X5X_GET_LIMIT_NSAMPLES           0x85A  //IOCTL_PAC194X5X_GET_LIMIT_NSAMPLES
#define FNUMBER_PAC194X5X_SET_LIMIT_NSAMPLES           0x85B  //IOCTL_PAC194X5X_SET_LIMIT_NSAMPLES
#define FNUMBER_PAC194X5X_GET_ACCUM_LIMITS             0x85C  //IOCTL_PAC194X5X_GET_ACCUM_LIMITS
#define FNUMBER_PAC194X5X_SET_ACCUM_LIMITS             0x85D  //IOCTL_PAC194X5X_SET_ACCUM_LIMITS
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_INTERFACE_REVISION
//
// Returns the driver private interface revision number.
//
// - input buffer: none
// - output buffer: PAC194X5X_GET_INTF_REV
////////////////////////////////////////////////////////////////////////////////////////////////
#define PAC_IOCTL_INTERFACE_REVISION_01 0x01
typedef struct _PAC194X5X_GET_INTF_REV {
    USHORT PrivIntfRev;     // private interface revision value
} PAC194X5X_GET_INTF_REV, *PPAC194X5X_GET_INTF_REV;

#define IOCTL_PAC194X5X_GET_INTERFACE_REVISION  CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                         FNUMBER_PAC194X5X_GET_INTERFACE_REVISION,\
                                                         METHOD_BUFFERED,\
                                                         FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_DEVICE_INFO_SIZE
//
// Returns the size allocation requirements for the buffer that must hold
// the device info data structure returned by IOCTL_PAC194X5X_GET_DEVICE_INFO.
// - input buffer: none
// - output buffer: PAC194X5X_GET_DEVICE_INFO_SIZE
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_DEVICE_INFO_SIZE {
    ULONG BufferSize;
}PAC194X5X_GET_DEVICE_INFO_SIZE, *PPAC194X5X_GET_DEVICE_INFO_SIZE;

#define IOCTL_PAC194X5X_GET_DEVICE_INFO_SIZE    CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                         FNUMBER_PAC194X5X_GET_DEVICE_INFO_SIZE,\
                                                         METHOD_BUFFERED,\
                                                         FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_DEVICE_INFO
//
// Returns information about the device version and about the channel
// connections (rail names, sense resistors, connection status).
// - input buffer: none
// - output buffer: PAC194X5X_GET_DEVICE_INFO
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_DEVICE_CHANNEL_INFO {
    ULONG NextChannelInfoOffset;

    BOOLEAN ChannelIsOFF;               // TRUE if the channel is disabled ->
                                        // CHn_OFF bit is set in CTRL_ACT register

    BOOLEAN ChannelHasEMI;              // TRUE if EMI is created
                                        // (the channel is named)

    ULONG RsenseUOhm;                   // Rsense in micro-ohms
                                        // 0 indicates non-connected channel

    ULONG VbusKfactor;                  // Vbus multiplication factor
                                        // K = 1000 * (Vrail / Vbus)

    WCHAR ChannelName[ANYSIZE_ARRAY];   // EMI channel name.
                                        // If the string is empty ("),
                                        // EMI is not created for this channel
                                        // reported also as
                                        // EMI_METADATA.MeteredHardwareName
} PAC194X5X_DEVICE_CHANNEL_INFO, *PPAC194X5X_DEVICE_CHANNEL_INFO;
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
typedef struct _PAC194X5X_GET_DEVICE_INFO {
    UCHAR ProductId;                    // PRODUCT ID register value
    UCHAR ManufacturerId;               // MANUFACTURER ID register value
    UCHAR ProductRevision;              // REVISION ID register value
                                        // reported also as EMI_METADATA.HardwareRevision

    ULONG ChannelCount;                 // the number of "connected" channels (Rsense > 0)

    PAC194X5X_DEVICE_CHANNEL_INFO ChannelInfo;  // there are 1, 2, 3 or 4
                                                // "CHANNEL_INFO" structures
                                                // depending on device type:
                                                // PAC1941/51, PAC1942/52, PAC1943/53, PAC1944/54
} PAC194X5X_GET_DEVICE_INFO, *PPAC194X5X_GET_DEVICE_INFO;

#define IOCTL_PAC194X5X_GET_DEVICE_INFO     CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                     FNUMBER_PAC194X5X_GET_DEVICE_INFO,\
                                                     METHOD_BUFFERED,\
                                                     FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_RSENSE_MICRO
//
// Returns the sense resistor values for all channels as micro-ohms values.
// NOTE: returns 0 for the un-connected sense resistors and for the channels
// which are not available on the devices with less than four channels.
//
// - input buffer: none
// - output buffer: PAC194X5X_GET_RSENSE_MICRO
////////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_RSENSE_MICRO {
    ULONG RsenseCH1;    //micro-ohms
    ULONG RsenseCH2;    //micro-ohms
    ULONG RsenseCH3;    //micro-ohms
    ULONG RsenseCH4;    //micro-ohms
} PAC194X5X_GET_RSENSE_MICRO, *PPAC194X5X_GET_RSENSE_MICRO;

#define IOCTL_PAC194X5X_GET_RSENSE_MICRO      CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                       FNUMBER_PAC194X5X_GET_RSENSE_MICRO,\
                                                       METHOD_BUFFERED,\
                                                       FILE_READ_DATA)
////////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_DATA_REFRESH
//
// Performs the update of the software accumulators using the REFRESH_V
// device command.
// - input buffer: none
// - output buffer: none (zero bytes returned)
////////////////////////////////////////////////////////////////////////////////////////////////////
#define IOCTL_PAC194X5X_DATA_REFRESH         CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                      FNUMBER_PAC194X5X_DATA_REFRESH,\
                                                      METHOD_BUFFERED,\
                                                      FILE_READ_DATA)
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_DATA_REFRESH_RESET
//
// Performs the update of the software accumulators using the REFRESH device
// command. Depending on the value of the input flag, this request can also clear
// (reset to 0) the software accumulators for the channels without open EMI
// interfaces.
// - input buffer: PAC194X5X_REFRESH_FLAGS
// - output buffer: none (zero bytes returned)
////////////////////////////////////////////////////////////////////////////////////////////////
#define PAC194X5X_REFRESH_FLAG_RESET_ACCUMULATORS (1 << 0)
typedef struct _PAC194X5X_REFRESH_FLAGS {
    ULONG Flags;    // In: the user set the value of this field to
                    // PAC194X5X_REFRESH_FLAG_RESET_ACCUMULATORS in order to
                    // request the driver to clear the software accumulators
                    // - SampleCountTotal
                    // - EnergyEMI[PAC_MAXIMUM_CHANNELS]
                    // - EnergyTotal[PAC_MAXIMUM_CHANNELS]
                    //
                    // CAUTION: EnergyEMI and EnergyTotal
                    //          for the EMI channels are not reset
} PAC194X5X_REFRESH_FLAGS, *PPAC194X5X_REFRESH_FLAGS;

