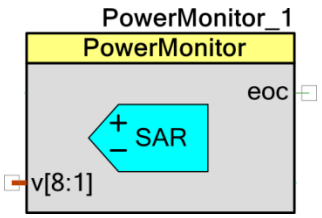
Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

PSoC 4 Power Monitor

2.20



Features

- Interfaces to up to 8 DC-DC power converters
- Measures power converter output voltages using a SAR ADC
- Monitors the health of the power converters generating warnings and faults based on user-defined thresholds

General Description

Power Converter Voltage Measurements

For power converter voltage measurements, the ADC shall be configured into single-ended mode. The ADC can also be configured to use bypass capacitor or not. In cases where the analog voltage to be monitored equals or exceeds Vdda or the ADC range, external resistor dividers are recommended to scale the monitored voltages down to an appropriate range.

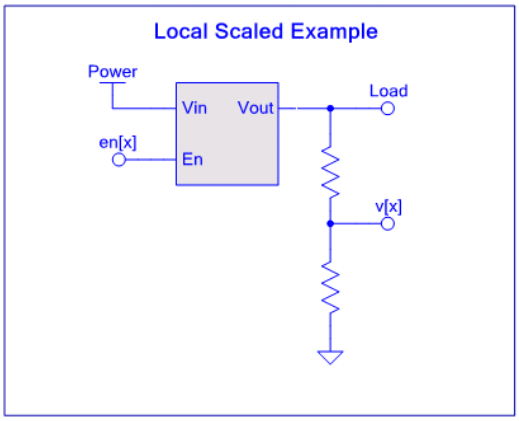
ADC Sequential Scanning

The ADC sequences through all power converters, if enabled, in a round-robin fashion, taking voltage measurements. This Component measures the voltages of all the power converters in the system.

This Component needs some knowledge of Components external to the PSoC device, because scaling factors for input voltages that have been attenuated to meet IO input range limits or ADC dynamic range limits where applicable.

When to Use a Power Monitor

The following diagram shows the connection methodology for a power converter that has an output voltage higher than the range of the ADC. The voltage sense point is scaled down to a voltage level that can directly connect to this Component.



If the power converter has an output voltage in the range of the ADC, there is no need to use the divider circuit, simply connect the power converter output directly to the PSoC input pin.

Input/Output Connections

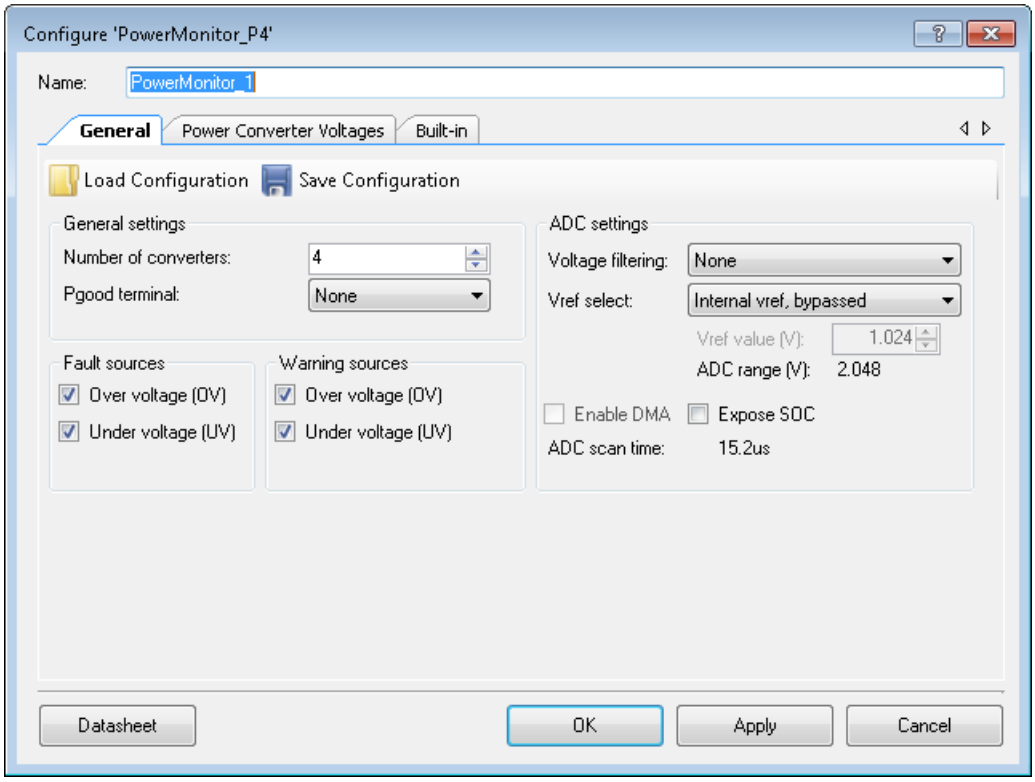
This section describes the various input and output connections for the Power Monitor Component. An asterisk (*)in the following list indicates that it may not be shown on the Component symbol for the conditions listed in the description of that I/O.

Terminal Name	I/O Type	Description
v[x]	Analog Input x x = 1…8	Analog inputs that connect to the power converter output voltage as seen by their loads. This could be a direct connection to the power converter output, or a scaled version using external scaling resistors. Every power converter has voltage measurement enabled. The Component supports a maximum of 8 voltage input terminal pins.
eoc	Digital Output	This terminal is directly connected to the SAR ADC “eoc” terminal.
*pgood	Digital Output	This digital output terminal is driven active high when all power converter voltages and currents (if measured) are within a user specified fault range. An option exists in the customizer to make this terminal a bus to expose the individual pgood status outputs for each converter. The terminals are updated on calling GetPgoodStatus() API. This terminal is hidden if the part does not support UDBs.

