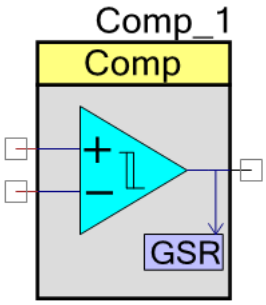
PSoC 4 Voltage Comparator (Comp)

1.20

Features



- Low input offset
- User controlled offset calibration
- Multiple speed modes
- Operates in Deep Sleep power mode
- Output routable to digital logic blocks or pins
- Selectable output polarity
- Multiple interrupt modes

General Description

The PSoC 4 Voltage Comparator (aka Comparator) Component gives a hardware solution to compare two analog input voltages. The output is sampled in the software or routed to a digital component. There are three speed levels to allow optimizing for speed or power consumption. A reference or external voltage can also be connected to either input.

The input offset is designed to be less than 1 mV over the temperature and voltage. The hysteresis is selectable between 10 mV and no hysteresis.

For all devices, except PSoC 4100/PSoC 4200, the Comparator can operate in Deep Sleep power mode.

When to Use Comparator

The Comparator gives you a fast comparison between two voltages, compared to using an ADC. Although you can use an ADC with software to compare multiple voltage levels, applications requiring a fast response or little software intervention are good candidates for this comparator. Some example applications include CapSense®, power supplies or simple translation from an analog level to a digital signal.

A common configuration is to create an adjustable comparator by connecting a voltage DAC to the negative input terminal.

Input/Output Connections

This section describes various input and output connections for the Comparator.

Positive Input–Analog Input

This input is usually connected to the voltage that is being compared. This input can be routed from a GPIO or from an internal source.

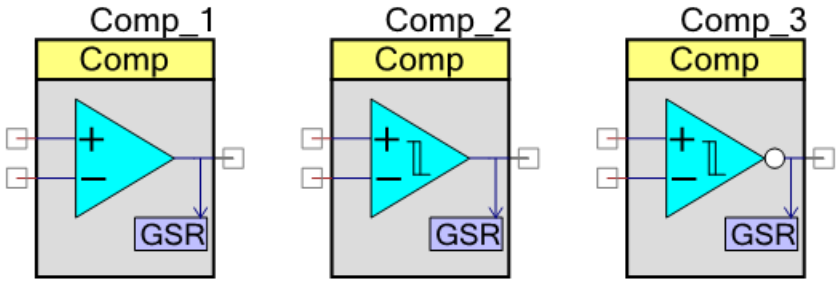
Negative Input– Analog Input

This input is usually connected to the reference voltage. This input can be routed from a GPIO or from an internal source.