#define IOCTL_PAC194X5X_DATA_REFRESH_RESET      CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                         FNUMBER_PAC194X5X_DATA_REFRESH_RESET,\
                                                         METHOD_BUFFERED,\
                                                         FILE_READ_DATA | FILE_WRITE_DATA)
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_DATA_REFRESH_RESET_GLOBAL
//
// Performs a global update of the software accumulators for all the PAC194x/5x
// devices in the system. Depending on the value of the input flag, this request
// can also clear (reset to 0) the software accumulators for the channels without
// open EMI interfaces.
// - input buffer: PAC194X5X_REFRESH_FLAGS
// - output buffer: none (zero bytes returned)
//
// NOTE: because Windows SpbCx framework does not support the I2C general call
// address (0x00), the driver cannot implement this request using REFRESH_G
// device commands. Instead, the driver implements this request sending
// sequential REFRESH commands to all the devices in the system.
////////////////////////////////////////////////////////////////////////////////////////////////
#define IOCTL_PAC194X5X_DATA_REFRESH_RESET_GLOBAL  CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                         FNUMBER_PAC194X5X_DATA_REFRESH_RESET_GLOBAL,\
                                                            METHOD_BUFFERED,\
                                                            FILE_READ_DATA | FILE_WRITE_DATA)
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_DATA
//
// Returns the measured values (voltage, power, accumulated power, sample count
// and timestamp.
// - input buffer: none
// - output buffer: PAC194X5X_DATA
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_DATA {
    // Measurement device registers section
    ULONG     AccumulatorCount;                          // ACC_COUNT register value
    LONGLONG  AccumulatorOutput[PAC_MAXIMUM_CHANNELS];    // VACCn registers, sign extended to 64bit
    USHORT    VBus[PAC_MAXIMUM_CHANNELS];                 // VBUSn register values
    USHORT    VBusAvg[PAC_MAXIMUM_CHANNELS];              // VBUSn_AVG registers values
    USHORT    VSense[PAC_MAXIMUM_CHANNELS];               // VSENSEn registers values
    USHORT    VSenseAvg[PAC_MAXIMUM_CHANNELS];            // VSENSEn_AVG registers values
    ULONG     VPower[PAC_MAXIMUM_CHANNELS];               // VPOWERn registers values

    // CAUTION: If the bidirectional voltage and/or current is enabled for
    //          certain channels the binary format of the reported values is in
    //          2's complement (the msb is a sign bit).
    //          So, please make sure to cast the reported value to the necessary
    //          data type for the subsequent processing.

    // Multiplication factors (units per least significan bit - lsb)
    LONGLONG  VbusLSB_pV[PAC_MAXIMUM_CHANNELS];    // Vbus lsb value in pico-Volts.
                                                   // Multiplied with VBUSn or VACCn register (in
                                                   // Vbus count mode) gives the real Voltage value.
    LONGLONG  VsenseLSB_pV[PAC_MAXIMUM_CHANNELS];  // Vsense lsb value in pico-Volts.
                                                   // Multiplied with VSENSEn or VACCn register (in
                                                   // Vsense count mode) gives the real Voltage value.
    LONGLONG  VPowerLSB_pW[PAC_MAXIMUM_CHANNELS];  // VPower lsb value in pico-Watts.
                                                   // Multiplied with VPOWERn or VACCn register (in
                                                   // Vpower count mode) gives the real Watts value.
    LONGLONG  VRail_pV[PAC_MAXIMUM_CHANNELS];      // Rail voltage in pico-Volts
    LONGLONG  VRailAvg_pV[PAC_MAXIMUM_CHANNELS];   // Average rail voltage in pico-Volts
    LONGLONG  VSense_pV[PAC_MAXIMUM_CHANNELS];     // Sense resistor voltage drop in pico-Volts
    LONGLONG  VSenseAvg_pV[PAC_MAXIMUM_CHANNELS];  // Average sense resistor voltage drop in pico-Volts
    LONGLONG  ISense_pA[PAC_MAXIMUM_CHANNELS];     // Rail amperage in pico-Amps
    LONGLONG  ISenseAvg_pA[PAC_MAXIMUM_CHANNELS];  // Average rail amperage in pico-Amps
    LONGLONG  Power_pW[PAC_MAXIMUM_CHANNELS];      // Rail power in pico-Watts

    // Configuration device registers section
    PAC194X5X_NEG_PWR_FSR_REGISTER   NegPowerLat;         // NEG_PWR_FSR_LAT register value
    PAC194X5X_CTRL_REGISTER          CtrlLat;             // CTRL_LAT register value
    PAC194X5X_ACCUM_CONFIG_REGISTER  AccumConfigLat;      // ACCUM_CONFIG_LAT register value

    // Software accumulators section
    // The driver accumulates in software accumulators the values reported by
    // devices and clears the device accumulators in order to avoid the
    // devices accumulators saturation
    ULONGLONG  SampleCountTotal;                  // Software sample counter

    ULONGLONG  EnergyEMI[PAC_MAXIMUM_CHANNELS];   // Energy accumulator, as defined by EMI.
                                                  // If the channel is configured as bi-polar, only
                                                  // the positive energy increments are accumulated.
                                                  // The energy is reported in pico-Watt-hours (pWh).
                                                  // Same values are also reported by
                                                  // EMI_MEASUREMENT_DATA.AbsoluteEnergy.
    LONGLONG  EnergyTotal[PAC_MAXIMUM_CHANNELS];  // Signed energy accumulator.
                                                  // Both positive and negative energy values are
                                                  // accumulated. Msb is the sign bit.
                                                  // The energy is reported in pico-Watt-hours (pWh).
```

```
    LONGLONG  CoulombCount[PAC_MAXIMUM_CHANNELS]; // Coulomb software accumulator.
                                                  // Reported in pico-Amp-seconds (pAs).

