Comparator Datasheet CMP V 1.1

Copyright © 2012 Cypress Semiconductor Corporation. All Rights Reserved.

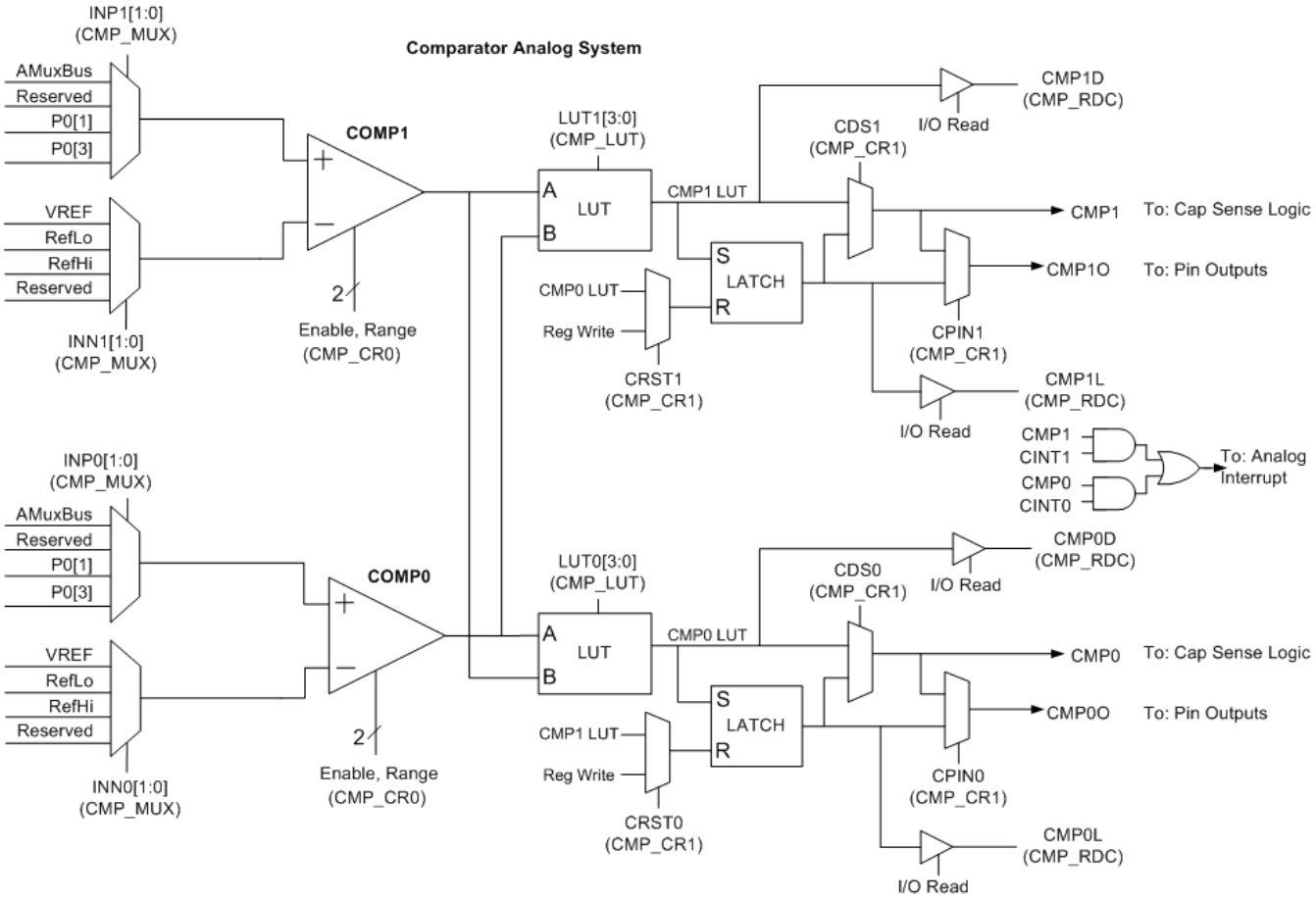
Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	CapSense®	I2C/SPI	Timer	Flash ±3%	RAM	
CY8C20xx7, CYRF89435, CYRF89535						
	-	-	-	141	0	0–2

Features and Overview

- Selectable positive and negative inputs
- Direct connection to interrupt controller
- Programmable power consumption

The CMP User Module has two comparators, each of which has an analog comparison of two selectable inputs. One of the two inputs is always a voltage reference. The inputs and the function and routing of the outputs are flexible. The output of the comparator and of the neighboring comparator can be combined with any two-input logic function. In addition, you have the option of combining this combinatorial output with a latched value and routing it to a pin output or to the interrupt controller. The CMP User Module controls the input multiplexers and the comparator. After the user module is placed, the combinatorial logic is automatically configured for typical operation and can be changed through the Device Editor.