Comparator Out–output

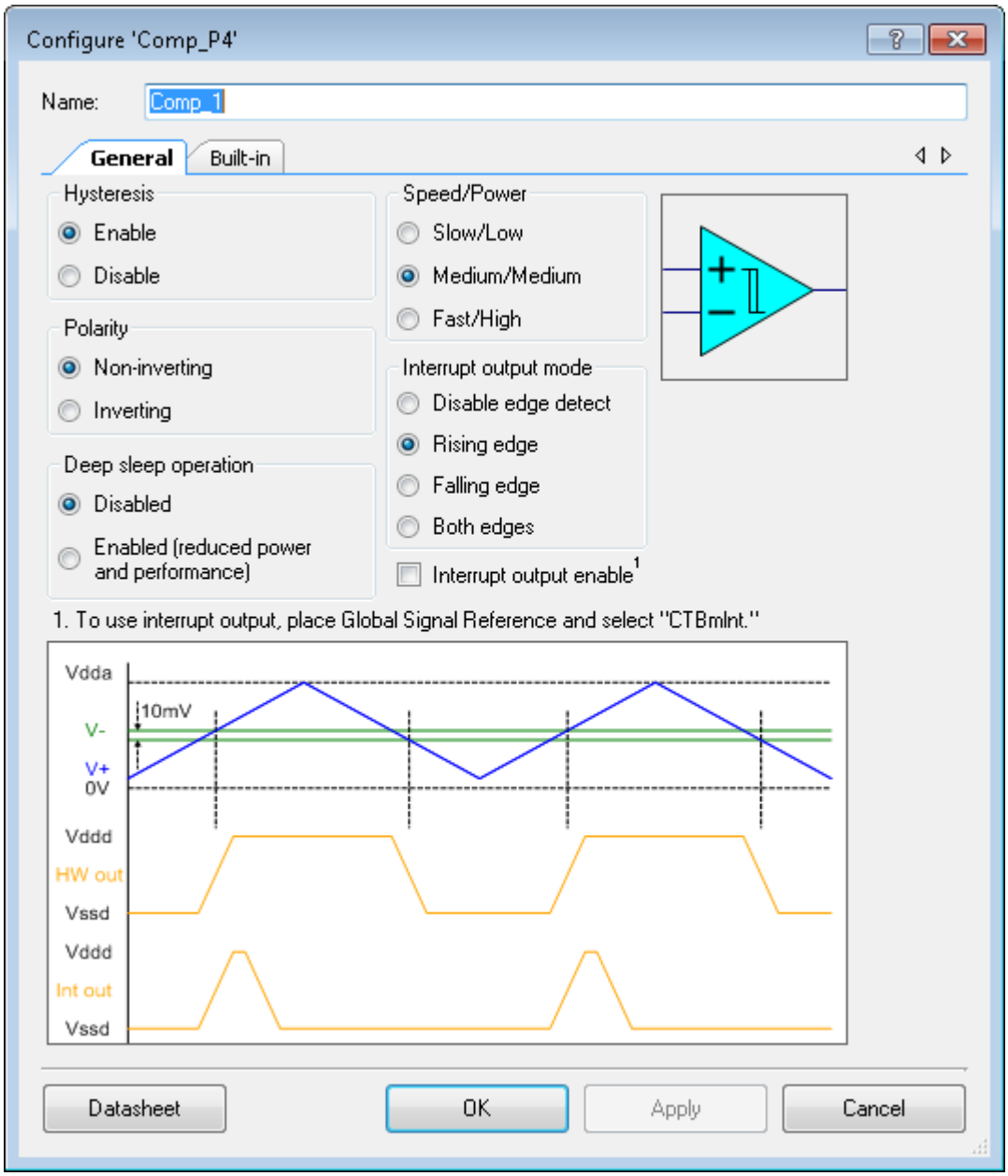
This is the digital comparison output. For a non-inverting configuration, this output goes high when the positive input voltage is greater than the negative input voltage. If the polarity is set to inverting, the output goes high when the negative input voltage is greater than the positive input voltage. The inverting configuration is implemented using an inverter in the UDB blocks and therefore requires a device that has UDB resources. The output can be routed to other component digital inputs such as interrupts, timers, etc.

The symbol for this component is annotated to denote the selection of hysteresis or inversion of the output.



Component Parameters

Drag the Comparator onto your design and double-click it to open the Configure dialog.

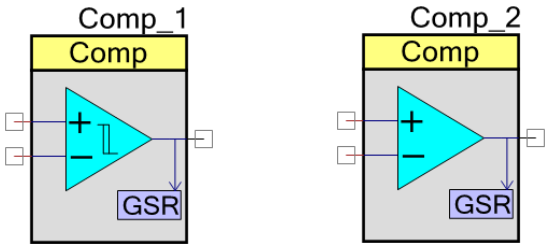


The component has the following parameters.

Hysteresis

This parameter gives you the ability to add approximately 10 mV of hysteresis to the Comparator. This helps to ensure that slowly moving voltages or slightly noisy voltages do not cause the output to oscillate when the two input voltages are nearly equal.

Hysteresis Enabled **Hysteresis Disabled**



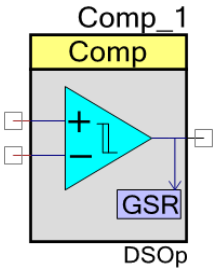
Polarity

This parameter allows inverting the output. This is useful for peripherals that require an inverted signal from the Comparator. The sampled signal state returned by the software API and the output seen by the power manager (see the *System Reference Guide* section on Alt Active and Sleep) is not affected by this parameter.

Note Inversion logic for the comparator is implemented using UDB resources. This option is not available when Deep Sleep mode is enabled.

Deep sleep operation

This parameter is not available for PSoC 4100/PSoC 4200 devices. It enables component operation in Deep Sleep mode. If this option is enabled, a "DSOp" label displays under the symbol. If the two Comparators of the same CTB/CTBm block are used in the project, both must have the same Deep Sleep settings.



Note When Deep Sleep mode is enabled and the component supply voltage is lower than 2.7 volts, only the "Slow/Low" **Speed/Power** setting is valid. The High and Medium settings may result in output oscillation.

