PAC193X
Microsoft® Windows® 10 and Windows 11 Device Driver User's Guide

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.

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

ISBN: 978-1-5224-9460-7

Table of Contents

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 website (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 PAC193X Device Driver for Microsoft® Windows® 10 or Windows 11. Items discussed in this chapter include:

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

DOCUMENT LAYOUT

This document presents the PAC193X Windows Device Driver technical aspects that users need to know in order to access the energy measurements reported by devices. The manual layout is as follows:

- **Chapter 1. "Product Overview"** – Important information about the PAC193X Windows Device Driver.
- **Chapter 2. "Driver Installation"** – Includes instructions on installing and starting the PAC193X Windows Device Driver.
- **Chapter 3. "Device Instantiation and Initialization"** – Important Information about the PAC193X Windows Device Driver initial configuration.
- **Chapter 4. "PAC193X Device Interfaces"** – Information about the PAC193X Windows Device Driver interfaces.
- **Chapter 5. "PAC193X Driver Characteristics"** – Information about the PAC193X Windows Device Driver features.
- **Appendix A. "Driver Revision History"** – Contains driver bug fixes and changes.
- **Appendix B. "PAC193X Device Control Interface"** – Contains the C definitions of EMI and control interface data structures.

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 PAC193X Windows Device Driver. Other useful documents are listed below. The following Microchip documents are available and recommended as supplemental reference resources:

- **PAC193X Application Note – *AN2534, "PAC193X Integration Notes for Microsoft® Windows® 10 and Windows 11 Driver Support"* (DS00002534)**
- **PAC193X Data Sheet – *"PAC1932/3/4 Multi-Channel DC Power/Energy Monitor with Accumulator"* (DS20005850)**
- **PAC1934 User Guide – *"PAC1934 Evaluation Board (ADM00805) User's Guide"* (DS50002673)**

THE MICROCHIP WEBSITE

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

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 website at:
http://www.microchip.com/support.

DOCUMENT REVISION HISTORY

Revision D (January 2022)

- Added the PAC193X device driver support for Windows 11
- The information in this document applies to the PAC193X Driver Release 1.4.1, 1.4.2 and later

Revision C (April 2021)

- Updated **Section 5.5 "Low-Power Mode and Power States"**
- Updated **Table A-1**
- Updated **Appendix B. "PAC193X Device Control Interface"**
- The information in this document applies to the PAC193X Driver Release 1.4.0

Revision B (June 2019)

- Updated **Section 2.2 "Driver Installation"**
- Updated **Chapter 3. "Device Instantiation and Initialization"**
- Updated **Table 4-2**, **Table 4-3** and **Table 4-4**
- Updated **Section 4.3 "Device Control Interface"**
- Updated **Chapter 5. "PAC193X Driver Characteristics"**
- Updated **Table A-1**
- Updated **Section B.2 "PAC193X Device Definitions"** and **Section B.3 "Interface and IOCTL Definitions"**
- The information in this document applies to the PAC193X Driver Release 1.3

Revision A (September 2017)

- Initial release of this document.
- The information in this document applies to the following PAC193X driver releases: 1.0, 1.1 and 1.2.

Chapter 1. Product Overview

1.1 INTRODUCTION

This document presents the PAC193X Windows Device Driver technical aspects that users need to know in order to access the energy measurements reported by PAC193X devices.

Starting with the 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 and Windows 11 uses estimation algorithms or the real-time energy measured by metering ICs connected into various power rails of the system.

The metering ICs's device drivers must report the energy measurements through a standard software interface: Energy Metering Interface (EMI). The E3 service or the user applications can access the energy data by calling a standard set of EMI IOCTL services.

TABLE 1-1: 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
E3	Energy Estimation Engine
EC	Embedded Controller
EMI	Energy Metering Interface
Forced Shutdown	Unscheduled Transition of the System G0/S0 to G2/S5 through ACPI Power Button Override
GUID	Globally Unique Identifier
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
IOCTL	I/O Control. The `DeviceIoControl` system call is a general purpose interface that can send control codes to a device. Each control code represents an operation for the device driver to perform.
OEM	Original Equipment Manufacturer
OS	Operating System
SMBus™	System Management Bus
SPB	Simple Peripheral Bus
SoC	System on Chip
SpbCx	SPB Framework Extension
UUID	Universal Unique Identifier
WDF	Windows Driver Foundation

NOTES:

Chapter 2. Driver Installation

2.1 PREREQUISITES

Before installing the driver on the target system, the following driver dependencies must be resolved:

- The PAC193X devices are integrated in the host system
- The I^2C bus controller device driver is SPB Framework Extension (SpbCx) compliant

2.1.1 PAC193X Integration in the Host System

2.1.1.1 ACPI ASL

Refer to the indications listed in the PAC193X Application Note – *AN2534, "PAC193X Integration Notes for Microsoft® Windows® 10 and Windows 11 Driver Support"* (DS00002534) and make the necessary ACPI ASL changes to ensure that PAC193X devices are enumerated by the Microsoft Windows Device Manager.

2.1.1.2 SLOW PIN

The PAC 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 pin 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 Windows® driver doesn't monitor the SLOW pin transitions while the system is running in S0 state.

The PAC193X Windows 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 transition high-to-low saves into the device readable registers the data accumulated while the system was off

2.1.2 SpbCx Compliant I^2C Bus Controller Device Driver

To access the I^2C bus and communicate with the PAC devices, the PAC193X Windows Device Driver uses the standard interface and protocol implemented by the 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 Microsoft Windows installation or Microsoft Windows update.

2.2 DRIVER INSTALLATION

There are several options to install the PAC193X Windows Device Driver:

- Install through 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.

2.2.1 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 Microsoft® Windows® 10".