    LONGLONG  VRailAccumulator[PAC_MAXIMUM_CHANNELS]; // VRail software accumulator.
                                                      // Reported in pico-Volts (pV).

    ULONGLONG  Timestamp;  // The timestamp of the last measurements.
                           // Reported as a count of 100 - nanosecond units since January 1st, 1601
                           // The timestamp is created by the driver by reading the
                           // system performance counter (high resolution timer, <1us)
                           // Same values are also reported by EMI_MEASUREMENT_DATA.AbsoluteTime
}PAC194X5X_DATA, *PPAC194X5X_DATA;

#define IOCTL_PAC194X5X_GET_DATA            CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                     FNUMBER_PAC194X5X_GET_DATA,\
                                                     METHOD_BUFFERED,\
                                                     FILE_READ_DATA)


//////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_LAST_SLOW_DATA
//
// Returns for all the channels all the data measurements sampled by the last
// SLOW signal HL transition (SLOW triggered "limited" refresh) caused by the
// system state transition to S0 from other non-powered state.
// - input buffer: none
// - output buffer: PAC194X5X_LAST_SLOW_DATA
//////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_LAST_SLOW_DATA {
    ULONG     AccumulatorCount;                          // ACC_COUNT register value
    LONGLONG  AccumulatorOutput[PAC_MAXIMUM_CHANNELS];   // VACCn registers, sign extended to 64bit
    USHORT    VBus[PAC_MAXIMUM_CHANNELS];                // VBUSn register values
    USHORT    VBusAvg[PAC_MAXIMUM_CHANNELS];             // VBUSn_AVG registers values
    USHORT    VSense[PAC_MAXIMUM_CHANNELS];              // VSENSEn registers values
    USHORT    VSenseAvg[PAC_MAXIMUM_CHANNELS];           // VBUSn_AVG registers values
    ULONG     VPower[PAC_MAXIMUM_CHANNELS];              // VPOWERn registers values
    // CAUTION: If the bidirectional voltage and/or current is enabled for
    //          certain channels the binary format of the reported values is in
    //          2's complement (the msb is a sign bit). So, please make sure
    //          to cast the reported value to the necessary data type for the
    //          subsequent processing.
    //

    // Multiplication factors (units per least significant bit - lsb)
    LONGLONG  VbusLSB_pV[PAC_MAXIMUM_CHANNELS];    // Vbus lsb value in pico-Volts.
                                                   // Multiplied with VBUSn or VACCn register (in
                                                  // Vbus count mode) provides the real Voltage value.
    LONGLONG  VsenseLSB_pV[PAC_MAXIMUM_CHANNELS];  // Vsense lsb value in pico-Volts.
                                                   // Multiplied with VSENSEn or VACCn register (in
                                                // Vsense count mode) provides the real Voltage value.
    LONGLONG  VPowerLSB_pW[PAC_MAXIMUM_CHANNELS];  // VPower lsb value in pico-Watts.
                                                   // Multiplied with VPOWERn or VACCn register (in
                                                  // Vpower count mode) provides the real Watts value.

    LONGLONG  VRail_pV[PAC_MAXIMUM_CHANNELS];      // Rail voltage in pico-Volts
    LONGLONG  VRailAvg_pV[PAC_MAXIMUM_CHANNELS];   // Average rail voltage in pico-Volts
    LONGLONG  VSense_pV[PAC_MAXIMUM_CHANNELS];     // Sense resistor voltage drop in pico-Volts
    LONGLONG  VSenseAvg_pV[PAC_MAXIMUM_CHANNELS];  // Average sense resistor voltage drop in pico-Volts
    LONGLONG  ISense_pA[PAC_MAXIMUM_CHANNELS];     // Rail amperage in pico-Amps
    LONGLONG  ISenseAvg_pA[PAC_MAXIMUM_CHANNELS];  // Average rail amperage in pico-Amps
    LONGLONG  Power_pW[PAC_MAXIMUM_CHANNELS];      // Rail power in pico-Watts
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
// Configuration device registers section
    PAC194X5X_NEG_PWR_FSR_REGISTER     NegPowerLat;     // Channels' polarity and FSR settings.
    PAC194X5X_CTRL_REGISTER            CtrlLat;         // Sampling mode, GPIO configuration and
                                                        // channels' ON/OFF status.
    PAC194X5X_ACCUM_CONFIG_REGISTER    AccumConfigLat;  // Channels' accumulator configuration.
}PAC194X5X_LAST_SLOW_DATA, *PPAC194X5X_LAST_SLOW_DATA;

#define IOCTL_PAC194X5X_GET_LAST_SLOW_DATA      CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                         FNUMBER_PAC194X5X_GET_LAST_SLOW_DATA,\
                                                         METHOD_BUFFERED,\
                                                         FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_OVERFLOW
//
// Returns the status of the ACC_OVF and ACC_COUNT bits from ALERT STATUS
// Depending on the value of the input flag, this request can also clear
// (reset to 0) the ACC_OVF and ACC_COUNT flags.
// - input buffer: PAC194X5X_GET_OVERFLOW
// - output buffer: PAC194X5X_GET_OVERFLOW
////////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_OVERFLOW {
    UCHAR OvfClear;     // In: the user set nonzero value to request the driver
                        // to clear the ACC_OVF and ACC_COUNT flags

    UCHAR AccOvf;       // Out: the IOCTL_PAC194X5X_GET_OVERFLOW driver returns in
                        // the lsb the value of the ACC_OVF bit from ALERT_STATUS

    UCHAR CountOvf;     // Out: the IOCTL_PAC194X5X_GET_OVERFLOW driver returns in
                        // the lsb the value of the ACC_COUNT bit from ALERT_STATUS
} PAC194X5X_GET_OVERFLOW, *PPAC194X5X_GET_OVERFLOW;