Speed/Power

This parameter provides a way to optimize speed versus power consumption. The **Speed/Power** parameter allows selecting the next level: Fast Speed/High Power, Medium Speed/Medium Power, and Slow Speed/Low Power.

Interrupt output mode

This parameter defines the event that will cause a pulse to be generated on the comparator interrupt output. You can select the interrupt mode: Disable edge detect, Rising edge, Falling edge, or Both edges.

Interrupt output enable

This parameter allows you to configure the Interrupt Mask for PSoC 4 devices, except PSoC 4100/PSoC 4200 devices.

Note The pulse on the comparator interrupt output will be generated only if the **Interrupt output enable** option is enabled. Also the Comp_SetInterruptMask() API or Comp_EnableInterruptOutput() API can be used for this purpose.

The CTBmInt /CTBInt configuration of Global Signal Reference (GSR) component should be used to generate an interrupt from the comparator.

Application Programming Interface

The Application Programming Interface (API) routines allow configuring the component using software. This table lists and describes the interface to each function. The following sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "Comp_1" to the first instance of a component in a given design. It can be renamed 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 "Comp"

General Functions

Function	Description
Comp_Start()	Performs all of the required initialization for the component and enables power to the block.
Comp_Stop()	Turns off the Comparator block.
Comp_Init()	Initializes or restores the component according to the customizer Configure dialog settings.
Comp_Enable()	Activates the hardware and begins component operation.

Function	Description
Comp_GetCompare()	Returns compare result.
Comp_SetSpeed()	Sets the power and speed to one of three settings: SLOW_SPEED, MED_SPEED, HIGH_SPEED.
Comp_ZeroCal()	Performs custom calibration of the input offset to minimize error for a specific set of conditions.
Comp_LoadTrim()	This function writes a value into the comparator trim register.

void Comp_Start(void)

Description: Performs all of the required initialization for the component and enables power to the block. The first time the routine is executed, the speed/power level, and hysteresis are set. When called to restart the comparator following a Stop() call, the current component parameter settings are retained.

Parameters: None

Return Value: None

Side Effects: None

void Comp_Stop(void)

Description: Turn off the Comparator block.

Parameters: None

Return Value: None

Side Effects: Does not affect Comparator modes or power settings

void Comp_Init(void)

Description: Initializes or restores the component according to the customizer Configure dialog settings. It is not necessary to call Init() because the Start() API calls this function and is the preferred method to begin the component operation.

Parameters: None

Return Value: None

Side Effects: All registers will be set to values according to the customizer Configure dialog.

void Comp_Enable(void)

Description: Activates the hardware and begins the component operation. It is not necessary to call Enable() because the Start() API calls this function, which is the preferred method to begin the component operation.

Parameters: None

Return Value: None

Side Effects: None

uint32 Comp_GetCompare(void)

Description: This function returns a nonzero value when the voltage connected to the positive input is greater than the negative input voltage. This value is not affected by the Polarity parameter. This value always reflects non-inverted state configuration.

This function reads the direct (unflopped) comparator output which can also be metastable (since it may result in incorrect data).

Parameters: None

Return Value: The comparator output state. Nonzero value when the positive input voltage is greater than the negative input voltage; otherwise, the return value is zero.

Side Effects: None

void Comp_SetSpeed(uint32 speed)

Description: Sets the speed to one of three settings; slow, medium, or fast.

Parameters: (uint32) speed: Comp_SLOW_SPEED, Comp_MED_SPEED, Comp_HIGH_SPEED

Return Value: None

Side Effects: None

uint32 Comp_ZeroCal(void)

Description: Performs custom calibration of the input offset to minimize error for a specific set of conditions: the comparator reference voltage, supply voltage, and operating temperature. A reference voltage in the range at which the comparator will be used must be applied to the Negative and Positive inputs of the comparator while the offset calibration is performed. This can be done external to the device or by using an internal analog mux on the positive input that selects between the positive input signal for normal operation and the negative input signal for calibration.

Parameters: None

Return Value: (uint32) the value from the comparator trim register after the offset calibration is complete. This value has the same format as the input parameter for the LoadTrim() API routine.

Side Effects: During the calibration procedure, the comparator output may behave erratically. The comparator output should be ignored during calibration.

void Comp_LoadTrim(uint32 trimVal)

Description: This function writes a value into the comparator offset trim register.

Parameters: uint32 trimVal: the value to be stored in the comparator offset trim register. This value has the same format as the parameter returned by the ZeroCal() routine.