2.2.2 Install Using the DevCon Utility

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

1. Based on the target system OS (32-bit or 64-bit), extract the following files from the driver package:
   - `PAC193x.sys`
   - `PAC193x.inf`
   - `PAC193x.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 pac193x.inf ACPI\MCHP1930
```

2.2.2.1 VERIFY THE INSTALLATION

To verify that the PAC193X Windows Device Driver installation was successfully completed, open the Device Manager and check if the PAC193X devices are listed under the "Microchip Energy Metering Devices" category and they are reported as working properly. See Figure 2-1.

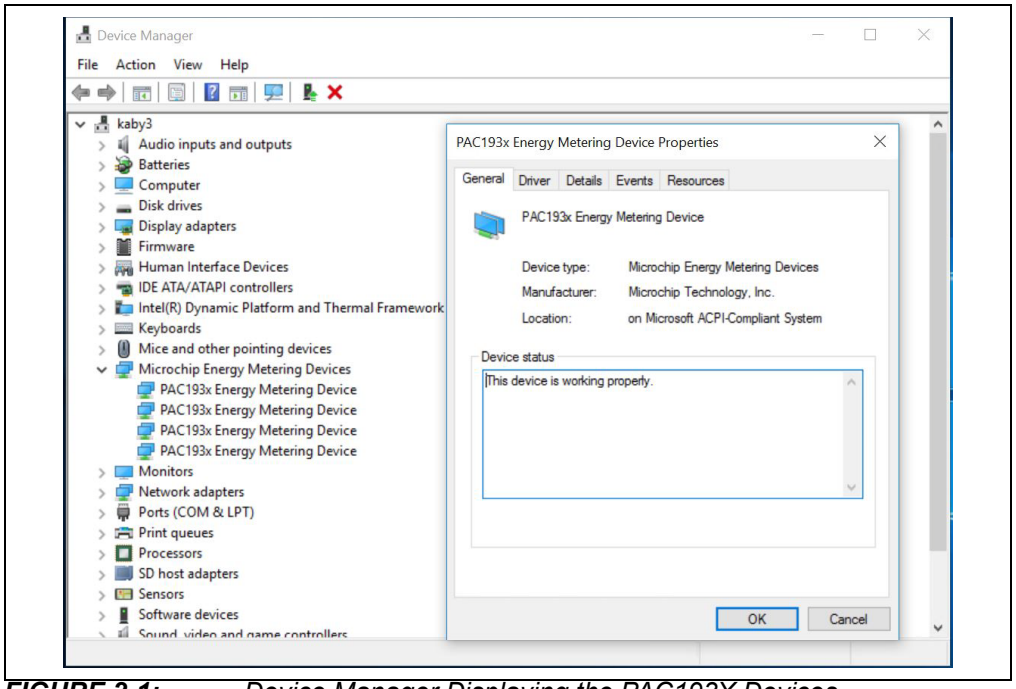

FIGURE 2-1: *Device Manager Displaying the PAC193X Devices.*

To open the Device Manager:

1. Click the bottom left **Start** button on the desktop, type device manager in the search box and click **Device Manager** on the menu.
2. Press <Windows>+<X> to open the Quick Access Menu and select **Device Manager**.

Chapter 3. Device Instantiation and Initialization

3.1 GETTING STARTED

The PAC193X devices connected to the system I^2C buses are enumerated by the Advanced Configuration and Power Interface (ACPI) in its role as a bus driver, as `ACPI\MCHP1930\<_UID>`, where `_UID` is the value defined for each device in the ACPI ASL code. The ACPI ASL code details specific for the PAC193X are listed in the PAC193X Application Notes.

The PAC193X Windows Device Driver is a standard Kernel-Mode Driver Framework (KMDF) function driver that, according to its INF, is instantiated by the PnP Manager whenever a bus driver enumerates devices with the Hardware ID: *ACPI\MCHP1930*.

During device initialization, the PAC193X Windows Device Driver evaluates the device's ACPI BIOS_DSM method, which is identified by the Universal Unique ID (UUID): *033771E0-1705-47B4-9535-D1BBE14D9A09*. The information that the BIOS_DSM method returns includes:

- The value of the Shunt Resistor (R_{SENSE}) for each channel, expressed in milliohms and microohms:
  - The resistor value is used for the energy calculations and must be an integer.
  - R_{SENSE} = 0 indicates a channel which is not connected to a power rail. The driver reports no data for this channel.
- The Windows specified name for 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 Metering Interface*"*. The Windows E3 service can use the data exported by the EMI interfaces if their names are compatible with Windows power rails' 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 are still collected, processed and reported through the PAC193X control interface. There is one single control interface created for each PAC193X device.
- EMI enable/disable bit mask indicating for which named channels the driver must NOT create an EMI interface:
  - This parameter is not available in DSM Rev. 0 and the driver assumes that no EMI is masked out.
- Channel polarity indication bit mask:
  - This bit mask has the same structure and functionality as the device `NEG_PWR` register.
  - This parameter is not available in DSM Rev. 0 and the driver configures all the channels in Unipolar mode.
- The sampling frequency options the driver must use to configure the PAC device when the system enters and exits the S0 Active state.
  - These parameters are not available in DSM Rev. 0 and the driver assumes 1024 sps and 8 sps for the two options.

- The driver "watchdog" interval:
  - If the system is Active, the driver must issue a new device `REFRESH` command and save the accumulated data at the end of the configured time period since the last `REFRESH`.
  - This parameter is not available in DSM Rev. 0 and the driver sets this time interval to 240 seconds.

3.2 DEVICE INITIAL CONFIGURATION

PAC193X Windows Device Driver preserves the device POR settings for some configuration options and applies the user configuration options provisioned by the BIOS ASL code:

- I^2C/SMBus™ interface is configured as I^2C. SMBus is not supported currently by Microsoft Windows. No communication settings change is allowed by the driver.
- Sample rate (SRN): Starting with DSM Rev. 1, the driver uses the BIOS ASL provisioned parameters to configure the SRN setting when the system enters Active mode and when the system exits the Active mode, respectively. For DSM Rev. 0, the driver uses the default 1024 Hz for the system Active mode sampling rate and the "Slow" 8 Hz sampling rate for non-Active mode.
- SLOW/ALERT pin is configured as Slow. No configuration change is allowed by the driver.
- Slow signal transitions trigger limited `REFRESH`. The limited `REFRESH` triggered by the SLOW pin is defined as all Results registers are updated and the accumulator and accumulator count are reset, but no changes to the device configuration are allowed.
- All the device channels are enabled, except the channels with 0Ω sense resistors, which are disabled by the driver.
- All the channels are unidirectional by default. Starting with DSM Rev 1.0, the driver uses the parameters provisioned by the BIOS ASL code to change the polarity configuration for either V_{BUS}, V_{SENSE} or both. The driver arithmetic is adjusted accordingly. Changing to signed measurements also implies a 1-bit resolution loss for those measured values.

In addition, the driver takes the necessary configuration actions in order to implement the work arounds for the errata affecting Silicon Revisions 0x01, 0x02 and 0x03 (silicon revision values reported by REVISION ID register 0xFF). Refer to the *"PAC193X Data Sheet"* for details regarding the Revision ID.

Chapter 4. PAC193X Device Interfaces

4.1 INTRODUCTION

A device interface contains the symbolic link name which the user applications 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 described in this paragraph are implemented by `setupapi.lib` and are available since Microsoft Windows 2000.)

Next, the user application can pass the symbolic link names to the `CreateFile` kernel function to open the device for I/O and receive the assigned file handle. The user application can use the `DeviceIoControl` kernel function and the file handle to communicate with the device, specifying the IOCTL code (the input/output control code) which identifies the requested operation from the device. The device drivers translate the request into device-specific operations and return the results back to the requester.

This chapter presents the class GUID and the IOCTL codes implemented by the PAC193X Windows Device Driver for each type of interface it creates:

- Energy Metering Interface
- Device Control Interface

4.2 ENERGY METERING INTERFACE

The driver creates an EMI interface for each device channel with a non-empty Windows rail name assigned in ACPI ASL. For details regarding the EMI interface, refer to the following Microsoft article: *"Energy Metering Interface"*. The definition details are available in the file, `emi.h`, distributed with the Microsoft Windows Driver Kit.

The GUID for the EMI device interface is:

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

TABLE 4-1: EMI IOCTLS

IOCTL Name	Description
IOCTL_EMI_GET_MEASUREMENT	Retrieves the current energy measurement and the time at which the measurement was taken. • Input: None • Output: EMI_MEASUREMENT_DATA
IOCTL_EMI_GET_METADATA	Retrieves EMI metadata from a device. • Input: None • Output: EMI_METADATA
IOCTL_EMI_GET_METADATA_SIZE	Retrieves the size of the EMI metadata object that can be obtained from the device by issuing an IOCTL_EMI_GET_METADATA request. • Input: None • Output: EMI_METADATA_SIZE
IOCTL_EMI_GET_VERSION	Retrieves the current version of the EMI interface supported by the device. • Input: None • Output: EMI_VERSION

TABLE 4-2: EMI ENUMERATIONS AND STRUCTURES

Topic	Type	Description
EMI_MEASUREMENT_DATA	Output Structure	Energy measurement and the time when the measurement was taken, as reported by IOCTL_EMI_GET_MEASUREMENT. • Energy unit: 1 pWh • Energy value and timestamp value: 64-bit unsigned • Timestamp unit: 100 ns
EMI_MEASUREMENT_UNIT	Enumeration	Represents the available units of energy measurements that can be reported by IOCTL_EMI_GET_MEASUREMENT. • EMI V1 implements only 1 pWh unit
EMI_METADATA	Output Structure	Information about: • The metering device: Hardware model and revision • The metered power rail: Rail name as reported by IOCTL_EMI_GET_METADATA.
EMI_METADATA_SIZE	Output Structure	The size of the EMI_METADATA structure, as reported by IOCTL_EMI_GET_METADATA_SIZE.
EMI_VERSION	Output Structure	The EMI version supported by this device, as reported by IOCTL_EMI_GET_VERSION. • Only EMI V1 specification is currently supported

Appendix B. "PAC193X Device Control Interface" contains the C definitions of the EMI interface data structures.

4.3 DEVICE CONTROL INTERFACE

The driver creates one control interface for each enumerated PAC193X device. Compared to the EMI interface, this interface accepts the `IOCTL_PAC193x_*` requests, and provides users with access to much more data and device configuration options:

- All the measured values (e.g., V_{BUS}, V_{SENSE}, in addition to the accumulated energy)
- Data for all the channels, including the channels which are non-EMI
- Configuration options for each device and channel (sampling rate, channel polarity, channel on/off)