Figure 1. CMP Block Diagram



Functional Description

The CMP User Module contains:

- Two comparators with two port inputs
- A global analog bus input
- An optional single low pass filter on the positive input (PMux) of either CMP (or both)
- Three selectable voltage references on the negative input (NMux)
- Configurable combinatorial logic on the output

When they are disabled, the comparators consume no power. Two active modes provide a full rail-to-rail input range, and a somewhat lower power option with limited input range. The output of the comparator is high when the positive input is higher than the negative input.

The configurable combinatorial logic automatically sets a default configuration when the user module is placed. The latch is not enabled by default. This configures the CMP User Module to pass through the comparator results to output. If other functions are needed, the combinatorial logic can be configured through the GUI interface.

The output combinatorial logic is made of the CMP LUT and the Latch Logic. Each LUT has two inputs, the output of its comparator (CMPx) and the output of the neighboring comparator (CMPy). Each CMP LUT can be configured to implement one of sixteen two-input logic functions. See the following table for possible configurations. The Latch Logic latches the output of the CMPx on a rising clock edge of SysClk.

To reset the latch, use either a rising edge of the output of the other CMP LUT or write a zero to the CMP_RDC register's CMP0L bit. You can use the CMP_ClearLatchx() API function. Refer to the PSOC Designer

Either of the comparators can be used to generate a single interrupt. The register bits CINT0 and CINT1 are inputs to AND gates that combine into a single OR gate that outputs to the analog interrupt. The output interrupt logic is set graphically in the Device Editor. Refer to the *PSoC Designer Integrated Development Environment User Guide* for details on working with the Digital Interconnect view in the Device Editor.

Table 1. 16 Two-Input Boolean Logic Functions

A	B	~A	~B
A AND B	A NAND B	A AND ~B	~A AND B
A OR B	A NOR B	A OR ~B	~A OR B
A XOR B	A XNOR B	TRUE	FALSE

DC and AC Electrical Characteristics

Table 2. Comparator Electrical Specifications for CY8C20xx7

Parameter	Description	Min	Typ	Max	Units	Notes
T_{COMP}	Comparator Response Time		70	100	ns	50 mV overdrive
V_{OS}	Input Offset Voltage		2.5	30	mV	
I_{DD}	Supply Current		20	80	µA	Average DC current, 50 mV overdrive
PSRR	Supply voltage >2V		80		dB	Power Supply Rejection Ratio
	Supply voltage <2V		40		dB	
V_{CM}	Common Mode Input Voltage Range	0		1.5	V	

Placement

The blocks for the user module are automatically placed when the UM is instantiated; alternate placements are not available.

Parameters and Resources

PMux0 and PMux1

This parameter controls the positive input selection for comparators 0 (PMux0) and 1 (PMux1). Either the Analog Mux Bus (AnalogMuxBus), or one of two pins (Port_0_1 or Port_0_3) can be selected as the input. The connection to each multiplexer can be selected independent from the other, so the PMux0 and PMux1 inputs can be connected together if required.

PMux0 and PMux1 possible positive input values:

- AnalogMuxBus (default)

- Port_0_1
- Port_0_3

NMux0 and NMux1

This parameter controls the negative input selection for comparators 0 (NMux0) and 1 (NMux1). The negative input can come from three different voltage references; Voltage Reference (VREF, 1.3 V), Voltage Reverence Lo (Ref Lo, approximately 0.9V), or Voltage Reference Hi (Ref Hi, approximately 1.8V). You can connect each multiplexer independent of the other, so NMux0 and NMux1 inputs can be connected if necessary.

NMux0 and NMux1 possible negative input values:

- VREF (default)
- RefLo
- RefHi

Interrupt Generation Control

The following two parameters InterruptAPI and IntDispatchMode are only accessible when the **Enable interrupt generation control** check box in PSoC Designer is checked. This is available under **Project > Settings > Chip Editor**.