Return Value: None

Side Effects: None

Interrupt Output Functions

Function	Description
Comp_SetInterruptMode()	Sets the interrupt edge detect mode.
Comp_EnableInterruptOutput()	Enables the output of the current comparator instance.
Comp_DisableInterruptOutput()	Disables the interrupt output for current comparator instance.
Comp_GetInterruptOutputStatus()	Get Interrupt status of current instance
Comp_ClearInterruptOutput()	Clears interrupt pending
Comp_SetInterruptOutput()	Sets the interrupt output.
Comp_GetInterruptSource()	Returns the status of all pending comparator interrupts
Comp_ClearInterrupt()	Clears the interrupt of all comparator interrupts with a mask
Comp_SetInterrupt()	Sets a software interrupt request for all comparators with mask.
Comp_SetInterruptMask()	Set interrupt mask for all comparators
Comp_GetInterruptMask()	Returns interrupt mask.
Comp_GetInterruptSourceMasked()	Returns interrupt request register masked by interrupt mask.

void Comp_SetInterruptMode(uint32 mode)

Description: Sets the interrupt edge detect mode.

Parameters: mode: Enumerated edge detect mode value. See table below.

Name	Description
Comp_INTR_DISABLE	Disable edge detect
Comp_INTR_RISING	Rising edge detect
Comp_INTR_FALLING	Falling edge detect
Comp_INTR_BOTH	Detect both edges

Return Value: None

Side Effects: None

void Comp_EnableInterruptOutput (void)

Description: Enables the interrupt output so that it will be ORed to the global Signal CTBmInt with the other comparator interrupt outputs. CTBmInt is available from the Global Signal component.

Parameters: None

Return Value: None

Side Effects: None

void Comp_DisableInterruptOutput (void)

Description: Disables the interrupt output so that it will not be ORed to the global Signal CTBmInt with the other comparator interrupt outputs.

Parameters: None

Return Value: None

Side Effects: None

uint32 Comp_GetInterruptOutputStatus (void)

Description: Returns the status of the comparators interrupt output.

Parameters: None

Return Value: 0 = Interrupt not active, 1 = Interrupt active

Side Effects: None

void Comp_ClearInterruptOutput (void)

Description: Clears the interrupt request. This function is used with the combined interrupt signal from the global signal reference.

Parameters: None

Return Value: None

Side Effects: None

void Comp_SetInterruptOutput (void)

Description: Sets a software interrupt request. This function is used with the combined interrupt signal from the global signal reference.

Parameters: None

Return Value: None

Side Effects: None

uint32 Comp_GetInterruptSource(void)

Description: Gets the interrupt requests. This function is used with the combined interrupt signal from the global signal reference. This function from either component instance can be used to determine the interrupt source for all the comparator interrupts combined in CTBm block.

Parameters: None

Return Value: Interrupt source. Each component instance has a mask value: Comp_INTR.

Side Effects: None

void Comp_ClearInterrupt(uint32 interruptMask)

Description: Clears the interrupt request. This function is used with the combined interrupt signal from the global signal reference. This function can be used to clear either or both interrupts in CTBm block.

Parameters: interruptMask: Mask of interrupts to clear. Each component instance has a mask value: Comp_INTR.

Return Value: None

Side Effects: None

void Comp_SetInterrupt(uint32 interruptMask)

Description: Sets a software interrupt request. This function is used with the combined interrupt signal from the global signal reference. This function can be used to trigger either or both software interrupts in CTBm block.

Parameters: interruptMask: Mask of interrupts to set. Each component instance has a mask value: Comp_INTR.

Return Value: None

Side Effects: None

void Comp_SetInterruptMask (uint32 interruptMask)

Description: Configures the interrupt mask for all comparators in CTBm block.

Parameters: interruptMask: bit-mask of interrupt sources to be enabled. Each component instance has a mask value: Comp_INTR_MASK.

Return Value: None

Side Effects: None

uint32 Comp_GetInterruptMask (void)

Description: Returns interrupt mask for all comparators in CTBm block.

Parameters: None

Return Value: Mask of enabled interrupt source. Each component instance has a mask value: Comp_INTR_MASK.

Side Effects: None

uint32 Comp_GetInterruptSourceMasked (void)

Description: Returns interrupt request register masked by interrupt mask for all comparators in CTBm block.

Returns the result of bitwise AND operation between corresponding interrupt request and mask bits.

Parameters: None

Return Value: Mask of enabled interrupt source. Each component instance has a mask value: Comp_INTR_MASKED.