NOTICE

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

- **The SMB/I^2C communication options or the SLOW/ALERT pin configuration options are not accessible for the user**
- **The channel on/off control option is allowed only if there is no EMI interface open for that channel**

The GUID for the control interface is:

4166FE9F-A865-4314-8942-7C12ABA290F6

TABLE 4-3: DEVICE CONTROL IOCTLs

IOCTL Name	Description
`IOCTL_PAC193X_GET_DEVICE_INFO_SIZE`	Returns the size allocation requirements for the buffer that must hold the device info data structure. • Input buffer: None • Output buffer: `PAC193X_GET_DEVICE_INFO_SIZE`
`IOCTL_PAC193X_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: `PAC193X_GET_DEVICE_INFO`
`IOCTL_PAC193X_GET_CTRL`	Returns the content of the device registers: `CTRL`, `CTRL_ACT`, `CTRL_LAT`. • Input buffer: None • Output buffer: `PAC193X_GET_CTRL`
`IOCTL_PAC193X_SET_CTRL`	Provides user-limited control over some device Configuration bits from CTRL registers: `SRN`, `SLEEP` and `SING`. The driver rejects the configuration change requests for `SLEEP` and `SING` if the device has open EMI channels. • Input buffer: `PAC193X_SET_CTRL` • Output buffer: None (zero bytes returned)
`IOCTL_PAC193X_GET_MEASUREMENTS`	Returns for all the channels the data measurements, the timestamp and the device configuration at the sampling time. • Input buffer: None • Output buffer: `PAC193X_MEASUREMENTS`

TABLE 4-3: DEVICE CONTROL IOCTLs (CONTINUED)

IOCTL Name	Description
IOCTL_PAC193X_GET_CHANNEL_CFG	Returns the content of the I²C/SMB and Channel On/Off Configuration registers: CHANNEL_DIS, CHANNEL_DIS_ACT, CHANNEL_DIS_LAT. • Input buffer: None • Output buffer: PAC193X_GET_CHANNEL_CFG
IOCTL_PAC193X_SET_CHANNEL_CFG	Provides user-limited control over some Configuration bits from CHANNEL_DIS registers: CH1, CH2, CH3 and CH4. The driver ignores the requests to change the I²C/SMB Configuration bits: TIMEOUT and BYTECOUNT, NOSKIP. The driver ignores the requests to change the Pointer Skip Configuration bit: NOSKIP. The driver ignores the change values for those channels which have open EMI channels. • Input buffer: PAC193X_SET_CHANNEL_CFG • Output buffer: None (zero bytes returned)
IOCTL_PAC193X_GET_NEG_POWER	Returns the content of the Channel Polarity Configuration registers: NEG_PWR, NEG_PWR_ACT and NEG_PWR_LAT. • Input buffer: None • Output buffer: PAC193X_GET_NEG_POWER
IOCTL_PAC193X_SET_NEG_POWER	Provides user control over Channel Polarity Configuration register: NEG_PWR. Channel polarity is set at the driver initialization but can be changed by the user at any moment, regardless of the EMI status. • Input buffer: PAC193X_SET_NEG_POWER • Output buffer: None (zero bytes returned)
IOCTL_PAC193X_GET_OVERFLOW	Returns the status of the OVF bit from CTRL_ACT and CTRL_LAT. Depending on the value of the input flag, this request can also clear (reset to '0') the OVF flag from the CTRL register. • Input buffer: PAC193X_GET_OVERFLOW • Output buffer: PAC193X_GET_OVERFLOW
IOCTL_PAC193X_DATA_REFRESH	Allows the device to report on measurements without resetting the accumulator. Performs the update of the software accumulators using the REFRESH_V device command. • Input buffer: None • Output buffer: None (zero bytes returned)
IOCTL_PAC193X_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 zero) the software accumulators for the channels without open EMI interfaces. • Input buffer: PAC193X_REFRESH_FLAGS • Output buffer: None (zero bytes returned)

TABLE 4-3: DEVICE CONTROL IOCTLs (CONTINUED)

IOCTL Name	Description
IOCTL_PAC193X_DATA_REFRESH_RESET_GLOBAL	Performs a global update of the software accumulators for all the PAC193X 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. Because Microsoft® Windows® SpbCx framework does not support the I^2C 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. • Input buffer: PAC193X_REFRESH_FLAGS • Output buffer: None (zero bytes returned)
IOCTL_PAC193X_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: PAC193X_LAST_SLOW_DATA
IOCTL_PAC193X_GET_SLOW	Returns the value of the SLOW register. The configuration of the SLOW register is controlled exclusively by the driver. • Input buffer: None • Output buffer: PAC193X_GET_SLOW
IOCTL_PAC193X_GET_INTERFACE_REVISION	Returns the device control interface revision number. This IOCTL is implemented starting with PAC193X Driver Version 1.3, introducing the Interface Revision Rev 0x2. It is assumed that previous driver versions implement the Interface Revision Rev 0x1. • Input buffer: None • Output buffer: PAC193X_GET_INTF_REV
IOCTL_PAC193X_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. This IOCTL is implemented starting with PAC193X Driver Version 1.3, introducing the Interface Revision Rev 0x2. • Input buffer: None • Output buffer: PAC193X_GET_RENSE_MICRO

TABLE 4-4: DEVICE CONTROL STRUCTURES

Structure Name	Type	Description
PAC193X_GET_DEVICE_INFO_SIZE	Output	The size of the PAC193X_GET_DEVICE_INFO structure, as reported by IOCTL_PAC193X_GET_DEVICE_INFO.
PAC193X_GET_DEVICE_INFO	Output	Device and channel information returned by IOCTL_PAC193X_GET_DEVICE_INFO: • Manufacturer ID, product ID, silicon revision • Number of connected channels and their on/off status • Per channel rail name, sense resistor value expressed in milliohm units, EMI interface status (created or not) **CAUTION:** This structure is variable in size, depending on how many channels the device has and the size of the channel name.
PAC193X_GET_CTRL	Output	Control registers' content returned by IOCTL_PAC193X_GET_CTRL. • CTRL • CTRL_ACT • CTRL_LAT
PAC193X_SET_CTRL	Input	CTRL register value the user must provide to IOCTL_PAC193X_SET_CTRL.
PAC193X_MEASUREMENTS	Output	Data structure returned by IOCTL_PAC193X_GET_MEASUREMENTS containing device registers' values: • ACC_COUNT • VPOWERn_ACC • VBUSn, VBUSn_AVG • VSENSEn, VSENSEn_AVG • VPOWERn • NEG_PWR_LAT • CTRL_LAT • CHANNEL_DIS_LAT and the Software registers implemented by the device driver: • Last sample report timestamp • Software accumulator count • Unsigned software accumulator • Signed software accumulator **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 MSB is a sign bit). The reported value must be cast to the necessary data type for the subsequent processing. The driver already implements the work around for the VPOWER_ACC and VPOWERN errata which affects the devices with Revision ID = 0x01.
PAC193X_GET_CHANNEL_CFG	Output	Channel Control registers' values returned by IOCTL_PAC193X_GET_CHANNEL_CFG: • CHANNEL_DIS • CHANNEL_DIS_ACT • CHANNEL_DIS_LAT

TABLE 4-4: DEVICE CONTROL STRUCTURES (CONTINUED)

Structure Name	Type	Description
PAC193X_SET_CHANNEL_CFG	Input	CHANNEL_DIS register value the user must provide to IOCTL_PAC193X_SET_CHANNEL_CFG.
PAC193X_GET_NEG_POWER	Output	Channel Polarity Configuration registers returned by IOCTL_PAC193X_GET_NEG_POWER: • NEG_PWR • NEG_PWR_ACT • NEG_PWR_LAT
PAC193X_SET_NEG_POWER	Output	NEG_PWR register value the user must provide to IOCTL_PAC193X_SET_NEG_POWER
PAC193X_GET_OVERFLOW	Input/ Output	Input/output structure for IOCTL_PAC193X_GET_OVERFLOW, containing one input and two output flags: • The input flag, if non-zero, requests the Reset of the OVF bit from CTRL. • The output flags return the OVF bits from CTRL_ACT and CTRL_LAT
PAC193X_REFRESH_FLAGS	Input	Input structure for IOCTL_PAC193X_DATA_REFRESH_RESET and IOCTL_PAC193X_DATA_REFRESH_RESET_GLOBAL requests containing one input flag. The flag value set to PAC193X_REFRESH_FLAG_RESET_ACCUMULATORS (defined as one) indicates the request to also clear the software accumulators.
PAC193X_LAST_SLOW_DATA	Output	Data structure returned by IOCTL_PAC193X_GET_LAST_SLOW_DATA containing device registers' values sampled by the Slow signal HL transition (Slow-triggered limited REFRESH) caused by the system state transition to S0 from other non-powered state: • ACC_COUNT • VPOWERn_ACC • VBUSn, VBUSn_AVG • VSENSEn, VSENSEn_AVG • VPOWERn • NEG_PWR_LAT • CTRL_LAT • CHANNEL_DIS_LAT
PAC193X_GET_SLOW	Output	Slow register value returned by IOCTL_PAC193X_GET_SLOW.
PAC193X_GET_INTF_REV	Output	The device control interface revision number, as reported by IOCTL_PAC193X_GET_INTERFACE_REVISION.
PAC193X_GET_RENSE_MICRO	Output	The sense resistor values for all channels, expressed in microohm units, as returned by IOCTL_PAC193X_GET_RSENSE_MICRO.

Appendix B. "PAC193X Device Control Interface" contains the C definition of the control interface data structures.

NOTES:

Chapter 5. PAC193X Driver Characteristics

5.1 DRIVER CHARACTERISTICS

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

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

5.2 MULTIPLE I/O REQUESTS

Windows E3 module and multiple other user applications (or multiple instances of the same application) may concurrently issue data report requests or device configuration commands. The 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.

5.3 ENERGY METERING INTERFACES (EMI) VS. DEVICE CONTROL INTERFACES

For each PAC device, the PAC193X Windows Device Driver creates one control interface and zero or more EMI interfaces, one for each connected channel (R_{SENSE} > 0). Each channel has a non-empty EMI power rail name ACPI ASL parameter which is not explicitly turned off by ACPI ASL EMI enable/disable flags.