InterruptAPI (Advanced, default = Enable)

The InterruptAPI parameter allows conditional generation of a user module's interrupt handler and interrupt vector table entry. Select **Enable** to generate the interrupt handler and interrupt vector table entry. Select **Disable** to bypass the generation of the interrupt handler and interrupt vector table entry. Select **Enable** only when it is necessary to generate an interrupt dispatch code. Disabling it when it is unneeded reduces overhead. This is particularly important with projects that have multiple overlays where a single block resource is used by the different overlays.

IntDispatchMode (Advanced, default = ActiveStatus)

The IntDispatchMode parameter is used to specify how an interrupt request is handled for interrupts shared by multiple user modules existing in the same block but in different overlays. Selecting **ActiveStatus** causes firmware to test which overlay is active before servicing the shared interrupt request. This test occurs every time the shared interrupt is requested. This adds latency and also produces a nondeterministic procedure of servicing shared interrupt requests, but does not require any RAM. Selecting **OffsetPreCalc** causes firmware to calculate the source of a shared interrupt request only when an overlay is initially loaded. This calculation decreases interrupt latency and produces a deterministic procedure for servicing shared interrupt requests, but at the expense of a byte of RAM.

Application Programming Interface

API functions allow designers to interact with the user module at a higher level. This section specifies the interface to each function together with related constants provided by the include files.

Note

In all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X prior to the call if those values are required after the call. This "registers are volatile" policy was selected for efficiency reasons. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API functions may leave A and X unchanged, there is no guarantee they will do so in the future.

For Large Memory Model devices, the caller must preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, that condition might change in future releases.

CMP_Start0

Description:

Sets the mode (off, normal power, or low power) of Comparator 0.

C Prototype:

```
void CMP_Start0(BYTE bMode);
```

Assembler:

```
mov  A, CMP_CMP0MODE_NORMALPOWER
lcall CMP_Start0
```

Parameters:

bMode – Mode value to be set.

Symbolic Name	Value
CMP_CMP0MODE_OFF	0x00
CMP_CMP0MODE_NORMALPOWER	0x01
CMP_CMP0MODE_LOWPOWER	0x03

Return Value:

None

Side Effects:

The A and X registers may be altered by this function.

CMP_Start1

Description:

Sets the mode (off, normal power, or low power) of Comparator 1.

C Prototype:

```
void CMP_Start1(BYTE bMode);
```

Assembler:

```
mov  A, CMP_CMP1MODE_NORMALPOWER
lcall CMP_Start1
```

Parameters:

bMode – Mode value to be set.

Symbolic Name	Value
CMP_CMP1MODE_OFF	0x00
CMP_CMP1MODE_NORMALPOWER	0x10
CMP_CMP1MODE_LOWPOWER	0x30

Return Value:

None

Side Effects:

This function might alter the A and X registers.

CMP_Stop0

Description:

Removes power from Comparator 0. The outputs will not be driven.

C Prototype:

```
void CMP_Stop0(void);
```

Assembler:

```
lcall CMP_Stop0
```

Parameters:

None

Return Value:

None

Side Effects:

This function might alter the A and X registers.

CMP_Stop1

Description:

Removes power from Comparator 1. The outputs will not be driven.

C Prototype:

```
void CMP_Stop1(void);
```

Assembler:

```
lcall CMP_Stop1
```

Parameters:

None

Return Value:

None

Side Effects:

This function might alter the A and X registers.

CMP_EnableInt

Description:

Enables the analog interrupt and allows either comparator 0 or comparator 1 to drive the interrupt but not both.

C Prototype:

```
void CMP_EnableInt(BYTE bComparator);
```

Assembler:

```
mov  A, CMP_CINT0_MASK
lcall CMP_EnableInt
```

Parameters:

bComparator – The mask of either comparator 0 or comparator 1 (CINT0 or CINT1 of the CMP_CR1 register) to set which comparator drives the analog interrupt. If any other value other than the ones below (for example, 0x88 (CMP_CINT0_MASK | CMP_CINT1_MASK) or 0x00) are passed to this function then the analog interrupt will be enabled, but neither comparator will drive the interrupt.

Symbolic Name	Value
CMP_CINT0_MASK	0x08
CMP_CINT1_MASK	0x80

Return Value:

None

Side Effects:

This function might alter the A and X registers.

CMP_DisableInt

Description:

Disables the analog interrupt and the comparator interrupt select so that neither CINT0 or CINT1 can drive the interrupt.