Side Effects: None

Low Power Functions

Function	Description
Comp_Sleep()	This is the preferred API to prepare the component for sleep.
Comp_Wakeup()	This is the preferred API to restore the component to the state when Comp_Sleep() was called.

void Comp_Sleep(void)

Description: This is the preferred API to prepare the component for sleep. The Sleep() API saves the current component state and stops the component. Call the Comp_Sleep() function before calling the CySysPmDeepSleep() or the CySysPmHibernate() functions. If the "Deep sleep operation" is enabled then this function does nothing and the component continues to operate during low power state.

Parameters: None

Return Value: None

Side Effects: None

void Comp_Wakeup(void)

Description: This is the preferred API to restore the component to the state when Sleep() was called. The Wakeup() function will also re-enable the component. If the "Deep sleep operation" is enabled then this function does nothing because the component operates during low power state as well as in active state.

Parameters: None

Return Value: None

Side Effects: None

Global Variables

Function	Description
Comp_initVar()	Indicates whether or not the Comparator has been initialized. The variable is initialized to 0 and set to 1 the first time Comp_Start() is called. This allows the component to restart without reinitialization after the first call to the Comp_Start() routine. If reinitialization of the component is required, call Comp_Init() before calling Comp_Start(). Alternatively, the Comparator can be reinitialized by calling the Comp_Init() and Comp_Enable() functions

Sample Firmware Source Code

PSoC Creator provides numerous example projects that include schematics and example code in the Find Example Project dialog. For component-specific examples, open the dialog from the Component Catalog or an instance of the component in a schematic. For general examples, open the dialog from the Start Page or **File** menu. As needed, use the **Filter Options** in the dialog to narrow the list of projects available to select.

Refer to the "Find Example Project" topic in the PSoC Creator Help for more information.

MISRA Compliance

This section describes the MISRA-C:2004 compliance and deviations for 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 Comparator component has the following specific deviations:

Rule	Rule Class	Rule Description	Description of Deviation(s)
19.7	Advisory	A function should be used in preference to a function-like macro.	Deviated since function-like macros are used to allow more efficient code.

API Memory Usage

The component memory usage varies significantly, depending on the compiler, device, number of APIs used and component configuration. This table shows the memory usage for all APIs available in a given component configuration.

The measurements were done with the 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)

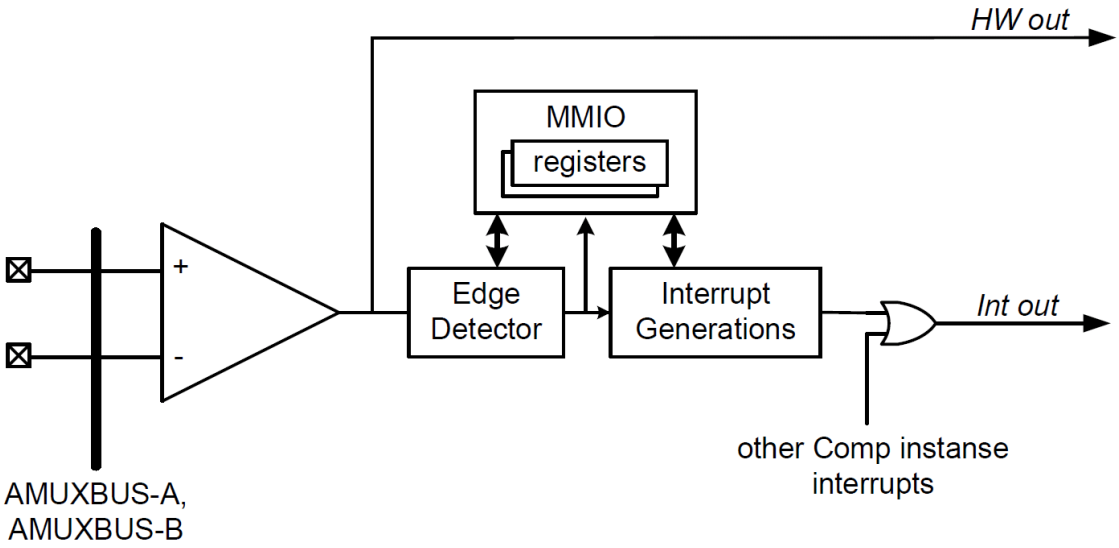
Configuration	PSoC 4100/PSoC 4200		Other PSoC 4 Devices (GCC)	
	Flash Bytes	SRAM Bytes	Flash Bytes	SRAM Bytes
Default (Deep sleep operation is disabled)	476	9	484	9
Deep sleep operation is enabled	N/A	N/A	432	8

Functional Description

The PSoC 4 Comparator component uses the opamp in the CTBm configured as a comparator. The CTBm supports 3 power choices that can be used to balance the response rate of the comparator with the power usage.