The EMI interfaces are reporting only the positive energy accumulation and the timestamp of the last measurement.

The control interface reports all the measurements performed by the PAC193X devices, including the energy and timestamp figures reported by the EMI interfaces.

The Windows Energy Estimation Engine (E3) collects only the energy figures reported by the EMI interfaces (the EMI power rail names must comply with Microsoft naming conventions in order to ensure that E3 is properly using the data).

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

- EMI interface has Microsoft defined control codes and data structures, which make the application capable of collecting data from any EMI compliant metering device vendor
- Control interface provides much more information using Microchip defined control codes and data structures

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

- Some device configuration options are controlled exclusively by the driver: I^2C interface parameters, pointer skip, ALERT and SLOW pin configuration
- The channel enable/disable status change is accepted only if there is no EMI interface open for the channels affected by the configuration change.

5.4 ENERGY SOFTWARE ACCUMULATORS

The driver implements a set of energy software accumulators for each PAC device to:

- Measure the real energy figure, computed by the driver from the device power product accumulator value, R_{SENSE} value and sampling rate. This is the energy value reported through the EMI and control interface, expressed in pico-watt-hours (pWh).
- Extend the energy accumulation time limit beyond the device capabilities. The 64-bit resolution of the software accumulators should extend the 17 minutes minimum accumulation time, until register saturation, to about two years until the software accumulator overflows.

The device Sample Counter register has a software sample counter companion implemented by the driver.

Every time the driver receives an energy report request, it reads the device Accumulator registers, updates the software accumulators and clears the device accumulators.

For instance, the Windows E3 module is typically collecting the EMI energy every few minutes. But even if there is no EMI interface and no user application to request energy reports, the driver has an internal Watchdog Timer, and it still can update the software accumulators periodically and clear the device accumulators. The driver configures the Watchdog Timer interval with the value provided by the ACPI DSM Rev. 1 function, if the value is less than 900 seconds and more than 60 seconds. If the value is not compliant or the DSM revision is 0, the driver configures the Watchdog Timer to the default 240-second interval.

5.5 LOW-POWER MODE AND POWER STATES

The recommended integration scenario with Windows 10 and Windows 11 systems described for the PAC193X 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 PAC193X devices continue to accumulate the energy data. (For details about the device integration with the host system refer the chapter **Chapter 2. "Driver Installation"**) 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:

- 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 PAC193X 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 PAC193X 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.

5.5.1 Low-Power Mode

The PAC193X 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_PAC193X_SET_CTRL 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 PAC193X 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 PAC193X 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 PAC193X 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 5-1 summarizes the operating power states and the corresponding PAC193X device sampling frequency.

TABLE 5-1: PAC193X DEVICE SAMPLING FREQUENCY

Windows® Power State	PAC193X Sample Frequency	Comments
S0	Active	• Initialized with ACPI BIOS DSM "Active" sample frequency parameter. • User can change it with `IOCTL_PAC193X_SET_CTRL`
S0 low-power idle	Idle	• ACPI BIOS DSM "Idle" sample frequency parameter
S1, S2, S3	Idle or SLOW	• ACPI BIOS DSM "Idle" sample frequency parameter or • 8 Hz if SLOW pin is activated
S4	Idle or SLOW	• ACPI BIOS DSM "Idle" sample frequency parameter or • 8 Hz if SLOW pin is activated
S5	SINGLE_SHOT or SLOW	• No sampling or • 8 Hz if SLOW pin is activated

Appendix A. Driver Revision History

A.1 BUG FIXES AND CHANGES

TABLE A-1: DRIVER RELEASE HISTORY

Releases	Date	Description
1.4.2	20 August 2021	• Added the PAC193X device driver support for Windows® 11
1.4.1	26 April 2021	• Improved the support for the DFx/RTD3 power state: the sampling rate is set to low sps only if the system power state is changed to a low-power state (S1/S2/S3/S4/S5 hybrid or CS). • The driver ensures that non-supported channels on the devices with less than 4 channels are disabled. The user requests to change the channel's ON/OFF setting for the non-supported channels are ignored. • Included code optimizations to reduce the number of I^2C transfers when entering D0 power state and when reporting the device ID values. • Improved sampling rate configuration code (makes it resilient against potential initialization issue).
1.4.0	28 April 2020	• Added support for Windows® DFx (Directed Power Framework) and RTD3 power state. • Issue fix: Fixed the measurement report time-stamp jump ahead observed after about 26 hours from the machine cold boot. Note: the time-stamp now is reported as a count of 100-nanosecond intervals since January 1, 1601.

TABLE A-1: DRIVER RELEASE HISTORY (CONTINUED)

Releases	Date	Description
1.3	22 May 2019	• Added support for one-channel device – PAC1931. • Added support for microohm Sense Resistors (R_{SENSE}). • Added support for ACPI DSM Rev. 1, which provides additional driver and device initialization options while preserving the backward compatibility with DSM Rev. 0: - R_{SENSE} value expressed as microohm units. - EMI interface enable/disable bit mask. The channels with non-empty EMI names, but disabled EMI interface, are still functional as "private" channels and they keep their name string. So the private channels can be easy to identify now. - Channel polarity bit mask with the same structure and meaning as the device `NEG_PWR` register. - Device "Active" and "Idle" sampling frequency. - Driver Watchdog Timer interval. • Added new IOCTL reporting the control interface revision number – `IOCTL_PAC193X_GET_INTERFACE_REVISION`. • Added new IOCTL reporting the Sense Resistor (R_{SENSE}) values as microohm units. • Restriction removed: Sample frequency can be changed even if the device has open EMI channels (non-private). • Restriction removed: Private channels' status can be changed (enabled/disabled) and the software counters for the private channels can be reset, even if other channels on the device are EMI channels (non-private). • I^2C communication optimization (eliminate redundant commands and reduce the delay between "`REFRESH`" command and the subsequent device data read). • Device power optimization: turn off the channels with 0Ω Sense Resistor (R_{SENSE}). • Consistent use of `_ACT` registers for device operating state checks. • Issue fix concerning bipolar channel data computation: The very first data sample is computed as an unsigned number. • Issue fix – reject the EMI IOCTL requests issued to the control interface. • `IOCTL_PAC193X_DATA_REFRESH_RESET_GLOBAL` control code changed in order to define it as read/write code. • Implemented the software work around for the silicon errata that prevents the reduction of the device internal power consumption between sample processing, affecting the Silicon Revisions 0x02 and 0x03 (Revision ID register 0xFF).

TABLE A-1: DRIVER RELEASE HISTORY (CONTINUED)

Releases	Date	Description
First Public Release	07 September 2017	• Three Latch registers added to the `LAST_SLOW_DATA`. • The `SET_NEG_POWER IOCTL` allows register setting regardless of the presence of public channels on the device. • The `SET_CHANNEL_CFG IOCTL` allows enabling or disabling individual private channels, rather than requiring no public channels on the device. • The ByteCount and Timeout fields can no longer be set by the user with the `SET_CHANNEL_CFG IOCTL`. • TotalAccumulationCount is cached from the previous cycle to properly support `REFRESH_V`. • SoftwareAccumulatorCount is cleared when the clear option is specified to the `REFRESH_RESET IOCTL`. • Single Shot mode is supported. • The polling timer is rearmed only when the hardware accumulators are cleared, not when an IOCTL is received. • Fixed an outstanding issue with resource rebalancing (issue was present during the HLKs). • Added support for the second hardware revision. • `PACTest` accommodates and displays IOCTL changes.

Appendix B. PAC193X Device Control Interface

B.1 INTRODUCTION

PAC193X device control interface created by the PAC193X Windows Device Driver is defined by the example C header files described in this appendix:

- `PAC193x_HW.h`: Contains PAC193X device-specific definitions
- `PAC193x_INTF.h`: Contains the interface and the IOCTL definitions

B.2 PAC193X DEVICE DEFINITIONS

EXAMPLE B-1: `PAC193X_HW.h` HEADER FILE