#define IOCTL_PAC194X5X_GET_OVERFLOW      CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                   FNUMBER_PAC194X5X_GET_OVERFLOW,\
                                                   METHOD_BUFFERED,\
                                                   FILE_READ_DATA | FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_CTRL
//
// Returns the content of the device registers CTRL, CTRL_ACT, CTRL_LAT
// - input buffer: none
// - output buffer: PAC194X5X_GET_CTRL
////////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_CTRL {
    PAC194X5X_CTRL_REGISTER ControlReg;           // CTRL register value
    PAC194X5X_CTRL_REGISTER ControlAct;           // CTRL_ACT register value
    PAC194X5X_CTRL_REGISTER ControlLat;           // CTRL_LAT register value
} PAC194X5X_GET_CTRL, *PPAC194X5X_GET_CTRL;
#define IOCTL_PAC194X5X_GET_CTRL            CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                     FNUMBER_PAC194X5X_GET_CTRL,\
                                                     METHOD_BUFFERED,\
                                                     FILE_READ_DATA)
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_CTRL
//
// Provides limited user control over some device configuration bits from
// CTRL register: SAMPLE_MODE, CHx_OFF, GPIO_ALERT2.
// The driver rejects the "reserved" or unsupported SAMPLE_MODE values.
// The driver ignores the CHx_OFF configuration requests which may affect
// channels with open EMI interface.
// The driver activates the new device configuration using a REFRESH command.
// NOTES:
//      - Change requests for the CHx_OFF bits of the channels with EMI interfaces
//        are ignored.
//      - Change requests for SLOW_ALERT1 are ignored
//      - The "Reserved" or the unsupported values for SAMPLE_MODE bits are rejected,
//        with "invalid parameters" error code STATUS_INVALID_PARAMETER.
//      - SAMPLE_MODE setting to non-adaptive accumulation mode is not supported
//        together with GPIO_ALERT2 setting as SLOW pin. The change requests are
//        rejected (STATUS_INVALID_PARAMETER error).
// - input buffer: PAC194X5X_SET_CTRL
// - output buffer: none (zero bytes returned)
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_SET_CTRL {
    PAC194X5X_CTRL_REGISTER ControlReg;     // value to be written in
                                            // CTRL register
} PAC194X5X_SET_CTRL, *PPAC194X5X_SET_CTRL;

#define IOCTL_PAC194X5X_SET_CTRL            CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                     FNUMBER_PAC194X5X_SET_CTRL,\
                                                     METHOD_BUFFERED,\
                                                     FILE_WRITE_DATA)
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_CHANNEL_OFF
//
// Returns the content of the channel ON/OFF configuration bits from
// CTRL, CTRL_ACT and CTRL_LAT registers
// - input buffer: none
// - output buffer: PAC194X5X_GET_CHANNEL_OFF
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_CHANNEL_OFF {
    UCHAR CH1_OFF;                          // 0 = ON, 1 = OFF
    UCHAR CH2_OFF;                          // 0 = ON, 1 = OFF
    UCHAR CH3_OFF;                          // 0 = ON, 1 = OFF
    UCHAR CH4_OFF;                          // 0 = ON, 1 = OFF
} PAC194X5X_CHANNEL_OFF;

typedef struct _PAC194X5X_GET_CHANNEL_OFF {
    PAC194X5X_CHANNEL_OFF ConfigOFF;        // CHn_OFF config in CTRL register
    PAC194X5X_CHANNEL_OFF ConfigOFFAct;     // CHn_OFF config in CTRL_ACT register
    PAC194X5X_CHANNEL_OFF ConfigOFFLat;     // CHn_OFF config in CTRL_LAT register
} PAC194X5X_GET_CHANNEL_OFF, *PPAC194X5X_GET_CHANNEL_OFF;

#define IOCTL_PAC194X5X_GET_CHANNEL_OFF     CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                     FNUMBER_PAC194X5X_GET_CHANNEL_OFF,\
                                                     METHOD_BUFFERED,\
                                                     FILE_READ_DATA)
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_CHANNEL_OFF
//
// Provides limited user control over CHn_OFF bits from CTRL register.
// The driver activates the new device configuration using a REFRESH command.
// Notes:
//  - The only accepted values are "0" (ON) and "1" (OFF) for each CHx_OFF
//    control structure field otherwise the request is rejected (STATUS_INVALID_PARAMETER error).
//  - The driver ignores the change values for those channels which has
//    open EMI channels.
//
// - input buffer: PAC194X5X_CHANNEL_OFF
// - output buffer: none (zero bytes returned)
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_SET_CHANNEL_OFF {
    PAC194X5X_CHANNEL_OFF ConfigOFF;
} PAC194X5X_SET_CHANNEL_OFF, *PPAC194X5X_SET_CHANNEL_OFF;

#define IOCTL_PAC194X5X_SET_CHANNEL_OFF      CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                      FNUMBER_PAC194X5X_SET_CHANNEL_OFF,\
                                                      METHOD_BUFFERED,\
                                                      FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_SAMPLE_MODE
//
// Provides limited user control over device configuration bits SAMPLE_MODE
// from CTRL register.
// The driver activates the new device configuration using a REFRESH command.
// NOTES:
//      - The unsupported and the "Reserved" values for SAMPLE_MODE bits
//      are rejected as "Invalid Parameter" failure.
//      - Any of the "non-adaptive" SAMPLE_MODE settings requests are rejected
//      if GPIO_ALERT2 is already set as SLOW pin.
//      - "SINGLE SHOT", "SINGLE SHOT8X", and SLEEP are rejected if EMI handles
//       are open on this device
// - input buffer: PAC194X5X_SET_SAMPLE_MODE
// - output buffer: none (zero bytes returned)
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_SET_SAMPLE_MODE {
    UCHAR SampleMode;                       // value to be written in
                                            // CTRL SAMPLE_MODE bits
} PAC194X5X_SET_SAMPLE_MODE, *PPAC194X5X_SET_SAMPLE_MODE;

#define IOCTL_PAC194X5X_SET_SAMPLE_MODE     CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                     FNUMBER_PAC194X5X_SET_SAMPLE_MODE,\
                                                     METHOD_BUFFERED,\
                                                     FILE_WRITE_DATA)
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_GPIO2_MODE
//
// Provides user control over device configuration bits GPIO_ALERT2 from CTRL register.
// The driver activates the new device configuration using a REFRESH_V command.
// NOTES:
//      - The "SLOW" configuration value is rejected if SAMPLE_MODE is not set
//      to an adaptive accumulation mode.
//      - The unsupported input config values are rejected also.
// - input buffer: PAC194X5X_SET_GPIO2_MODE
// - output buffer: none (zero bytes returned)
////////////////////////////////////////////////////////////////////////////////////////////////
```

```
typedef struct _PAC194X5X_SET_GPIO2_MODE {
    UCHAR Gpio2Mode;                        // value to be written in
                                            // CTRL GPIO_ALERT2 bits
} PAC194X5X_SET_GPIO2_MODE, *PPAC194X5X_SET_GPIO2_MODE;