Component Parameters

Drag a Power Monitor Component onto your design and double-click it to open the Configure dialog. This dialog has the following tabs with different parameters.

General Tab



At the top of the tab, there are the following buttons:

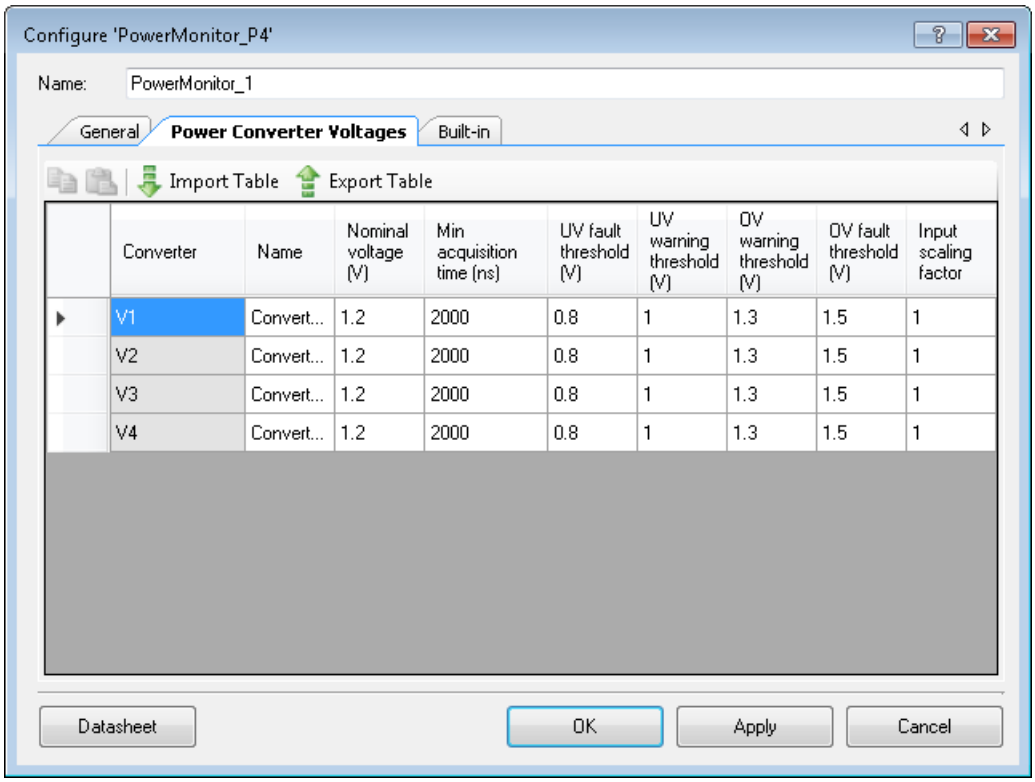
- Load Configuration: Opens a dialog to find and load a previously saved configuration file.
- Save Configuration: Opens a dialog to save this configuration into a file.

The following table provides descriptions of the various **General** tab parameters.

Parameter Name	Description
Number of Converters	Sets the number of converters to be monitored.
Pgood terminal	Pulldown to select pgood terminal configuration. Global: single terminal Individual: bus terminal None: no terminal This option is only available if the PSoC supports UDBs.
Fault Sources	List of check boxes to set the under-voltage (UV) and over voltage (OV) fault sources.
Warning Sources	List of check boxes to set the under-voltage (UV) and over voltage (OV) warning sources.
Voltage filtering	Sets the averaging rate for all measurements. The fixed resolution mode is used.
Vref select	Pulldown to select the reference voltage value that is used for the SAR ADC reference. If “Internal Vref, bypassed” is selected, a pin is reserved to connect an external bypass capacitor. If External Vref is selected, an input field appears to set the Vref value. A pin is also reserved to connect the external Vref. Valid Vref input values are between 1 V and Vdda. It is recommended to use a Vref that is less than ½ of Vdda.
ADC range	The ADC Range is calculate using the following formula: Range = 0.0 V to 2*Vref If the internal Vref is used: • Vref = 1.2 V for PSoC 4100S/PSoC 4100S Plus devices • Vref = 1.024 V for all other devices If “External Vref” is used, the range depends on the input value provided in the customizer. The maximum range is 0.0 V to Vdda.
Enable DMA	Check box to select the usage of the DMAs to reduce the overhead in the CPU. This option is only available in devices that support both DMAs and UDBs.
Enable SOC	Exposes the SOC input to trigger a start of conversion.
Max Response time	Total scanning rate in µs. Absolute time depends on the number of rails, samples averaged and clock input frequency.

Power Converter Voltages Tab

The **Power Converter Voltages** tab sets the voltage measurements configuration for the Component.



At the top of the tab, there are the following buttons:

- Copy: copy one converter configuration from the table
- Paste: paste one converter configuration in the table
- Import Table: imports data from file to table cells on active tab. Supports .csv file format.
- Export Table: exports data from table cells of active tab to file. Supports .csv file format.

The following table provides descriptions of the various **Power Converter Voltages** tab parameters.