```
//////////////////////////////////////////////////////////////////////////////
//      (C) 2017-2021 Microchip Technology Inc. and its subsidiaries.
//      (C) Copyright 2016-2017 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:
//       PAC193x_HW.h
//
//    ABSTRACT:
//       Device-Specific Hardware Interface for the Microchip PAC193x EMI Driver
//
//    AUTHOR(S):
//       OSR Open Systems Resources, Inc.
//       Microchip Technology, Inc.
//
//    REVISION:
//       v1.4.2
//
//////////////////////////////////////////////////////////////////////////////
#pragma once

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

EXAMPLE B-1: `PAC193X_HW.h` HEADER FILE (CONTINUED)

```
#define PAC_MAXIMUM_CHANNELS 4

////////////////////////////////////
// DEVICE REGISTER VALUE CONSTANTS //
////////////////////////////////////
#define PAC_PRODUCT_ID_1934         0x5B
#define PAC_PRODUCT_ID_1933         0x5A
#define PAC_PRODUCT_ID_1932         0x59
#define PAC_PRODUCT_ID_1931         0x58

#define PAC_MANUFACTURER_ID         0x5D
#define PAC_REVISION                0x01

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

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

//
// Measurement registers
//
#define PAC_REGISTER_ACC_COUNT          0x02
#define PAC_REGISTER_CH1_ACC_COUNT      0x03
#define PAC_REGISTER_CH2_ACC_COUNT      0x04
#define PAC_REGISTER_CH3_ACC_COUNT      0x05
#define PAC_REGISTER_CH4_ACC_COUNT      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

#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

//
// Various other control registers
//
```