#define IOCTL_PAC194X5X_SET_GPIO2_MODE     CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                    FNUMBER_PAC194X5X_SET_GPIO2_MODE,\
                                                    METHOD_BUFFERED,\
                                                    FILE_WRITE_DATA)
////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_SMBUS_SETTINGS
//
// Returns the value of the SMBUS_SETTINGS register.
// Note: the configuration of the SMBUS_SETTINGS register is controlled
//       exclusively by the driver, except the GPIO_DATA2.
// - input buffer: none
// - output buffer: PAC194X5X_GET_SMBUS_SETTINGS
////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_SMBUS_SETTINGS {
    PAC194X5X_SMBUS_SETTINGS_REGISTER SMBUSreg;     // SMBUS_REGISTER value
} PAC194X5X_GET_SMBUS_SETTINGS, *PPAC194X5X_GET_SMBUS_SETTINGS;

#define IOCTL_PAC194X5X_GET_SMBUS_SETTINGS           CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                              FNUMBER_PAC194X5X_GET_SMBUS_SETTINGS,\
                                                              METHOD_BUFFERED,\
                                                              FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_ANY_ALERT
//
// Returns the value of the ANY_ALERT flag from the SMBUS_SETTINGS register.
// - input buffer: none
// - output buffer: PAC194X5X_GET_ANY_ALERT
////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_ANY_ALERT {
    UCHAR AnyAlert;     // Out: the IOCTL_PAC194X5X_GET_ANY_ALERT driver operation returns
                        // in the lsb the value of the ANY_ALERT bit from SMBUS_SETTINGS
} PAC194X5X_GET_ANY_ALERT, *PPAC194X5X_GET_ANY_ALERT;

#define IOCTL_PAC194X5X_GET_ANY_ALERT           CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                         FNUMBER_PAC194X5X_GET_ANY_ALERT,\
                                                         METHOD_BUFFERED,\
                                                         FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_GPIO_DATA2
//
// Returns the value of the GPIO_DATA2 flag from the SMBUS_SETTINGS register.
// - input buffer: none
// - output buffer: PAC194X5X_GET_GPIO_DATA2
////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_GPIO_DATA2 {
    UCHAR GpioData2;     // Out: the IOCTL_PAC194X5X_GET_GPIO_DATA2 driver returns in
                         // the lsb the value of the GPIO_DATA2 bit from SMBUS_SETTINGS
} PAC194X5X_GET_GPIO_DATA2, *PPAC194X5X_GET_GPIO_DATA2;

#define IOCTL_PAC194X5X_GET_GPIO_DATA2           CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                          FNUMBER_PAC194X5X_GET_GPIO_DATA2,\
                                                          METHOD_BUFFERED,\
                                                          FILE_READ_DATA)
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_GPIO_DATA2
//
// Provides user control over the value of the GPIO_DATA2 flag in the SMBUS_SETTINGS register.
// - input buffer: PAC194X5X_SET_GPIO_DATA2
// - output buffer: none
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_SET_GPIO_DATA2 {
    UCHAR GpioData2;     // The lsb is the value to be copied into GPIO_DATA2 bit from SMBUS_SETTINGS
} PAC194X5X_SET_GPIO_DATA2, *PPAC194X5X_SET_GPIO_DATA2;

#define IOCTL_PAC194X5X_SET_GPIO_DATA2           CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                          FNUMBER_PAC194X5X_SET_GPIO_DATA2,\
                                                          METHOD_BUFFERED,\
                                                          FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_NEG_POWER
//
// Returns the content of the channel polarity and full-scale range (FSR)
// configuration registers NEG_PWR_FSR, NEG_PWR_FSR_ACT and NEG_PWR_FSR_LAT
// - input buffer: none
// - output buffer: PAC194X5X_GET_NEG_POWER
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_NEG_POWER {
    PAC194X5X_NEG_PWR_FSR_REGISTER NegPowerReg;           // NEG_PWR_FSR register value
    PAC194X5X_NEG_PWR_FSR_REGISTER NegPowerAct;           // NEG_PWR_FSR_ACT register value
    PAC194X5X_NEG_PWR_FSR_REGISTER NegPowerLat;           // NEG_PWR_FSR_LAT register value
} PAC194X5X_GET_NEG_POWER, *PPAC194X5X_GET_NEG_POWER;

#define IOCTL_PAC194X5X_GET_NEG_POWER        CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                      FNUMBER_PAC194X5X_GET_NEG_POWER,\
                                                      METHOD_BUFFERED,\
                                                      FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_NEG_POWER
//
// Provides user control over channel polarity and full-scale range (FSR)
// configuration register NEG_PWR_FSR. Channel polarity and FSR can be changed
// by the user at any moment, regardless the EMI status.
// The driver activates the new device configuration using a REFRESH command.
// - input buffer: PAC194X5X_SET_NEG_POWER
// - output buffer: none (zero bytes returned)
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_SET_NEG_POWER {
    PAC194X5X_NEG_PWR_FSR_REGISTER NegPowerReg;           // value to be written in
                                                          // NEG_PWR_FSR register
} PAC194X5X_SET_NEG_POWER, *PPAC194X5X_SET_NEG_POWER;

#define IOCTL_PAC194X5X_SET_NEG_POWER        CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                      FNUMBER_PAC194X5X_SET_NEG_POWER,\
                                                      METHOD_BUFFERED,\
                                                      FILE_WRITE_DATA)
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_SLOW
//
// Returns the value of the SLOW register.
// Note: the configuration of the SLOW register is controlled exclusively
//       by the driver.
// - input buffer: none
// - output buffer: PAC194X5X_GET_SLOW
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_SLOW {
    PAC194X5X_SLOW_REGISTER SlowReg;     // SLOW register value
} PAC194X5X_GET_SLOW, *PPAC194X5X_GET_SLOW;

#define IOCTL_PAC194X5X_GET_SLOW            CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                     FNUMBER_PAC194X5X_GET_SLOW,\
                                                     METHOD_BUFFERED,\
                                                     FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_ACCUM_CONFIG
//
// Returns the value of the ACCUM_CONFIG register.
// - input buffer: none
// - output buffer: PAC194X5X_GET_ACCUM_CONFIG
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_ACCUM_CONFIG {
    PAC194X5X_ACCUM_CONFIG_REGISTER AccumConfigReg;     // ACCUM_CONFIG register value
    PAC194X5X_ACCUM_CONFIG_REGISTER AccumConfigAct;     // ACCUM_CONFIG_ACT register value
    PAC194X5X_ACCUM_CONFIG_REGISTER AccumConfigLat;     // ACCUM_CONFIG_LAT register value
} PAC194X5X_GET_ACCUM_CONFIG, *PPAC194X5X_GET_ACCUM_CONFIG;