Block Diagram and Configuration

The following figure shows a high-level block diagram.



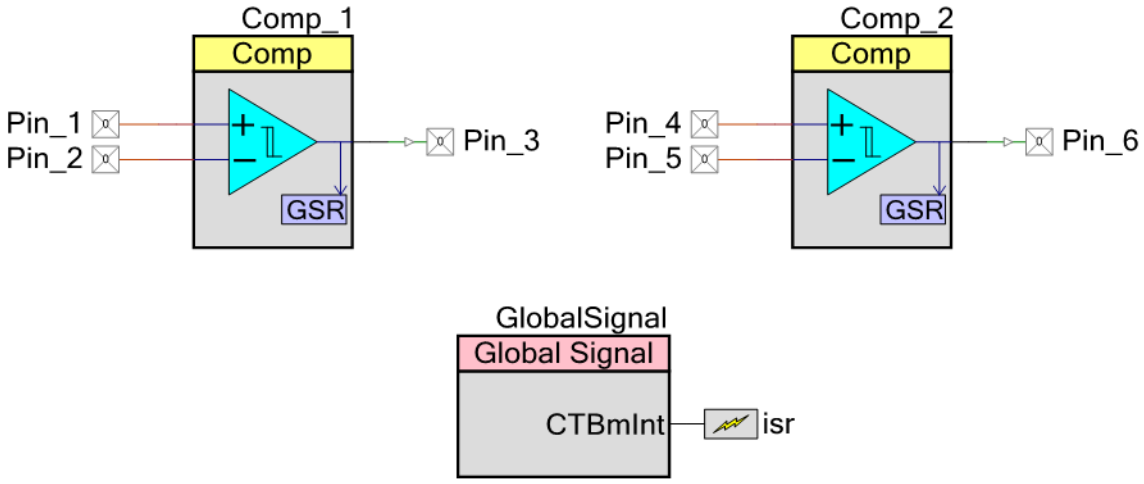
Individual comparator interrupt outputs are ORed together as a single asynchronous interrupt source before sent out and used to wake up the system in low power mode. The individual comparator interrupt is masked by INTR_MASK. The masked result is captured in INTR_MASKED register. Writing 1 to INTR register bit will clear the interrupt. Refer to the appropriate device *Technical Reference Manual (TRM)* for a detailed description of the registers.

Interrupt Service Routine

The Comparator component supports interrupts on the various events, depending on *Interrupt output mode* settings: Rising edge, Falling edge or Both edges. The interrupt signal goes high when any of the enabled interrupt configurations are true.

Note Interrupt is not cleared automatically. It is the user's responsibility to do that. The interrupt is cleared by writing a '1' in the corresponding interrupt register bit position. The preferred way to clear interrupt sources is using the *Comp_ClearInterrupt(Comp_INTR)* or *Comp_ClearInterruptOutput()* API.

Example Schematic



The following code is suggested:

```
CY_ISR(Comp_Interrupt)
{
if (Comp_1_GetInterruptSourceMasked() & Comp_1_INTR_MASKED)
    {
        /* Interrupt is not cleared automatically.
           It is user responsibility to do that.   */
        Comp_1_ClearInterrupt(Comp_1_INTR); /* or Comp_1_ClearInterruptOutput() */

        /* Add user interrupt code to manage interrupt. */
    }

/* This is alternative to (Comp_2_GetInterruptSourceMasked() &
   Comp_2_INTR_MASKED) */
if (Comp_2_GetInterruptOutputStatus())
    {
        /* Interrupt is not cleared automatically.
           It is user responsibility to do that.   */
        Comp_2_ClearInterrupt(Comp_2_INTR); /* or Comp_2_ClearInterruptOutput() */

        /* Add user interrupt code to manage interrupt.   */
    }
}
void main()
{
    Comp_1_Start();
    Comp_2_Start();
    isr_StartEx(Comp_Interrupt);
    CyGlobalIntEnable; /* Enable global interrupts. */

    for(;;)
    {
        /* Place your application code here. */
    }
}
```