EXAMPLE B-1: `PAC193X_HW.h` HEADER FILE (CONTINUED)

```
#define PAC_REGISTER_CHANNEL_DIS        0x1C
#define PAC_REGISTER_NEG_PWR            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_CHANNEL_DIS_ACT    0x22
#define PAC_REGISTER_NEG_PWR_ACT        0x23
#define PAC_REGISTER_CTRL_LAT           0x24
#define PAC_REGISTER_CHANNEL_DIS_LAT    0x25
#define PAC_REGISTER_NEG_PWR_LAT        0x26

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

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

#pragma push(pack:1)
typedef struct _PAC193X_CTRL_REGISTER {
    UCHAR Overflow             : 1;
    UCHAR OverflowAlert        : 1;
    UCHAR AlertConversion      : 1;
    UCHAR AlertPin             : 1;
    UCHAR SingleShotMode       : 1;
    UCHAR Sleep                : 1;
    UCHAR SampleRateNormalMode : 2;
} PAC193X_CTRL_REGISTER, *PPAC193X_CTRL_REGISTER;

typedef struct _PAC193X_CHANNEL_DIS_ACT_REGISTER {
    UCHAR             : 1;
    UCHAR NoSkip      : 1;
    UCHAR ByteCount   : 1;
    UCHAR Timeout     : 1;
    UCHAR Channel4Off : 1;
    UCHAR Channel3Off : 1;
    UCHAR Channel2Off : 1;
    UCHAR Channel1Off : 1;
} PAC193X_CHANNEL_DIS_REGISTER, *PPAC193X_CHANNEL_DIS_REGISTER;

typedef struct _PAC193X_NEG_PWR_REGISTER {
    UCHAR Channel4BIDV : 1;
    UCHAR Channel3BIDV : 1;
    UCHAR Channel2BIDV : 1;
    UCHAR Channel1BIDV : 1;
    UCHAR Channel4BIDI : 1;
    UCHAR Channel3BIDI : 1;
    UCHAR Channel2BIDI : 1;
    UCHAR Channel1BIDI : 1;
} PAC193X_NEG_PWR_REGISTER, *PPAC193X_NEG_PWR_REGISTER;
```

EXAMPLE B-1: `PAC193X_HW.h` HEADER FILE (CONTINUED)

```
typedef struct _PAC193X_SLOW_REGISTER {
    UCHAR PowerOnReset : 1;
    UCHAR RefreshVFall : 1;
    UCHAR RefreshFall  : 1;
    UCHAR RefreshVRise : 1;
    UCHAR RefreshRise  : 1;
    UCHAR SlowHighLow  : 1;
    UCHAR SlowLowHigh  : 1;
    UCHAR Slow         : 1;
} PAC193X_SLOW_REGISTER, *PPAC193X_SLOW_REGISTER;
#pragma pop(pack)
```

B.3 INTERFACE AND IOCTL DEFINITIONS

EXAMPLE B-2: `PAC193X_INTF.h` HEADER FILE

```
//////////////////////////////////////////////////////////////////////////////
//      (C) 2017-2021 Microchip Technology Inc. and its subsidiaries.
//      (C) Copyright 2016-2017 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:
//       PAC193x_INTF.h
//
//    ABSTRACT:
//       Device-Specific Interface and IOCTL definitions for
//       the Microchip PAC193x EMI Driver
//
//    AUTHOR(S):
//        OSR Open Systems Resources, Inc.
//        Microchip Technology, Inc.
//
//    REVISION:
//        v1.4.2
//
//////////////////////////////////////////////////////////////////////////////

#pragma  once

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

// {4166FE9F-A865-4314-8942-7C12ABA290F6}
// Identifies the control (non-EMI) portion of the interface
//
#define  GUID_PAC193X_CONTROL_INTERFACE_STR L"{4166FE9F-A865-4314-8942-7C12ABA290F6}"
DEFINE_GUID(GUID_PAC193X_CONTROL_INTERFACE,
    0x4166fe9f, 0xa865, 0x4314, 0x89, 0x42, 0x7c, 0x12, 0xab, 0xa2, 0x90, 0xf6);

//
// Register definitions
//
#include "..\Inc\PAC193x_HW.h"

#define  FILE_DEVICE_PAC193X   0x913E
```

EXAMPLE B-2: `PAC193X_INTF.h` HEADER FILE (CONTINUED)

```
//////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_GET_DEVICE_INFO_SIZE
//
// Returns the size allocation requirements for the buffer that must hold
// the device info data structure.
// - input buffer: none
// - output buffer: PAC193X_GET_DEVICE_INFO_SIZE
//////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_GET_DEVICE_INFO_SIZE {
    ULONG BufferSize;
}PAC193X_GET_DEVICE_INFO_SIZE, *PPAC193X_GET_DEVICE_INFO_SIZE;

#define IOCTL_PAC193X_GET_DEVICE_INFO_SIZE CTL_CODE(FILE_DEVICE_PAC193X,\
                                                    1929,\
                                                    METHOD_BUFFERED,\
                                                    FILE_READ_DATA)


//////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_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: PAC193X_GET_DEVICE_INFO
//////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_DEVICE_CHANNEL_INFO {
    ULONG NextChannelInfoOffset;

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

    ULONG RsenseMOhm;                   // Rsense in milliohms
                                        // 0 indicates non-connected channel

    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
} PAC193X_DEVICE_CHANNEL_INFO, *PPAC193X_DEVICE_CHANNEL_INFO;

typedef struct _PAC193X_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
    UCHAR ChannelDescRegister;  // CHANNEL_DIS register value
    ULONG ChannelCount;         // the number of "connected"
                                // channels (Rsense > 0)

    PAC193X_DEVICE_CHANNEL_INFO ChannelInfo;  // there are 2,3 or 4
                                              // "CHANNEL_INFO" structures
                                              // depending on device type:
                                              // PAC1932, PAC1933 or PAC1934
} PAC193X_GET_DEVICE_INFO, *PPAC193X_GET_DEVICE_INFO;

#define IOCTL_PAC193X_GET_DEVICE_INFO      CTL_CODE(FILE_DEVICE_PAC193X,\
                                                    1930,\
                                                    METHOD_BUFFERED,\
                                                    FILE_READ_DATA)
```