#define IOCTL_PAC194X5X_GET_ACCUM_CONFIG            CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                             FNUMBER_PAC194X5X_GET_ACCUM_CONFIG,\
                                                             METHOD_BUFFERED,\
                                                             FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_ACCUM_CONFIG
//
// Provides limited user control over the value of the ACCUM_CONFIG register.
// The driver activates the new device configuration using a REFRESH command.
// Notes:
//  - The "Reserved" values for the ACCUM_CONFIG device register are rejected,
//    with "invalid parameters" error code STATUS_INVALID_PARAMETER.
//  - The accumulator configuration change requests for the channels which have
//    EMI interfaces are ignored.
//
// - input buffer: PAC194X5X_SET_ACCUM_CONFIG
// - output buffer: none (zero bytes return)
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_SET_ACCUM_CONFIG {
    PAC194X5X_ACCUM_CONFIG_REGISTER AccumConfigReg;     // ACCUM_CONFIG register value
} PAC194X5X_SET_ACCUM_CONFIG, *PPAC194X5X_SET_ACCUM_CONFIG;

#define IOCTL_PAC194X5X_SET_ACCUM_CONFIG            CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                             FNUMBER_PAC194X5X_SET_ACCUM_CONFIG,\
                                                             METHOD_BUFFERED,\
                                                             FILE_WRITE_DATA)
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_ALERT_STATUS
//
// Returns the value of the ALERT_STATUS register.
// - input buffer: none
// - output buffer: PAC194X5X_GET_ALERT_STATUS
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_ALERT_STATUS {
    PAC194X5X_ALERT_STATUS_REGISTER AlertStatus;     // ALERT_STATUS register value
} PAC194X5X_GET_ALERT_STATUS, *PPAC194X5X_GET_ALERT_STATUS;

#define IOCTL_PAC194X5X_GET_ALERT_STATUS            CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                             FNUMBER_PAC194X5X_GET_ALERT_STATUS,\
                                                             METHOD_BUFFERED,\
                                                             FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_ALERT_ENABLE
//
// Returns the value of the ALERT_ENABLE register.
// - input buffer: none
// - output buffer: PAC194X5X_GET_ALERT_ENABLE
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_ALERT_ENABLE {
    PAC194X5X_ALERT_ENABLE_REGISTER AlertEnable;     // ALERT_ENABLE register value
} PAC194X5X_GET_ALERT_ENABLE, *PPAC194X5X_GET_ALERT_ENABLE;

#define IOCTL_PAC194X5X_GET_ALERT_ENABLE            CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                             FNUMBER_PAC194X5X_GET_ALERT_ENABLE,\
                                                             METHOD_BUFFERED,\
                                                             FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_ALERT_ENABLE
//
// Provides user control over the device configuration bits from
// ALERT_ENABLE register.
// - input buffer: PAC194X5X_SET_ALERT_ENABLE
// - output buffer: none
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_SET_ALERT_ENABLE {
    PAC194X5X_ALERT_ENABLE_REGISTER AlertEnable;     // ALERT_ENABLE register value
} PAC194X5X_SET_ALERT_ENABLE, *PPAC194X5X_SET_ALERT_ENABLE;

#define IOCTL_PAC194X5X_SET_ALERT_ENABLE            CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                             FNUMBER_PAC194X5X_SET_ALERT_ENABLE,\
                                                             METHOD_BUFFERED,\
                                                             FILE_WRITE_DATA)
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_GPIO_ALERT2
//
// Returns the value of the GPIO_ALERT2 register.
// - input buffer: none
// - output buffer: PAC194X5X_GET_GPIO_ALERT2
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_GPIO_ALERT2 {
    PAC194X5X_GPIO_ALERT2_REGISTER GpioAlert2;     // GPIO_ALERT2 register value
} PAC194X5X_GET_GPIO_ALERT2, *PPAC194X5X_GET_GPIO_ALERT2;

#define IOCTL_PAC194X5X_GET_GPIO_ALERT2             CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                             FNUMBER_PAC194X5X_GET_GPIO_ALERT2,\
                                                             METHOD_BUFFERED,\
                                                             FILE_READ_DATA)
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_GPIO_ALERT2
//
// Provides user control over the device configuration bits from
// GPIO_ALERT2 register.
// Note:
//  -The GPIO/ALERT2 pin must be configured as alert pin in order to output
//   the alert signals.The GPIO / ALERT2 configuration can be controlled using
//   the IOCTL_PAC194X5X_SET_GPIO2_MODE operation or
//   the IOCTL_PAC194X5X_SET_CTRL operation.
//
// - input buffer: PAC194X5X_SET_GPIO_ALERT2
// - output buffer: none
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_SET_GPIO_ALERT2 {
    PAC194X5X_GPIO_ALERT2_REGISTER GpioAlert2;     // GPIO_ALERT2 register value
} PAC194X5X_SET_GPIO_ALERT2, *PPAC194X5X_SET_GPIO_ALERT2;

#define IOCTL_PAC194X5X_SET_GPIO_ALERT2             CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                             FNUMBER_PAC194X5X_SET_GPIO_ALERT2,\
                                                             METHOD_BUFFERED,\
                                                             FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_OC_LIMIT
//
// Returns the alert triggering limit values from the "OC limit" registers:
//      - OC limit Ch1 (30h)
//      - OC limit Ch2 (31h)
//      - OC limit Ch3 (32h)
//      - OC limit Ch4 (33h)
// - input buffer: none
// - output buffer: PAC194X5X_CURRENT_LIMIT
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_CURRENT_LIMIT {
    LONGLONG CurrentLimit[PAC_MAXIMUM_CHANNELS];       // OC limit Ch1->Ch4 (30h-33h) values or
                                                       // UC limit Ch1->Ch4 (34h-37h) values
                                                       // expressed in pico-Amps
} PAC194X5X_CURRENT_LIMIT, *PPAC194X5X_CURRENT_LIMIT;