C Prototype:

```
void CMP_DisableInt(void);
```

Assembler:

```
lcall CMP_DisableInt
```

Parameters:

None

Return Value:

None

Side Effects:

This function might alter the A and X registers.

CMP_ClearInt

Description:

Clears the posted analog interrupt.

C Prototype:

```
void CMP_ClearInt(void);
```

Assembler:

```
lcall CMP_ClearInt
```

Parameters:

None

Return Value:

None

Side Effects:

This function might alter the A and X registers.

CMP_ClearLatch0

Description:

Clears the latched output of comparator 0 by clearing the CRST0 bit of the CMP_CR1 register and then clearing the CMP0L bit of the CMP_RDC register. See the Technical Reference Manual (TRM) for a more detailed description of the comparator latch output.

C Prototype:

```
void CMP_ClearLatch0(void);
```

Assembler:

```
lcall CMP_ ClearLatch0
```

Parameters:

None

Return Value:

None

Side Effects:

This function might alter the A and X registers.

CMP_ClearLatch1

Description:

Clears the Latched output of comparator 1 by clearing the CRST1 bit of the CMP_CR1 register and then clearing the CMP1L bit of the CMP_RDC register. See the TRM for a more detailed description of the comparator latch output.

C Prototype:

```
void CMP_ClearLatch1(void);
```

Assembler:

```
lcall CMP_ ClearLatch1
```

Parameters:

None

Return Value:

None

Side Effects:

This function might alter the A and X registers.

CMP_ChangeMuxes0

Description:

Changes the negative (NMux0) and positive (PMux0) input selection for comparator 0 only.

C Prototype:

```
void CMP_ChangeMuxes0(BYTE bMuxSetting);
```

Assembler:

```
mov  A, (CMP_PMUX0_AMUXBUS | CMP_NMUX0_VREF_1P3V)
lcall CMP_ChangeMuxes0
```

Parameters:

bMuxSetting: OR of the PMux0 bits and NMux0 bits to be set. If values for PMux0 and NMux0 are not ORed, then the one that is not passed in gets set to 0x00. For example, if only CMP_PMUX0_PIN_0_1 is passed in, the comparator input from NMux0 is set to 0x00, which is CMP_NMUX0_VREF_1P3V.

PMux0 Symbolic Name	Value	NMux0 Symbolic Name	Value
CMP_PMUX0_AMUXBUS	0x00	CMP_NMUX0_VREF_1P3V	0x00
CMP_PMUX0_PIN_0_1	0x08	CMP_NMUX0_REF_LO_0P9V	0x01
CMP_PMUX0_PIN_0_3	0x0C	CMP_NMUX0_REF_HI_1P8V	0x02

Return Value:

None

Side Effects:

This function might alter the A and X registers.

CMP_ChangeMuxes1

Description:

Changes the negative (NMux1) and positive (PMux1) input selection for comparator 1 only.

C Prototype:

```
void CMP_ChangeMuxes1(BYTE bMuxSetting);
```

Assembler:

```
mov  A, (CMP_PMUX1_AMUXBUS | CMP_NMUX1_VREF_1P3V)
lcall CMP_ChangeMode
```

Parameters:

bMuxSetting: OR of the PMux1 bits and NMux1 bits to be set. If values for PMux1 and NMux1 are not ORed, then the one that is not passed in gets set to 0x00. For example, if only CMP_PMUX1_PIN_0_1 is passed in, the comparator input from NMux1 is set to 0x00, which is CMP_NMUX1_VREF_1P3V.

PMux1 Symbolic Name	Value	NMux1 Symbolic Name	Value
CMP_PMUX1_AMUXBUS	0x00	CMP_NMUX1_VREF_1P3V	0x00
CMP_PMUX1_PIN_0_1	0x80	CMP_NMUX1_REF_LO_0P9V	0x10
CMP_PMUX1_PIN_0_3	0xC0	CMP_NMUX1_REF_HI_1P8V	0x20

Return Value:

None

Side Effects:

This function might alter the A and X registers.

CMP_bStatus

Description:

Returns the status of the comparator data signal and latch state by reading the CMP_READ_CLEAR_REG (CMP_RDC) register. The data signal is the CMPxD signal from the block diagram. The latch state is the CMPxL signal from the same diagram. This can be different from the interrupt status, depending on the comparator output logic settings.