EXAMPLE B-2: `PAC193X_INTF.h` HEADER FILE (CONTINUED)

```
////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_GET_CTRL
//
// Returns the content of the device registers CTRL, CTRL_ACT, CTRL_LAT
// - input buffer: none
// - output buffer: PAC193X_GET_CTRL
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_GET_CTRL {
    PAC193X_CTRL_REGISTER ControlRegister;      // CTRL register value
    PAC193X_CTRL_REGISTER ControlAct;           // CTRL_ACT register value
    PAC193X_CTRL_REGISTER ControlLat;           // CTRL_ACT register value
} PAC193X_GET_CTRL, *PPAC193X_GET_CTRL;

#define IOCTL_PAC193X_GET_CTRL             CTL_CODE(FILE_DEVICE_PAC193X,\
                                                    1931,\
                                                    METHOD_BUFFERED,\
                                                    FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_SET_CTRL
//
// Provides user limited control over some device configuration bits from
// CTRL register: SRN, SLEEP and SING. The driver rejects the configuration
// requests if the device has open EMI channels.
// - input buffer: PAC193X_SET_CTRL
// - output buffer: none (zero bytes returned)
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_SET_CTRL {
    PAC193X_CTRL_REGISTER ControlRegister;  // value to be written in
                                             // CTRL register
} PAC193X_SET_CTRL, *PPAC193X_SET_CTRL;

#define IOCTL_PAC193X_SET_CTRL             CTL_CODE(FILE_DEVICE_PAC193X,\
                                                    1932,\
                                                    METHOD_BUFFERED,\
                                                    FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_GET_MEASUREMENTS
//
// Returns the measured values (voltage, power, accumulated power, sample count
// and timestamp.
// - input buffer: none
// - output buffer: PAC193X_MEASUREMENTS
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_MEASUREMENTS {
    ULONG     AccumulatorCount;                 // ACC_COUNT register value

    // unipolar or bipolar registers - section begin
    ULONGLONG AccumulatorOutput[PAC_MAXIMUM_CHANNELS];

                                                // VPOWERn_ACC registers values
    USHORT    VBus[PAC_MAXIMUM_CHANNELS];       // VBUSn register values
    USHORT    VSense[PAC_MAXIMUM_CHANNELS];     // VSENSEn registers values
    USHORT    VBusAvg[PAC_MAXIMUM_CHANNELS];    // VBUSn_AVG registers values
    USHORT    VSenseAvg[PAC_MAXIMUM_CHANNELS];  // VSENSEn_AVG registers
values
    ULONG     VPower[PAC_MAXIMUM_CHANNELS];     // VPOWERn registers values
```

EXAMPLE B-2: `PAC193X_INTF.h` HEADER FILE (CONTINUED)

```
    // 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.

    // REMARK: the driver already implements the workaround for the VPOWER_ACC
    //         and VPOWERN errata which affects the devices with RevisionID = 0x01
    // unipolar or bipolar registers - section end

    // software accumulators - section begin
    // 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 SofwareAccumulatorCount;      // software sample counter value

    ULONGLONG SofwareAccumulator[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 energy is reported in pico-watt-hours (pWh).
                      // Same values are also reported by
                      // EMI_MEASUREMENT_DATA.AbsoluteEnergy

    LONGLONG  SignedSoftwareAccumulator[PAC_MAXIMUM_CHANNELS];
                      // Signed energy accumulator (MSB is sign bit).
                      // Both positive and negative energy values are accumulated.
                      // There is one bit resolution loss due to sign bit but
                      // this allows bipolar energy measurements.
                      // The energy energy is reported in pico-watt-hours (pWh).
    // software accumulators - section end

    PAC193X_NEG_PWR_REGISTER     NegPowerLat;   // NEG_PWR_LAT register value
    PAC193X_CTRL_REGISTER        CtrlLat;       // CTRL_LAT register value
    PAC193X_CHANNEL_DIS_REGISTER ChannelDisLat; // CHANNEL_DIS_LAT register value

    ULONGLONG       Timestamp;      // timestamp of the last measurements
                                    // reported in multiples of 100ns.
                                    // 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
}PAC193X_MEASUREMENTS, *PPAC193X_MEASUREMENTS;

#define IOCTL_PAC193X_GET_MEASUREMENTS     CTL_CODE(FILE_DEVICE_PAC193X,\
                                                    1933,\
                                                    METHOD_BUFFERED,\
                                                    FILE_READ_DATA)
```

EXAMPLE B-2: `PAC193X_INTF.h` HEADER FILE (CONTINUED)

```
////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_GET_CHANNEL_CFG
//
// Returns the content of the I2C/SMB and channel ON/OFF configuration registers:
// CHANNEL_DIS, CHANNEL_DIS_ACT, CHANNEL_DIS_LAT
// - input buffer: none
// - output buffer: PAC193X_GET_CHANNEL_CFG
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_GET_CHANNEL_CFG {
    PAC193X_CHANNEL_DIS_REGISTER ConfigRegister;
                                                // CHANNEL_DIS register value
    PAC193X_CHANNEL_DIS_REGISTER ConfigAct;     // CHANNEL_DIS_ACT register value
    PAC193X_CHANNEL_DIS_REGISTER ConfigLat;     // CHANNEL_DIS_LAT register value
} PAC193X_GET_CHANNEL_CFG, *PPAC193X_GET_CHANNEL_CFG;

#define IOCTL_PAC193X_GET_CHANNEL_CFG      CTL_CODE(FILE_DEVICE_PAC193X,\
                                                    1934,\
                                                    METHOD_BUFFERED,\
                                                    FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_SET_CHANNEL_CFG
//
// Provides user limited control over some configuration bits from CHANNEL_DIS
// register: CH1, CH2, CH3 and CH4.
// Notes:
//  - The driver ignores the requests to change the I2C/SMB configuration bits:
//    TIMEOUT and BYTECOUNT, NOSKIP
//  - The driver ignores the requests to change the pointer skip configuration
//    bit: NOSKIP
//  - The driver ignores the change values for those channels which has
//    open EMI channels.
//
// - input buffer: PAC193X_SET_CHANNEL_CFG
// - output buffer: none (zero bytes returned)
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_GET_SET_CHANNEL_CFG {
    PAC193X_CHANNEL_DIS_REGISTER ConfigRegister;
} PAC193X_SET_CHANNEL_CFG, *PPAC193X_SET_CHANNEL_CFG;

#define IOCTL_PAC193X_SET_CHANNEL_CFG      CTL_CODE(FILE_DEVICE_PAC193X,\
                                                    1935,\
                                                    METHOD_BUFFERED,\
                                                    FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_GET_NEG_POWER
//
// Returns the content of the channel polarity configuration registers NEG_PWR,
// NEG_PWR_ACT, NEG_PWR_LAT
// - input buffer: none
// - output buffer: PAC193X_GET_NEG_POWER
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_GET_NEG_POWER {
    PAC193X_NEG_PWR_REGISTER NegPowerRegister;      // NEG_PWR register value
    PAC193X_NEG_PWR_REGISTER NegPowerAct;           // NEG_PWR_ACT register value
    PAC193X_NEG_PWR_REGISTER NegPowerLat;           // NEG_PWR_LAT register value
} PAC193X_GET_NEG_POWER, *PPAC193X_GET_NEG_POWER;

#define IOCTL_PAC193X_GET_NEG_POWER        CTL_CODE(FILE_DEVICE_PAC193X,\
                                                    1936,\
                                                    METHOD_BUFFERED,\
                                                    FILE_READ_DATA)
```