#define IOCTL_PAC194X5X_GET_OC_LIMIT          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                       FNUMBER_PAC194X5X_GET_OC_LIMIT,\
                                                       METHOD_BUFFERED,\
                                                       FILE_READ_DATA)
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_OC_LIMIT
//
// Provides user control over the alert triggering limit values from the "OC limit" registers:
//      - OC limit Ch1 (30h)
//      - OC limit Ch2 (31h)
//      - OC limit Ch3 (32h)
//      - OC limit Ch4 (33h)
// - input buffer: PAC194X5X_CURRENT_LIMIT
// - output buffer: none
////////////////////////////////////////////////////////////////////////////////////////////////

#define IOCTL_PAC194X5X_SET_OC_LIMIT          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                       FNUMBER_PAC194X5X_SET_OC_LIMIT,\
                                                       METHOD_BUFFERED,\
                                                       FILE_WRITE_DATA)
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_UC_LIMIT
//
// Returns the alert triggering limit values from the "UC limit" registers:
//      - UC limit Ch1 (34h)
//      - UC limit Ch2 (35h)
//      - UC limit Ch3 (36h)
//      - UC limit Ch4 (37h)
// - input buffer: none
// - output buffer: PAC194X5X_CURRENT_LIMIT
////////////////////////////////////////////////////////////////////////////////////////////////

#define IOCTL_PAC194X5X_GET_UC_LIMIT          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                       FNUMBER_PAC194X5X_GET_UC_LIMIT,\
                                                       METHOD_BUFFERED,\
                                                       FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_UC_LIMIT
//
// Provides user control over the alert triggering limit values from the "UC limit" registers:
//      - UC limit Ch1 (34h)
//      - UC limit Ch2 (35h)
//      - UC limit Ch3 (36h)
//      - UC limit Ch4 (37h)
// - input buffer: PAC194X5X_CURRENT_LIMIT
// - output buffer: none
////////////////////////////////////////////////////////////////////////////////////////////////

#define IOCTL_PAC194X5X_SET_UC_LIMIT          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                       FNUMBER_PAC194X5X_SET_UC_LIMIT,\
                                                       METHOD_BUFFERED,\
                                                       FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_OP_LIMIT
//
// Returns the alert triggering limit values from the "OP limit" registers:
//      - OP limit Ch1 (38h)
//      - OP limit Ch2 (39h)
//      - OP limit Ch3 (3Ah)
//      - OP limit Ch4 (3Bh)
// - input buffer: none
// - output buffer: PAC194X5X_POWER_LIMIT
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_POWER_LIMIT {
    LONGLONG VPowerLimit[PAC_MAXIMUM_CHANNELS];   // OP limit Ch1->Ch4 (38h-3Bh) values
                                                  // expressed in pico-Watts
} PAC194X5X_POWER_LIMIT, *PPAC194X5X_POWER_LIMIT;

#define IOCTL_PAC194X5X_GET_OP_LIMIT          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                       FNUMBER_PAC194X5X_GET_OP_LIMIT,\
                                                       METHOD_BUFFERED,\
                                                       FILE_READ_DATA)
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_OP_LIMIT
//
// Provides user control over the alert triggering limit values from the "OP limit" registers:
//      - OP limit Ch1 (38h)
//      - OP limit Ch2 (39h)
//      - OP limit Ch3 (3Ah)
//      - OP limit Ch4 (3Bh)
// - input buffer: PAC194X5X_POWER_LIMIT
// - output buffer: none
////////////////////////////////////////////////////////////////////////////////////////////////

#define IOCTL_PAC194X5X_SET_OP_LIMIT          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                       FNUMBER_PAC194X5X_SET_OP_LIMIT,\
                                                       METHOD_BUFFERED,\
                                                       FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_OV_LIMIT
//
// Returns the alert triggering limit values from the "OV limit" registers:
//      - OV limit Ch1 (3Ch)
//      - OV limit Ch2 (3Dh)
//      - OV limit Ch3 (3Eh)
//      - OV limit Ch4 (3Fh)
// - input buffer: none
// - output buffer: PAC194X5X_VOLTAGE_LIMIT
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_VOLTAGE_LIMIT {
    LONGLONG VoltageLimit[PAC_MAXIMUM_CHANNELS];    // OV limit Ch1->Ch4 (3Ch-3Fh) values or
                                                    // UV limit Ch1->Ch4 (40h-43h) values
                                                    // expressed in pico-Volts
} PAC194X5X_VOLTAGE_LIMIT, *PPAC194X5X_VOLTAGE_LIMIT;

#define IOCTL_PAC194X5X_GET_OV_LIMIT          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                       FNUMBER_PAC194X5X_GET_OV_LIMIT,\
                                                       METHOD_BUFFERED,\
                                                       FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_OV_LIMIT
//
// Provides user control over the alert triggering limit values from the "OV limit" registers:
//      - OV limit Ch1 (3Ch)
//      - OV limit Ch2 (3Dh)
//      - OV limit Ch3 (3Eh)
//      - OV limit Ch4 (3Fh)
// - input buffer: PAC194X5X_VOLTAGE_LIMIT
// - output buffer: none
////////////////////////////////////////////////////////////////////////////////////////////////