C Prototype:

```
BYTE CMP_bStatus(void);
```

Assembler:

```
lcall CMP_bStatus  ;Call function to read status of comparator
                   ;Reg A contains return value
```

Parameters:

None

Return Value:

Contains bits indicating output of CMPxD and CMPxL signals from the block diagram. The bits are left in their native positions. The return value is returned in the accumulator.

Mask Name	Value	Description
CMP_CMP0D_MASK	0x10	Comparator 0 data signal
CMP_CMP0L_MASK	0x01	Latched state for Comparator 0
CMP_CMP1D_MASK	0x20	Comparator 1 data signal
CMP_CMP1L_MASK	0x02	Latched state for Comparator 1
CMP_STATUS_MASK	0x33	Mask for data signal and latch state for both comparators

Side Effects:

This function might alter the A and X registers.

Sample Firmware Source Code

This sample code first clears the posted analog interrupt then enables analog and global interrupts and sets up Comparator 0 to be the input to the analog interrupt. Then it starts both comparators by setting them to normal power mode. In the loop, it checks the status of the comparators and checks the interrupt counter to see if it has reached the limit and raises and lowers a pin if the limit has been reached.

Note Please rename the required output pin in the Chip Editor to "Output".

```
//-------------------------------------------------------------------
// Sample Code for the CMP User Module.
//
// See below for parameter configurations:
//
// PMux0 and PMux1: AnalogMuxBus
// NMux0 and NMux1: VREF
// (Interrupt should happen when voltage reaches approximately 1.3V)
// The above parameters can change with the use of
// CMP_ChangeMuxes0(BYTE bMuxSetting) and CMP_ChangeMuxes1(BYTE
// bMuxSetting)
//
// LPFTimeConstant: 1us
// LPFSource:       NONE
//
// NOTE: The CMP_ISR routine in CMPint.asm must be changed to call
//       myCMP_ISR_Handler inside the custom code banners.
//-------------------------------------------------------------------
#include <m8c.h>         // part specific constants and macros
#include "PSoCAPI.h"     // PSoC API definitions for all user modules
#include "PSoCGPIOInt.h" // Definitions for all GPIO Ports

#define ISR_CNT_LIMIT 3 //interrupt count limit

BYTE  isrCounter; //interrupt counter

#pragma interrupt_handler myCMP_ISR_Handler;
```

```
void    myCMP_ISR_Handler(void);

void main(void)
{
    BYTE status;
    isrCounter = 0;

    //Clear posted analog interrupt
    CMP_ClearInt();

    //Enable Global Interrupts
    M8C_EnableGInt;

   //Enable analog interrupts with Comparator 0 being the input to
// the analog interrupt
    CMP_EnableInt(CMP_CINT0_MASK);

//Start both comparators and set to normal power mode
    CMP_Start0(CMP_CMP0MODE_NORMALPOWER);
    CMP_Start1(CMP_CMP1MODE_NORMALPOWER);

while(1)
    {
status = CMP_bStatus(); //check status of both comparators
        if (isrCounter > ISR_CNT_LIMIT)
{
   isrCounter = 0;
   //Output is set to a Strong Drive
   Output_Data_ADDR |= Output_MASK;  //Raise Output
   Output_Data_ADDR &= ~Output_MASK; //Lower Output
}
}
}

//-------------------------------------------------------------------------
// myCMP_ISR_Handler    // The handler for the CMP ISR
//-------------------------------------------------------------------------
void myCMP_ISR_Handler(void)
{
isrCounter++;
//Clear posted analog interrupt
    CMP_ClearInt();
}
```