Parameter Name	Description
Name	Sets a text field to give the name for the power converter. This is used only for annotation purposes. The maximum allowed characters are 16.
Nominal voltage	Sets the nominal converter output voltage. This is used only for annotation purposes

Parameter Name	Description
Min acquisition time	Sets the minimum acquisition time needed for the input signal to drive the input capacitor and settle to the desired resolution. This parameter is linked to the SAR ADC “Min Acq Time” parameter.
UV fault threshold	Sets the under voltage fault threshold for the specified power converter. This number multiplied by the scaling factor should be smaller than the maximum voltage range of the ADC.
UV warning threshold	Sets the under voltage warning threshold for the specified power converter. This number multiplied by the scaling factor should be smaller than the maximum voltage range of the ADC.
OV warning threshold	Sets the over voltage warning threshold for the specified power converter. This number multiplied by the scaling factor should be smaller than the maximum voltage range of the ADC.
OV fault threshold	Sets the over voltage fault threshold for the specified power converter. This number multiplied by the scaling factor should be smaller than the maximum voltage range of the ADC.
Input scaling factor	Sets the input voltage scaling factor for the specified power converter. This scaling factor indicates the amount of attenuation applied to the converter output voltage external to the PSoC.

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the Component using software. The following sections list and describe each function and dependencies.

API functions allow configuration of the Component using the CPU.

By default, PSoC Creator assigns the instance name **PowerMonitor_P4_1** to the first instance of a Component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is **PowerMonitor_P4**.

Modules

- Initialization
  General APIs are used for initialization of the Component with defaults taken from the Component customizer parameters.
- General
  General APIs are used for run-time configuration of the Component during active power mode.
- Interrupts
  Interrupt accessors.

Initialization

Description

General APIs are used for initialization of the Component with defaults taken from the Component customizer parameters.

These include starting and stopping the Component.

Functions

- void PowerMonitor_P4_Start(void)
  Enables the Component. Calls PowerMonitor_P4_Init() if the Component has not been initialized before. Calls PowerMonitor_P4_Enable().
- void PowerMonitor_P4_Stop(void)
  Disables the Component. ADC sampling stops.
- void PowerMonitor_P4_Init(void)
  Initialize the Component.
- void PowerMonitor_P4_Enable(void)
  Enables hardware blocks within the Component and starts scanning. The PowerMonitor_P4_Start or PowerMonitor_P4_Init function must be called prior to calling this function.

Function Documentation

void PowerMonitor_P4_Start (void)

Enables the Component. Calls PowerMonitor_P4_Init() if the Component has not been initialized before. Calls PowerMonitor_P4_Enable().

void PowerMonitor_P4_Stop (void)

Disables the Component. ADC sampling stops.

void PowerMonitor_P4_Init (void)

Initialize the Component.

Side Effects

Clears all pending fault and warning status and resets all thresholds to values from the customizer settings. All faults are disabled.

void PowerMonitor_P4_Enable (void)

Enables hardware blocks within the Component and starts scanning. The PowerMonitor_P4_Start or PowerMonitor_P4_Init function must be called prior to calling this function.

General

Description

General APIs are used for run-time configuration of the Component during active power mode.

These include reading from registers, and writing to registers.

Macros

- #define **PowerMonitor_P4_GetSamplesAveraged**() (PowerMonitor_P4_samplesAveraged)

Functions

- void PowerMonitor_P4_EnableFault(uint32 converterMask)
  Enables fault detection for the desired converters. When a fault happens in the specific converter, the Component stops scanning faults for that converter until it is re-enabled by calling this API.
- void PowerMonitor_P4_DisableFault(uint32 converterMask)
  Disables UV fault detection for the desired converters.
- void PowerMonitor_P4_SetFaultMode(uint32 converterNum, uint32 faultMode)
  Configures fault sources for each power converter. Two fault sources are available: OV and UV. The fault sources defined in the customizer set the initial fault source values. Updated fault sources are not active until API PowerMonitor_P4_EnableFault() is called.
- uint32 PowerMonitor_P4_GetFaultMode(uint32 converterNum)
  Returns enabled fault sources from the Component.
- uint32 PowerMonitor_P4_GetPGoodStatus(void)
  Returns the pgood status of all converters. Each bit represents the pgood of one converter.
- uint32 PowerMonitor_P4_GetFaultSource(void)
  Returns pending fault sources from the Component where each bit represents a converter. This API can be used to poll the fault status of the Component. When this API returns a non-zero value, the PowerMonitor_P4_GetOVFaultStatus(), PowerMonitor_P4_GetUVFaultStatus() APIs can provide further information on which power converter(s) caused the fault. The fault source bits are cleared by calling the relevant Get Fault Status APIs.
- uint32 PowerMonitor_P4_GetOVFaultStatus(void)
  Return over voltage fault status of each power converter where each bit represents a converter. The fault bits are cleared on read. Once a fault is detected, the Component will no longer update the fault status for the faulty converter until PowerMonitor_P4_EnableFault API is called to re-enable fault detection on the specific converter.
- uint32 PowerMonitor_P4_GetUVFaultStatus(void)
  Return under voltage fault status of each power converter where each bit represents a converter. The fault bits are cleared on read. Once a fault is detected, the Component will not longer update the fault status for the faulty converter until PowerMonitor_P4_EnableFault API is called to re-enable fault detection on the specific converter.
- uint32 PowerMonitor_P4_GetOVWarnStatus(void)
  Returns over voltage warning status of each power converter where each bit represents a converter. The warning status is updated on calling this API. No background task updates the warning bits. The intent is to optimize the fault detection as the warning status is not called frequently.
- uint32 PowerMonitor_P4_GetUVWarnStatus(void)
  Returns under voltage warning status of each power converter where each bit represents a converter. The warning status is updated on calling this API. No background task updates the warning bits. The intent is to optimize the fault detection as the warning status is not called frequently.