#define IOCTL_PAC194X5X_SET_OV_LIMIT          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                       FNUMBER_PAC194X5X_SET_OV_LIMIT,\
                                                       METHOD_BUFFERED,\
                                                       FILE_WRITE_DATA)
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

```
////////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_UV_LIMIT
//
// Returns the alert triggering limit values from the "UV limit" registers:
//      - UV limit Ch1 (40h)
//      - UV limit Ch2 (41h)
//      - UV limit Ch3 (42h)
//      - UV limit Ch4 (43h)
// - input buffer: none
// - output buffer: PAC194X5X_VOLTAGE_LIMIT
////////////////////////////////////////////////////////////////////////////////////////////////////

#define IOCTL_PAC194X5X_GET_UV_LIMIT          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                       FNUMBER_PAC194X5X_GET_UV_LIMIT,\
                                                       METHOD_BUFFERED,\
                                                       FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_UV_LIMIT
//
// Provides user control over the alert triggering limit values from the "UV limit" registers:
//      - UV limit Ch1 (40h)
//      - UV limit Ch2 (41h)
//      - UV limit Ch3 (42h)
//      - UV limit Ch4 (43h)
// - input buffer: PAC194X5X_VOLTAGE_LIMIT
// - output buffer: none
////////////////////////////////////////////////////////////////////////////////////////////////////

#define IOCTL_PAC194X5X_SET_UV_LIMIT          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                       FNUMBER_PAC194X5X_SET_UV_LIMIT,\
                                                       METHOD_BUFFERED,\
                                                       FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_LIMIT_NSAMPLES
//
// Returns the values of the limit Nsamples registers:
//      - OC limit Nsamples
//      - UC limit Nsamples
//      - OP limit Nsamples
//      - OV limit Nsamples
//      - UV limit Nsamples
// - input buffer: none
// - output buffer: PAC194X5X_GET_LIMIT_NSAMPLES
////////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_LIMIT_NSAMPLES {
    PAC194X5X_LIMIT_NSAMPLES_REGISTER OClimit_Nsamples;       // OC limit Nsamples register value
    PAC194X5X_LIMIT_NSAMPLES_REGISTER UClimit_Nsamples;       // UC limit Nsamples register value
    PAC194X5X_LIMIT_NSAMPLES_REGISTER OPlimit_Nsamples;       // OP limit Nsamples register value
    PAC194X5X_LIMIT_NSAMPLES_REGISTER OVlimit_Nsamples;       // OV limit Nsamples register value
    PAC194X5X_LIMIT_NSAMPLES_REGISTER UVlimit_Nsamples;       // UV limit Nsamples register value
} PAC194X5X_GET_LIMIT_NSAMPLES, *PPAC194X5X_GET_LIMIT_NSAMPLES;

#define IOCTL_PAC194X5X_GET_LIMIT_NSAMPLES          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                             FNUMBER_PAC194X5X_GET_LIMIT_NSAMPLES,\
                                                             METHOD_BUFFERED,\
                                                             FILE_READ_DATA)
```

```
////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_LIMIT_NSAMPLES
//
// Provides user control over the limit Nsamples registers:
//      - OC limit Nsamples
//      - UC limit Nsamples
//      - OP limit Nsamples
//      - OV limit Nsamples
//      - UV limit Nsamples
// - input buffer: PAC194X5X_SET_LIMIT_NSAMPLES
// - output buffer: none
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_SET_LIMIT_NSAMPLES {
    PAC194X5X_LIMIT_NSAMPLES_REGISTER OClimit_Nsamples;       // OC limit Nsamples register value
    PAC194X5X_LIMIT_NSAMPLES_REGISTER UClimit_Nsamples;       // UC limit Nsamples register value
    PAC194X5X_LIMIT_NSAMPLES_REGISTER OPlimit_Nsamples;       // OP limit Nsamples register value
    PAC194X5X_LIMIT_NSAMPLES_REGISTER OVlimit_Nsamples;       // OV limit Nsamples register value
    PAC194X5X_LIMIT_NSAMPLES_REGISTER UVlimit_Nsamples;       // UV limit Nsamples register value
} PAC194X5X_SET_LIMIT_NSAMPLES, *PPAC194X5X_SET_LIMIT_NSAMPLES;

#define IOCTL_PAC194X5X_SET_LIMIT_NSAMPLES          CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                             FNUMBER_PAC194X5X_SET_LIMIT_NSAMPLES,\
                                                             METHOD_BUFFERED,\
                                                             FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_GET_ACCUM_LIMITS
//
// Returns the alert triggering limit values of the ACC_Fullness_limits register:
// - input buffer: none
// - output buffer: PAC194X5X_GET_ACCUM_LIMITS
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_GET_ACCUM_LIMITS {
    PAC194X5X_ACCUM_LIMITS_REGISTER AccumLimitsReg;           // ACC_FULLNESS_LIMITS register value
} PAC194X5X_GET_ACCUM_LIMITS, *PPAC194X5X_GET_ACCUM_LIMITS;

#define IOCTL_PAC194X5X_GET_ACCUM_LIMITS        CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                         FNUMBER_PAC194X5X_GET_ACCUM_LIMITS,\
                                                         METHOD_BUFFERED,\
                                                         FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC194X5X_SET_ACCUM_LIMITS
//
// Provides user control over the alert triggering limit values from the
// ACC_Fullness_limits register:
// - input buffer: PAC194X5X_SET_ACCUM_LIMITS
// - output buffer: none
////////////////////////////////////////////////////////////////////////////////////////////////
typedef struct _PAC194X5X_SET_ACCUM_LIMITS {
    PAC194X5X_ACCUM_LIMITS_REGISTER AccumLimitsReg;           // ACC_FULLNESS_LIMITS register value
} PAC194X5X_SET_ACCUM_LIMITS, *PPAC194X5X_SET_ACCUM_LIMITS;

#define IOCTL_PAC194X5X_SET_ACCUM_LIMITS        CTL_CODE(FILE_DEVICE_PAC194X5X,\
                                                         FNUMBER_PAC194X5X_SET_ACCUM_LIMITS,\
                                                         METHOD_BUFFERED,\
                                                         FILE_WRITE_DATA)
```


© 2021-2022 Microchip Technology Inc. and its subsidiaries

Appendix D. Glossary of Terms and Acronyms

TABLE D-1: GLOSSARY OF TERMS AND ACRONYMS

Term	Description
ACPI	Advanced Configuration and Power Interface
ASL	ACPI source Language
BIOS	Basic Input/Output System
CPU	Central Processing Unit
DSM	Device Specific Method
E3	Energy Estimation Engine
EC	Embedded Controller
EMI	Energy Meter Interface
Forced Shutdown	Unscheduled transition of the system G0/S0 to G2/S5 via ACPI power button override
FSR	Full Scale Range
GPIO	General Purpose Input/Output
GUID	Globally Unique Identifier
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IOCTL	I/O Control code. The `DeviceIoControl` system call is a general purpose interface that can send control codes to a device driver. Each control code represents an operation for the device driver to perform.
OEM	Original Equipment Manufacturer
OS	Operating System
POR	Power-On Reset
SMBus™	System Management Bus
SPB	Simple Peripheral Bus
SpbCx	SPB Framework Extension
SoC	System on Chip
UUID	Universal Unique Identifier
WDF	Windows Driver Foundation

© 2021-2022 Microchip Technology Inc. and its subsidiaries

NOTES:

© 2021-2022 Microchip Technology Inc. and its subsidiaries

Worldwide Sales and Service

AMERICAS

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

ASIA/PACIFIC

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

ASIA/PACIFIC

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

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4485-5910
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: 49-7131-72400

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: 47-7288-4388

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

© 2021-2022 Microchip Technology Inc. and its subsidiaries