Operation in Low Power Mode

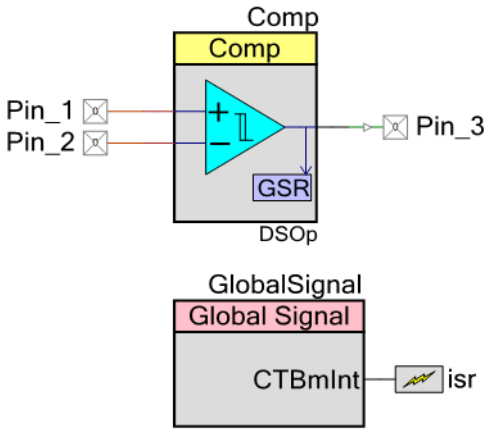
In PSoC 4 (except PSoC 4100/PSoC 4200) the component can be used along with the Global Signal component as a wakeup source from Deep Sleep mode as described below.

Create schematic as described in previous section. The option “Combined CTBm interrupt (CTBInt / CTBmInt)” should be selected as a source for an interrupt in the Global Signal Reference component. This component triggers the output each time any of the enabled comparators generates an interrupt.

For Deep sleep mode, the Comparator component input pins must be assigned to dedicated device pins only. Refer to the device datasheet for the part being used for the specific physical pin connections.

The wakeup event is when the voltage connected to the positive input (Pin_1) is greater than the negative input voltage (Pin_2).

Example Schematic



The following code is suggested:

```
CY_ISR(Comp_Interrupt)
{
    /* Interrupt is not cleared automatically.
       It is user responsibility to do that.   */
    Comp_ClearInterrupt(Comp_INTR); /* or Comp_ClearInterruptOutput() */

    /* Add user interrupt code to manage interrupt.  */
}

void main()
{
    Comp_Start();
    isr_StartEx(Comp_Interrupt);
    CyGlobalIntEnable;     /* Enable global interrupts. */
    Comp_SetInterruptMode(Comp_INTR_RISING);
    Comp_SetInterruptMask(Comp_INTR_MASK); /* or Comp_EnableInterruptOutput() */
```

```
        for(;;)
        {
            /* Place your application code here. */

            CySysPmDeepSleep(); /* Enter Deep Sleep mode*/

            /* Place your application code here. */
        }
    }
```

Placement

Each Comparator is directly connected to specific GPIOs for its inputs along with being connected to the internal fabric. The output connection is routed to the digital fabric. Refer to the device datasheet for the part being used for specific physical pin connections.

Registers

See the chip Technical Reference Manual (TRM) for more information about registers.

Component Debug Window

PSoC Creator allows viewing debug information about the components in the design. Each component window lists the memory and registers for the instance. For detailed hardware registers descriptions, refer to the appropriate device technical reference manual. For detailed UDB registers descriptions used in the component, refer to the Registers section of this datasheet.

To open the Component Debug window:

1. Make sure the debugger is running or in break mode.
2. Choose **Windows > Components…** from the **Debug** menu.
3. In the Component Window Selector dialog, select the component instances to view and click **OK**.

The selected Component Debug window(s) will open within the debugger framework. Refer to the "Component Debug Window" topic in the PSoC Creator Help for more information.

Resources

The Comparator uses one of the two opamps in PSoC 4 CTBm block (Continuous Time Block mini). The number of available CTBm blocks depends on used device. Refer to the device datasheet for details. If the inverting output option is chosen a single macrocell in the UDB array is also used.

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.

Note Final characterization data for the PSoC Analog Coprocessor device is not available at this time. Once the data is available, the component datasheet will be updated on the Cypress web site.

DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
I_{DD}	Opamp Block current. No load.		–	–	–	–
I_{DD_HI}	Power = high		–	1000	1300	µA
I_{DD_HI}	Power = high	PSoC 4100M / PSoC 4200M / PSoC 4200L/PSoC 4100S	–	1100	1850	µA
I_{DD_MED}	Power = medium		–	320	500	µA
I_{DD_MED}	Power = medium	PSoC 4100M / PSoC 4200M / PSoC 4200L/PSoC 4100S	–	550	950	µA
I_{DD_LOW}	Power = low		–	250	350	µA
I_{DD_LOW}	Power = low	PSoC 4100M / PSoC 4200M / PSoC 4200L/PSoC 4100S	–	150	350	µA
V_{IN}	Charge pump on, $V_{DDA} \geq 2.7$ V		–0.05	–	VDDA – 0.2	V
V_{CM}	Charge pump on, $V_{DDA} \geq 2.7$ V		–0.05	–	VDDA – 0.2	V
V_{OS_TR}	Offset voltage, trimmed	High mode	1	±0.5	1	mV
V_{OS_TR}	Offset voltage, trimmed	Medium mode	–	±1	–	mV
V_{OS_TR}	Offset voltage, trimmed	Low mode	–	±2	–	mV
$V_{OS_DR_TR}$	Offset voltage drift, trimmed	High mode	–10	±3	10	µV/C
$V_{OS_DR_TR}$	Offset voltage drift, trimmed	Medium mode	–	±10	–	µV/C
$V_{OS_DR_TR}$	Offset voltage drift, trimmed	Low mode	–	±10	–	µV/C
CMRR	DC	VDDD = 3.6 V	70	80	–	dB
CMRR	DC, VDDD = 3.6 V, High-Power Mode	PSoC 4200 BLE	65	70	–	dB