- cystatus PowerMonitor_P4_SetUVWarnThreshold(uint32 converterNum, uint32 uvWarnThreshold_mV)
  Sets the power converter under voltage warning threshold for the specified power converter.
- uint32 PowerMonitor_P4_GetUVWarnThreshold(uint32 converterNum)
  Returns the power converter under voltage warning threshold for the specified power converter in mV.
- cystatus PowerMonitor_P4_SetOVWarnThreshold(uint32 converterNum, uint32 ovWarnThreshold_mV)
  Sets the power converter over voltage warning threshold for the specified power converter.
- uint32 PowerMonitor_P4_GetOVWarnThreshold(uint32 converterNum)
  Returns the power converter over voltage warning threshold for the specified power converter in mV.
- cystatus PowerMonitor_P4_SetUVFaultThreshold(uint32 converterNum, uint32 uvFaultThreshold_mV)
  Sets the power converter under voltage fault threshold for the specified power converter.
- uint32 PowerMonitor_P4_GetUVFaultThreshold(uint32 converterNum)
  Returns the power converter under voltage fault threshold for the specified power converter in mV.
- cystatus PowerMonitor_P4_SetOVFaultThreshold(uint32 converterNum, uint32 ovFaultThreshold_mV)
  Sets the power converter over voltage fault threshold for the specified power converter.
- uint32 PowerMonitor_P4_GetOVFaultThreshold(uint32 converterNum)
  Returns the power converter over voltage fault threshold for the specified power converter in mV.
- uint32 PowerMonitor_P4_GetConverterVoltage(uint32 converterNum)
  Returns the converter output voltage for the specified power converter in mV.
- cystatus PowerMonitor_P4_SetSamplesAveraged(uint32 numSample)
  Set number of samples taken for averaged channels. If a number that is not a power of 2 is provided a CYRET_BAD_PARAM value is returned.
- void PowerMonitor_P4_SetFaultCallbackFunction(PowerMonitor_P4_callback func)
  Returns the samples averaged configured in the customizer or by the PowerMonitor_P4_SetSamplesAveraged() API.

Function Documentation

void PowerMonitor_P4_EnableFault (uint32 *converterMask*)

Enables fault detection for the desired converters. When a fault happens in the specific converter, the Component stops scanning faults for that converter until it is re-enabled by calling this API.

Specifically which fault sources are enabled is configured using the PowerMonitor_P4_SetFaultMode().

Only OV faults (if set as a fault source) are automatically enabled by PowerMonitor_P4_Init. Calling this API enables checking for all configured faults.

Parameters:

converterMask	Mask that sets which converters should be checked for faults. Each bit is linked to one converter. If the bit is not set, the current configuration is kept.

Side Effects

Enabled fault settings are updated even when Component is stopped.

void PowerMonitor_P4_DisableFault (uint32 *converterMask*)

Disables UV fault detection for the desired converters.

To disable OV fault detection use PowerMonitor_P4_SetFaultMode(). This ensures that OV faults will still be detected after calling PowerMonitor_P4_DisableFault().

Parameters:

converterMask	Mask that sets which converters should be disabled for faults. Each bit is linked to one converter. If the bit is not set, the current configuration is kept.

Side Effects

Disabled fault settings are updated even when Component is stopped.

void PowerMonitor_P4_SetFaultMode (uint32 *converterNum*, uint32 *faultMode*)

Configures fault sources for each power converter. Two fault sources are available: OV and UV. The fault sources defined in the customizer set the initial fault source values. Updated fault sources are not active until API PowerMonitor_P4_EnableFault() is called.

Parameters:

converterNum	Specifies the converter number. – 1 = Converter 1, – 2 = Converter 2, – ... – 8 = Converter 8
faultMode	- Bit 0: Enable OV Fault – Bit 1: Enable UV Fault

Side Effects

Fault mode settings are updated even when Component is stopped.

uint32 PowerMonitor_P4_GetFaultMode (uint32 *converterNum*)

Returns enabled fault sources from the Component.

- Bit0: OV fault source
- Bit1: UV fault source

Parameters:

converterNum	Specifies the converter number.

Returns:

uint32

Side Effects

Returns zero if converterNum is an invalid value.

uint32 PowerMonitor_P4_GetPGoodStatus (void)

Returns the pgood status of all converters. Each bit represents the pgood of one converter.

The pgood status and the pgood terminal are updated when this API is called.

The pgood status of a converter is a combination of PowerMonitor_P4_faultEnable[] state, set in the GUI or the PowerMonitor_P4_SetFaultMode() API, and the voltage of the converter.

If the OV and UV faults are not enabled in PowerMonitor_P4_faultEnable[] then pgood is always high.

If only UV faults are enabled then pgood is high when the converter's voltage is above the UV limit.

If only OV faults are enabled then pgood is high when the converter's voltage is below the OV limit.

If both OV and UV faults are enabled then pgood is high when the converter's voltage is between the OV and UV limits.

Note that enabling/disabling OV/UV detection through the PowerMonitor_P4_EnableFault() and PowerMonitor_P4_DisableFault() APIs do not affect the pgood status.