EXAMPLE B-2: `PAC193X_INTF.h` HEADER FILE (CONTINUED)

```
////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_SET_NEG_POWER
//
// Provides user control over channel polarity configuration register: NEG_PWR.
// Channel polarity is set to unipolar at the driver initialization but can be
// changed by the user at any moment, regardless the EMI status.
// - input buffer: PAC193X_SET_NEG_POWER
// - output buffer: none (zero bytes returned)
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_SET_NEG_POWER {
    PAC193X_NEG_PWR_REGISTER NegPowerRegister;      // value to be written in
                                                    // NEG_PWR register
} PAC193X_SET_NEG_POWER, *PPAC193X_SET_NEG_POWER;

#define IOCTL_PAC193X_SET_NEG_POWER        CTL_CODE(FILE_DEVICE_PAC193X,\
                                                    1937,\
                                                    METHOD_BUFFERED,\
                                                    FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_GET_OVERFLOW
//
// Returns the status of the OVF bit from CTRL_ACT and CTRL_LAT.
// Depending on the value of the input flag, this request can also clear
// (reset to 0) the OVF flag from CTRL register.
// - input buffer: PAC193X_GET_OVERFLOW
// - output buffer: PAC193X_GET_OVERFLOW
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_GET_OVERFLOW {
    UCHAR OvfClear;     // In: the user set nonzero value to request the driver
                        // to clear the OVF flag (in device CTRL register)

    UCHAR OvfActual;    // Out: the IOCTL_PAC193X_GET_OVERFLOW driver returns in
                        // the LSB the value of the OVF bit from CTRL_ACT

    UCHAR OvfLatch;     // Out: the IOCTL_PAC193X_GET_OVERFLOW driver returns in
                        // the LSB the value of the OVF bit from CTRL_ACT
} PAC193X_GET_OVERFLOW, *PPAC193X_GET_OVERFLOW;

#define IOCTL_PAC193X_GET_OVERFLOW      CTL_CODE(FILE_DEVICE_PAC193X,\
                                                 1938,\
                                                 METHOD_BUFFERED,\
                                                 FILE_READ_DATA | FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_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_PAC193X_DATA_REFRESH         CTL_CODE(FILE_DEVICE_PAC193X,\
                                                    1939,\
                                                    METHOD_BUFFERED,\
                                                    FILE_READ_DATA)
```

EXAMPLE B-2: `PAC193X_INTF.h` HEADER FILE (CONTINUED)

```
////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_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: PAC193X_REFRESH_FLAGS
// - output buffer: none (zero bytes returned)
////////////////////////////////////////////////////////////////////////////
#define PAC193X_REFRESH_FLAG_RESET_ACCUMULATORS (1 << 0)
typedef struct _PAC193X_REFRESH_FLAGS {
    ULONG Flags;    // In: the user set the value of this field to
                    // PAC193X_REFRESH_FLAG_RESET_ACCUMULATORS (1) in order to
                    // request the driver to clear the software accumulators
                    // - SofwareAccumulatorCount
                    // - SofwareAccumulator[PAC_MAXIMUM_CHANNELS]
                    // - SignedSoftwareAccumulator[PAC_MAXIMUM_CHANNELS]
                    //
                    // CAUTION: SofwareAccumulator and SignedSoftwareAccumulator
                    //          for the EMI channels are not reset
} PAC193X_REFRESH_FLAGS, *PPAC193X_REFRESH_FLAGS;

#define IOCTL_PAC193X_DATA_REFRESH_RESET  CTL_CODE(FILE_DEVICE_PAC193X,\
                                                   1940,\
                                                   METHOD_BUFFERED,\
                                                   FILE_READ_DATA | FILE_WRITE_DATA)


////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_DATA_REFRESH_RESET_GLOBAL
//
// Performs a global update of the software accumulators for all the PAC193x
// 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: PAC193X_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_PAC193X_DATA_REFRESH_RESET_GLOBAL  CTL_CODE(FILE_DEVICE_PAC193X,\
                                                          1941,\
                                                          METHOD_BUFFERED,\
                                                          FILE_READ_DATA | FILE_WRITE_DATA)
                                                          //in the interface revision 0x01 was:
                                                          //FILE_READ_DATA )
```

EXAMPLE B-2: `PAC193X_INTF.h` HEADER FILE (CONTINUED)

```
////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_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: PAC193X_LAST_SLOW_DATA
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_LAST_SLOW_DATA {
    ULONG       AccumulatorCount;               // ACC_COUNT register value

    // unipolar or bipolar registers - section begin
    ULONGLONG   AccumulatorOutput[PAC_MAXIMUM_CHANNELS];
                                                // VPOWERn_ACC registers values
    USHORT      VBus[PAC_MAXIMUM_CHANNELS];     // VBUSn register values
    USHORT      VSense[PAC_MAXIMUM_CHANNELS];   // VSENSEn registers values
    USHORT      VBusAvg[PAC_MAXIMUM_CHANNELS];  // VBUSn_AVG 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.
    //
    // REMARK: the driver already implements the workaround for the VPOWER_ACC
    //         and VPOWERN errata which affects the devices with
    //         Revision ID = 0x01
    // unipolar or bipolar registers - section end

    PAC193X_NEG_PWR_REGISTER     NegPowerLat;   // NEG_PWR_LAT register value
    PAC193X_CTRL_REGISTER        CtrlLat;       // CTRL_LAT register value
    PAC193X_CHANNEL_DIS_REGISTER ChannelDisLat; // CHANNEL_DIS_LAT register value
}PAC193X_LAST_SLOW_DATA, *PPAC193X_LAST_SLOW_DATA;

#define IOCTL_PAC193X_GET_LAST_SLOW_DATA  CTL_CODE(FILE_DEVICE_PAC193X,\
                                                   1942,\
                                                   METHOD_BUFFERED,\
                                                   FILE_READ_DATA)


////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_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: PAC193X_GET_SLOW
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_GET_SLOW {
    PAC193X_SLOW_REGISTER SlowRegister;     // SLOW register value
} PAC193X_GET_SLOW, *PPAC193X_GET_SLOW;

#define IOCTL_PAC193X_GET_SLOW            CTL_CODE(FILE_DEVICE_PAC193X,\
                                                   1943,\
                                                   METHOD_BUFFERED,\
                                                   FILE_READ_DATA)
```

EXAMPLE B-2: `PAC193X_INTF.h` HEADER FILE (CONTINUED)

```
////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_GET_INTERFACE_REVISION
//
// Returns the driver private interface revision number.
//
// - input buffer: none
// - output buffer: PAC193X_GET_INTF_REV
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_GET_INTF_REV {
    USHORT PrivIntfRev;     // private interface revision value
} PAC193X_GET_INTF_REV, *PPAC193X_GET_INTF_REV;

#define IOCTL_PAC193X_GET_INTERFACE_REVISION        CTL_CODE(FILE_DEVICE_PAC193X,\
                                                             1944,\
                                                             METHOD_BUFFERED,\
                                                             FILE_READ_DATA)

////////////////////////////////////////////////////////////////////////////
// IOCTL_PAC193X_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: PAC193X_GET_RSENSE_MICRO
////////////////////////////////////////////////////////////////////////////
typedef struct _PAC193X_GET_RSENSE_MICRO {
    ULONG RsenseCH1;    //micro-ohms
    ULONG RsenseCH2;    //micro-ohms
    ULONG RsenseCH3;    //micro-ohms
    ULONG RsenseCH4;    //micro-ohms
} PAC193X_GET_RSENSE_MICRO, *PPAC193X_GET_RSENSE_MICRO;

#define IOCTL_PAC193X_GET_RSENSE_MICRO      CTL_CODE(FILE_DEVICE_PAC193X,\
                                                     1945,\
                                                     METHOD_BUFFERED,\
                                                     FILE_READ_DATA)
```

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