Parameter	Description	Conditions	Min	Typ	Max	Units
CMRR	DC Common mode rejection ratio. High Power mode. Common Model Voltage Range from 0.5V to VDDA - 0.5V. VDDD = 3.6 V	PSoC 4100M / PSoC 4200M / PSoC 4200L/PSoC4100S	60	70	–	dB
Vhyst_op	Hysteresis		–	10	–	mV
Z_{CMP}	DC Input impedance of comparator		35	–	–	MΩ

AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
Comp_mode	Comparator mode; 50 mV drive, Trise = Tfall (approx.)		–	–	–	–
T_{PD1}	Response time; power = high		–	150	–	nsec
T_{PD2}	Response time; power = medium		–	400	–	nsec
T_{PD2}	Response time; power = medium, Input is 0.2 V to V_{DDA}-0.2 V	PSoC 4100S	–	500	–	nsec
T_{PD3}	Response time; power = low		–	1000	–	nsec
T_{PD3}	Response time; power = low	PSoC 4100M / PSoC 4200M / PSoC 4200L	–	2000	–	nsec
T_{PD3}	Response time; power = lo, Input is 0.2 V to V_{DDA}-0.2 V	PSoC 4100S	–	2500	–	nsec
T_op_wake	From disable to enable, no external RC dominating		–	300	–	µSec
T_op_wake	From disable to enable, no external RC dominating	PSoC 4100M / PSoC 4200M / PSoC 4200L/PSoC 4100S	–	25	–	µSec

Component Errata

This section lists known problems with the component.

Cypress ID	Component Version	Problem	Workaround
232797	1.20	Whenever a hardware interrupt is generated by Comp_P4 component or _Start API is called on the PSoC Analog Coprocessor, the associated logic causes the interrupt to set again.	If you are facing this problem (when try to clear it through SW), the two clear interrupt operations should be provided to clear it.

Component Changes

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

Version	Description of Changes	Reason for Changes / Impact
1.20.b	Edited datasheet.	Added final characterization data for PSoC 4100S device.
1.20.a	Edited datasheet.	Final characterization data for PSoC 4000S, PSoC 4100S and PSoC Analog Coprocessor devices is not available at this time. Once the data is available, the component datasheet will be updated on the Cypress web site.
	Edited datasheet to add Component Errata section.	Document an issue with interrupts on PSoC Analog Coprocessor devices.
1.20	Added support for PSoC Analog Coprocessor devices.	New device support.
	Edit to datasheet.	Updated Component Parameters section with clarification of the deep sleep operation and the interrupt generation.
1.10.d	Edit to datasheet.	Updated information to include PSoC 4200L devices.
1.10.c	Edit to datasheet.	Updated information to include PSoC 4100M/ PSoC 4200M devices. Updated DC and AC Electrical Characteristics section with PSoC 4100M/PSoC 4200M data.
1.10.b	Edit to datasheet.	Added CMRR char data for PSoC 4200 BLE devices.
1.10.a	Edits to datasheet.	Updates to clarify text and conform to template.
1.10	Added functions to enable and control interrupts.	Provide similar functionality as PSoC 4 Low Power Comparator.

Version	Description of Changes	Reason for Changes / Impact
	Added “Deep sleep operation” parameter on General Tab for control component availability in Deep Sleep mode.	Updates to support PSoC 4200 BLE devices.
	Updated API Memory usage and MISRA compliance sections.	
	Removed mention about SaveConfig() and RestoreConfig() APIs because they are empty.	
1.0	New Component	

© Cypress Semiconductor Corporation, 2016-2017. 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.