Returns:

uint32

Side Effects

The pgood terminal values are updated with the values returned by this API. Returns 0 if Component is disabled.

uint32 PowerMonitor_P4_GetFaultSource (void)

Returns pending fault sources from the Component where each bit represents a converter. This API can be used to poll the fault status of the Component. When this API returns a non-zero value, the PowerMonitor_P4_GetOVFaultStatus(), PowerMonitor_P4_GetUVFaultStatus() APIs can provide further information on which power converter(s) caused the fault. The fault source bits are cleared by calling the relevant Get Fault Status APIs.

If DMA is not used, the fault status is updated on calling this API. This API has the same behavior as the DMA, performing the equivalent OV and UV comparison in firmware rather than hardware.

Returns:

uint32

Side Effects

Fault sources are cleared by the Stop API.

uint32 PowerMonitor_P4_GetOVFaultStatus (void)

Return over voltage fault status of each power converter where each bit represents a converter. The fault bits are cleared on read. Once a fault is detected, the Component will no longer update the fault status for the faulty converter until PowerMonitor_P4_EnableFault API is called to re-enable fault detection on the specific converter.

Returns:

32-bit status indicating the over voltage fault status of each power converter.

uint32 PowerMonitor_P4_GetUVFaultStatus (void)

Return under voltage fault status of each power converter where each bit represents a converter. The fault bits are cleared on read. Once a fault is detected, the Component will not longer update the fault status for the faulty converter until PowerMonitor_P4_EnableFault API is called to re-enable fault detection on the specific converter.

Returns:

32-bit status indicating the under voltage fault status of each power converter.

uint32 PowerMonitor_P4_GetOVWarnStatus (void)

Returns over voltage warning status of each power converter where each bit represents a converter. The warning status is updated on calling this API. No background task updates the warning bits. The intent is to optimize the fault detection as the warning status is not called frequently.

Returns:

32-bit status indicating the over voltage warning status of each power converter.

uint32 PowerMonitor_P4_GetUVWarnStatus (void)

Returns under voltage warning status of each power converter where each bit represents a converter. The warning status is updated on calling this API. No background task updates the warning bits. The intent is to optimize the fault detection as the warning status is not called frequently.

Returns:

32-bit value indicating under voltage warning status for each power converter.

cystatus PowerMonitor_P4_SetUVWarnThreshold (uint32 *converterNum*, uint32 *uvWarnThreshold_mV*)

Sets the power converter under voltage warning threshold for the specified power converter.

Parameters:

converterNum	Specifies the converter number
uvWarnThreshold_mV	Specifies the under voltage warning threshold in mV. – Min value = 1 mV. – Max value = 65535 mV.

Returns:

cystatus CYRET_SUCCESS or CYRET_BAD_PARAM

uint32 PowerMonitor_P4_GetUVWarnThreshold (uint32 *converterNum*)

Returns the power converter under voltage warning threshold for the specified power converter in mV.

Parameters:

converterNum	Specifies the converter number

Returns:

32-bit value indicating the under voltage threshold in mV. Return zero if converterNum is more than the number of converters.

cystatus PowerMonitor_P4_SetOVWarnThreshold (uint32 *converterNum*, uint32 *ovWarnThreshold_mV*)

Sets the power converter over voltage warning threshold for the specified power converter.

Parameters:

converterNum	Specifies the converter number
ovWarnThreshold_mv	Specifies the over voltage warning threshold in mV. – Min value = 1 mV. – Max value = 65535 mV.

Returns:

cystatus CYRET_SUCCESS or CYRET_BAD_PARAM

uint32 PowerMonitor_P4_GetOVWarnThreshold (uint32 *converterNum*)

Returns the power converter over voltage warning threshold for the specified power converter in mV.

Parameters:

converterNum	Specifies the converter number

Returns:

32-bit value indicating the over voltage threshold in mV.

cystatus PowerMonitor_P4_SetUVFaultThreshold (uint32 *converterNum*, uint32 *uvFaultThreshold_mV*)

Sets the power converter under voltage fault threshold for the specified power converter.

Parameters:

converterNum	Specifies the converter number UVFaultThreshold_mV Specifies the under voltage fault threshold in mV. – Min value = 1vmV. – Max = 65535 mV.

Returns:

cystatus CYRET_SUCCESS or CYRET_BAD_PARAM

uint32 PowerMonitor_P4_GetUVFaultThreshold (uint32 *converterNum*)

Returns the power converter under voltage fault threshold for the specified power converter in mV.

Parameters:

converterNum	Specifies the converter number

Returns:

32-bit value indicating the under voltage fault threshold in mV.

cystatus PowerMonitor_P4_SetOVFaultThreshold (uint32 *converterNum*, uint32 *ovFaultThreshold_mV*)

Sets the power converter over voltage fault threshold for the specified power converter.

Parameters:

converterNum	Specifies the converter number
OVFaultThreshold_mV	Specifies the over voltage fault threshold in mV. – Min value = 1 mV. – Max value = 65535 mV.

Returns:

cystatus CYRET_SUCCESS or CYRET_BAD_PARAM

uint32 PowerMonitor_P4_GetOVFaultThreshold (uint32 *converterNum*)

Returns the power converter over voltage fault threshold for the specified power converter in mV.

Parameters:

converterNum	Specifies the converter number.

Returns:

32-bit value indicating the over voltage fault threshold in mV.

uint32 PowerMonitor_P4_GetConverterVoltage (uint32 *converterNum*)

Returns the converter output voltage for the specified power converter in mV.

Parameters:

converterNum	Specifies the converter number.

Returns:

Output voltage for the specified converter in mV. Valid range is 1 - 65535. (uint32)

cystatus PowerMonitor_P4_SetSamplesAveraged (uint32 *numSample*)

Set number of samples taken for averaged channels. If a number that is not a power of 2 is provided a CYRET_BAD_PARAM value is returned.

Parameters:

numSample	1, 2, 4, 8, 16, 32, 64, 128, 256

Returns:

cystatus CYRET_SUCCESS or CYRET_BAD_PARAM

void PowerMonitor_P4_SetFaultCallbackFunction (PowerMonitor_P4_callback *func*)

Returns the samples averaged configured in the customizer or by the PowerMonitor_P4_SetSamplesAveraged() API.

Returns:

Number of samples averaged.

Set the fault callback function. The callback function is called every time a fault happens. The callback function requires two arguments:

– typedef (*PowerMonitor_P4_callback_func) (uint32 converterNum, uint32 fault_type)

Parameters:

func	fault callback function

Interrupts

Description

Interrupt accessors.

Functions

- **CY_ISR_PROTO** (PowerMonitor_P4_Uv_ISR_Handler)
- **CY_ISR_PROTO** (PowerMonitor_P4_Ov_ISR_Handler)
- **CY_ISR_PROTO** (PowerMonitor_P4_ADC_DMA_ISR)

Global Variables

Description

Global variables used in the Component.

The following global variables are used in the Component.

Globals are noted in the description of the functions that use globals.

Macros

- #define PowerMonitor_P4_NUM_CONVERTERS (`$NumConverters`u)
- #define PowerMonitor_P4_ADC_MAX_VOLTAGE (`$AdcRange_mV`)
- #define PowerMonitor_P4_ADC_MAX_COUNTS ((uint32)(1uL << PowerMonitor_P4_ADC_MAX_RESOLUTION) - 1uL)

Variables

- uint32 PowerMonitor_P4_initVar
- uint16 PowerMonitor_P4_UVFaultThreshold_Count[(`$NumConverters`u)]
- uint16 PowerMonitor_P4_OVFaultThreshold_Count[(`$NumConverters`u)]
- uint16 PowerMonitor_P4_UVWarnThreshold_Count[(`$NumConverters`u)]
- uint16 PowerMonitor_P4_OVWarnThreshold_Count[(`$NumConverters`u)]
- const uint32 PowerMonitor_P4_VoltageScalingFactor[(`$NumConverters`u)]
- const uint32 PowerMonitor_P4_CountScalingFactor[(`$NumConverters`u)]
- uint8 PowerMonitor_P4_faultEnable[(`$NumConverters`u)]
- volatile uint32 PowerMonitor_P4_UVWarnStatus
- volatile uint32 PowerMonitor_P4_OVWarnStatus
- volatile uint32 PowerMonitor_P4_UVFaultStatus
- volatile uint32 PowerMonitor_P4_OVFaultStatus
- volatile uint32 PowerMonitor_P4_PGoodStatus
- volatile uint8 PowerMonitor_P4_faultConfig[(`$NumConverters`u)]

Macro Definition Documentation

#define PowerMonitor_P4_NUM_CONVERTERS (`$NumConverters`u)

Number of converters to be monitored. Range 1 to 8.

#define PowerMonitor_P4_ADC_MAX_VOLTAGE (`$AdcRange_mV`)

ADC maximum voltage in mV. Depends on the range chosen in the customizer.

#define PowerMonitor_P4_ADC_MAX_COUNTS ((uint32)(1uL << PowerMonitor_P4_ADC_MAX_RESOLUTION) - 1uL)

ADC maximum number of counts. Depends on the resolution of the ADC.

Variable Documentation

uint32 PowerMonitor_P4_initVar

Indicates whether the PowerMonitor has been initialized.

uint16 PowerMonitor_P4_UVFaultThreshold_Count[(`$NumConverters`u)]

Array containing the UV fault threshold translated to ADC counts.

uint16 PowerMonitor_P4_OVFaultThreshold_Count[(`$NumConverters`u)]

Array containing the OV fault threshold translated to ADC counts.

uint16 PowerMonitor_P4_UVWarnThreshold_Count[(`$NumConverters`u)]

Array containing the UV warning threshold translated to ADC counts.

uint16 PowerMonitor_P4_OVWarnThreshold_Count[(`$NumConverters`u)]

Array containing the OV warning threshold translated to ADC counts.

const uint32 PowerMonitor_P4_VoltageScalingFactor[(`$NumConverters`u)]

Array containing the voltage scaling factors. It is calculated in the following format:

Y = Scale Factor x 2^16 x ADC Max Counts / ADC Max Voltage (mV)

const uint32 PowerMonitor_P4_CountScalingFactor[(`$NumConverters`u)]

Array containing the count scaling factors. It should be calculated in the following format: Y = 2^16 / Scale Factor ADC Max Voltage (mV) / ADC Max Counts

uint8 PowerMonitor_P4_faultEnable[(`$NumConverters`u)]

Indicates what kind of faults are currently enabled for each of the converters.

volatile uint32 PowerMonitor_P4_UVWarnStatus

Holds the under voltage warning status for each of the power converters.

volatile uint32 PowerMonitor_P4_OVWarnStatus

Holds the over voltage warning status for each of the power converters.

volatile uint32 PowerMonitor_P4_UVFaultStatus

Holds the under voltage fault status for each of the power converters.

volatile uint32 PowerMonitor_P4_OVFaultStatus

Holds the over voltage fault status for each of the power converters.

volatile uint32 PowerMonitor_P4_PGoodStatus

Holds the PGood Status for each of the power converter.

volatile uint8 PowerMonitor_P4_faultConfig[(`$NumConverters`u)]

Internal register controlling which faults are active.

Interrupt Service Routine

The Power Monitor Interrupt Service Routine (ISR) is only available if DMAs are used. The ISR disables fault detection to the faulty rail that triggered the interrupt. It then services the user defined callback function described below.

The ISR of the SAR ADC is used to start the DMA chain. This ensures that the SAR has valid samples before the DMA transfers them to the datapath comparator. This interrupt is triggered once and then disabled.

Macro Callbacks

Macro callbacks allow users to execute code from the API files that are automatically generated by PSoC Creator. Refer to the PSoC Creator Help and *Component Author Guide* for the more details.

In order to add code to the macro callback present in the Component's generated source files, perform the following:

- Create a function of type PowerMonitor_callback as defined in PowerMonitor.h.
- Write the function implementation (in any user file).
- Call the PowerMonitor_SetFaultCallbackFunction() API, passing the defined callback function's name.

API Memory Usage

The Component memory usage varies significantly depending on the compiler, device, number of APIs used and Component configuration. The following table provides the memory usage for all APIs available in the given Component configuration.

The measurements have been done with an associated compiler configured in Release mode with optimization set for Size. For a specific design, the map file generated by the compiler can be analyzed to determine the memory usage.

PSoC 4 (GCC)

Configuration	Flash Bytes	SRAM Bytes
8 Converters with DMA	4568	464
8 Converters without DMA	3144	328

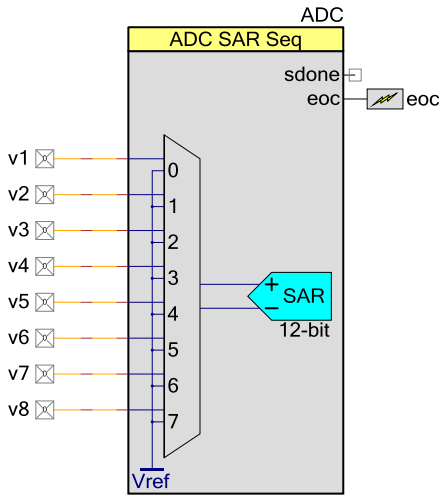
Functional Description

The Power Monitor Component is intended for designers of Power Supervisors who want to quickly and easily design a full-featured power monitor without having to learn the low-level details of PSoC's analog subsystem, manually setting up and configuring the ADC, configuring analog input multiplexers or worrying about calibration issues. Users can configure exactly the functionality they need for their application graphically using the Component customizer. The Component will take care of the implementation details for you automatically.

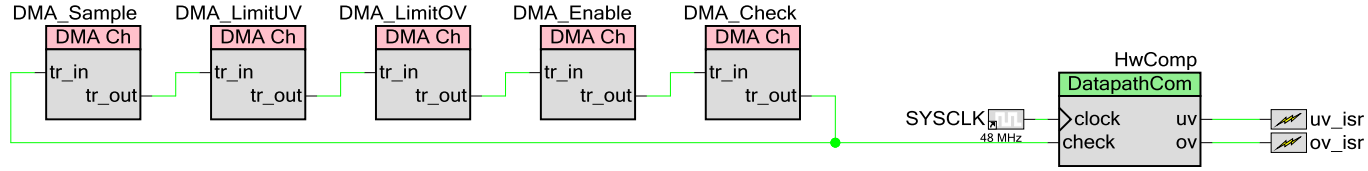
The Component can use either the internal precision voltage reference or an external reference to the SAR ADC's Vref. This gives the ADC a range of 0 to 2 x Vref Volts.

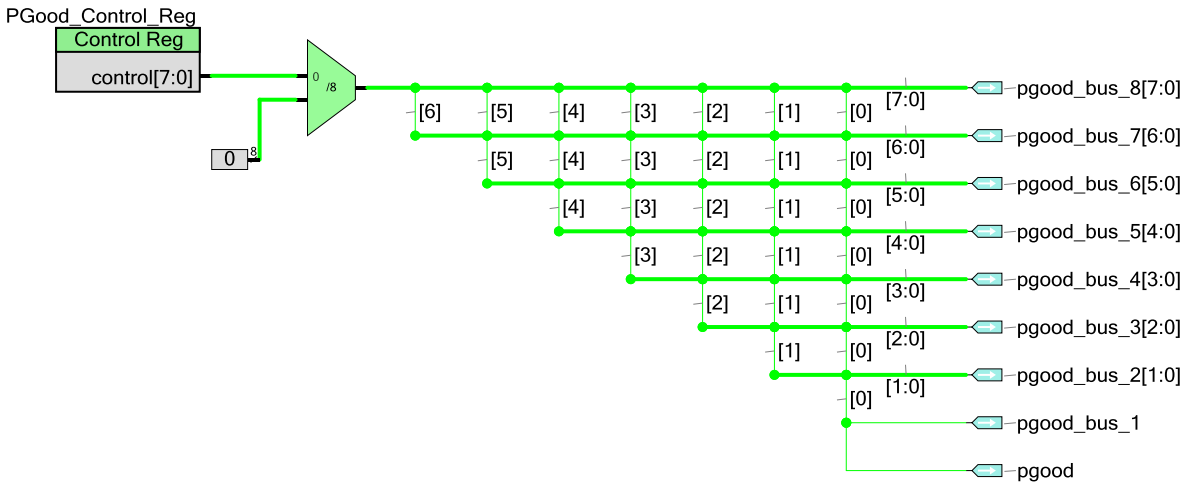
Block Diagram and Configuration

A simplified diagram of the Power Monitor hardware is shown below:



The SAR ADC Component is used in all forms of the Component. When not using DMA, the user must call the provided API to check for faults and warnings. When using DMA, the hardware controlled DMA chain, datapath comparator, and pgood control register below are used.





Clock Selection

Clocks are automatically selected by the Component, optimized for the fastest response time given the clocks defined in the project. The clocks are set upon opening the Configure dialog. If the clocks in the PSoC Creator Design-Wide Resources are changed after the Component is configured, the clock selection of the Component may be incorrect or sub-optimal. To fix this, re-open the Configure dialog and click the **OK** button.

DMA Support

The Power Monitor Component uses DMA when the Enable DMA box is checked. This box is only available on parts that have both DMAs and UDBs. The DMA chain transfers the ADC sample, fault limits, fault enable information and converter number to the UDB based Datapath Comparator for automatic fault detection.

Users can also use DMA to directly access the ADC conversion results as described in the datasheet of the Sequencing SAR ADC Component.

Industry Standards

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviationsfor the Component. There are two types of deviations defined:

- project deviations – deviations that are applicable for all PSoC Creator Components
- specific deviations – deviations that are applicable only for this Component

This section provides information on Component-specific deviations. Project deviations are described in the MISRA Compliance section of the *System Reference Guide* along with information on the MISRA compliance verification environment.

The Power Monitor Component does not have any specific deviations.

This Component has the following embedded Components: Sequencing SAR ADC, DMA Channel, Interrupt, and Clock. Refer to the corresponding Component datasheets for information on their MISRA compliance and specific deviations.

Registers

For fixed-function blocks, refer to the chip *Technical Reference Manual (TRM)* for more information about the registers.

Bits	7	6	5	4	3	2	1	0
SW Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Retention	Nonretention							
Name	Pgood[8:1]							

Name	Description
Pgood[8:1]	Pgood hardware status bits.

Resources

The Power Monitor uses the following device resources:

- SAR ADC
- With DMA Enabled:

Configuration	Resource Type					
	Datapath Cells	Macrocells	Status Cells	Control Cells	DMA Channels	Interrupts
Datapath Comparator	2	5	1	1	5	3

DC and AC Electrical Characteristics

Specifications are valid for -40° C = T_A= 85° C and T_J= 100° C, except where noted.

Specifications are valid for 1.71 V to 5.5 V, except where noted.

DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
A_RES	Resolution		–	–	12	bits
A_CHNIS_S	Number of channels - single ended		–	–	8	
A_GAINERR	Gain error	With external reference	–	–	±0.1	%
A_OFFSET	Input offset voltage	Measured with 1-V V_{REF}	–	–	2	mV
A_ISAR	Current consumption		–	–	1	mA
A_INRES	Input resistance	Based on characterization	–	–	2.2	KΩ
A_INCAP	Input capacitance	Based on characterization	–	–	10	pF

AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
A_PSRR	Power supply rejection ratio		70	–	–	dB
A_CMRR	Common mode rejection ratio	Measured at 1 V	66	–	–	dB
A_SAMP_1	Sample rate with external reference bypass cap	806 Ksps for PSoC 4100 BLE, PSoC 4100M	–	–	1	Msps
A_SAMP_3	Sample rate with no bypass cap. Internal reference		–	–	100	Ksps
A_SNDR	Signal-to-noise and distortion ratio (SINAD)	F_{IN} = 10 kHz	65	–	–	dB
A_INL	Integral non linearity	V_{DD} = 1.71 to 5.5, 1 Msps, Vref = 1.024	–1.7	–	+2	LSB
A_THD	Total harmonic distortion	F_{IN} = 10 kHz.	–	–	–65	dB

Block Specs

Parameter	Description	Min	Typ	Max	Units	Conditions
VREFSAR	Trimmed internal reference to SAR	-1	–	+1	%	Percentage of Vbg (1.024 V). Guaranteed by characterization. Note that Vbg is 1.2 V for PSoC 4100S/ PSoC 4100S Plus with the same accuracy.
RESPTIME	Response Time to a fault	6	-	-	µs	When using DMA, no voltage filtering, with bypass capacitor and ADC clock input frequency = 16 MHz

Component Changes

This section lists the major changes in the Component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
2.20	Removed redundant synchronization flop from datapath comparator.	Extra synchronization resulted in hold time violation on PSoC 4200L devices.
2.10	New Component version. Added support for PSoC 4100S Plus parts. Updated underlying SAR Component to remove unnecessary clock restriction on PSoC 4100S parts.	Adding support for more devices and improving performance on existing devices.
2.0.a	Added link to Component web page from symbol.	
2.0	Added DMA support for PSoC 4200 BLE parts. Moved Component from “Prototype” to “Production”. Fixed clock selection process to support PSoC Creator 4.1.	The clocks are selected when the Configure dialog is opened. If clocks are changed after the Component is configured, open the Configure dialog and click the **OK** button to update the clock selection.
1.0	Initial Component release.	

© Cypress Semiconductor Corporation, 2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical Components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical Component is any Component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.