The following code is the same sample in assembly language:

```
include "m8c.inc"
include "PSoCAPI.inc"
include "PSoCGPIOInt.inc"

ISR_CNT_LIMIT: equ 3

area    bss    (RAM,REL)
isrCounter:                blk     1
```

```
area text(ROM,REL)
export _main
export _myCMP_ISR_Handler

_main:
    mov  [isrCounter], 0
call CMP_ClearInt                ; clear posted analog interrupt
    M8C_EnableGInt                   ; enable global interrupts
   mov  A, CMP_CINT0_MASK         ; specify comparator 0 to be input to analog interrupt
call CMP_EnableInt             ; enable analog interrupts with input from comparator 0
mov  A, CMP_CMP0MODE_NORMALPOWER ; specify Comparator 0 power level
call CMP_Start0                  ; and turn comparator 0 on
mov  A, CMP_CMP1MODE_NORMALPOWER ; do the same for comparator 1
call CMP_Start1

loop:
    call CMP_bStatus      ; check status of both comparators.  Return value is in A
mov  A,ISR_CNT_LIMIT  ; check if CMP ISR limit reached
    cmp  A,[isrCounter]
jnc  loop
mov  [isrCounter], 0  ; reset counter
    ;Output is set to a Strong Drive
or   reg[Output_Data_ADDR],Output_MASK   ;Raise Output
and  reg[Output_Data_ADDR],~Output_MASK  ;Lower Output
jmp  loop


;-----------------------------------------------------------------------
; myTimer_ISR_Handler    // The handler for the Timer ISR
;-----------------------------------------------------------------------
_myCMP_ISR_Handler:
inc   [isrCounter]
    call CMP_ClearInt  ;Clear posted analog interrupt
reti
```

Configuration Registers

The basic topology of the comparator sets most of the bits in the register configuration for the analog CT block that is used.

Table 3. Resource CMP_RDC: Bank 0, reg[78h] Comparator Read/Clear Register

Bit	7	6	5	4	3	2	1	0
Value	Reserved		CMP1D	CMP0D	Reserved		CMP1L	CMP0L

This register is used to read the state of the comparator data signal and the latched state of the comparator.

CMP1D – Read-only bit that returns the dynamically changing state of comparator 1. This bit reads zero whenever the comparator is disabled.

CMP0D – Read-only bit that returns the dynamically changing state of comparator 0. This bit reads zero whenever the comparator is disabled.

CMP1L – Bit that reads the latch output for comparator 1. This bit is cleared either by a writing a zero to it or by a rising edge of the comparator 0 LUT, depending on the state of the CRST1 bit in the CMP_CR1 register.

CMP0L – Bit that reads the latch output for comparator 0. This bit is cleared either by a writing a zero to it or by a rising edge of the comparator 1 LUT, depending upon the state of the CRST0 bit in the CMP_CR1 register.

Table 4. Resource CMP_MUX: Bank 0, reg[79h] Comparator Multiplexer Register

Bit	7	6	5	4	3	2	1	0
CMP_MUX	INP1		INN1		INP0		INN0	

This register contains control bits for input selection of comparators 0 and 1.

Table 5. INPx – Comparator x Positive Input Select

INPx	Value
Analog Global Mux Bus	00b
P0[1]	10b
P0[3]	11b

Table 6. INNx – Comparator x Negative Input Select

INNx	Value
VREF (1.3V)	00b
Ref Lo (approximately 0.9V)	01b
Ref Hi (approximately 1.8V)	10b

Table 7. Resource CMP_CR0: Bank 0, reg[7Ah] Comparator Control Register 0

Bit	7	6	5	4	3	2	1	0
CMP_CR0	Reserved		CMP1R	CMP1EN	Reserved		CMP0R	CMP0EN

This register enables and configures the input range of the comparators.

Table 8. CMPxEN

Value	Description
0b	Comparator disabled, powered off.
1b	Comparator enabled.

Table 9. CMPxR

Value	Description
0b	Comparator set to rail-to-rail input range, with approximately 20 µA cell current. Used for Normal Power mode.
1b	Comparator set to limited input range (Vss to Vdd - 1V), with approximately 10 µA cell current. Used for Low Power mode.

Table 10. Resource CMP_ CR1: Bank 0, reg[7Bh] Comparator Control Register 1

Bit	7	6	5	4	3	2	1	0
CMP_CR1	CINT1	CPIN1	CRST1	CDS1	CINT0	CPIN0	CRST0	CDS0

This register is used to configure the comparator output options.

CINTx – This bit selects comparator x for input to the analog interrupt. Note that if both CINT1 and CINT0 are set high, a rising edge on either comparator output may cause an interrupt.

Table 11. CINTx

Value	Description
0b	Comparator x does not connect to the analog interrupt.
1b	Comparator x connects to the analog interrupt. A rising edge will assert that interrupt, if it is enabled in the INT_MSK0 register.

CPINx – This bit selects the Comparator x signal for possible connection to the GPIO pin. Connection to the pin also depends on the configuration of the OUT_P1 register.

Table 12. CPINx

Value	Description
0b	Select Comparator x LUT output.
1b	Select Comparator x Latch output.

CRSTx – This bit selects the source for resetting the Comparator x latch.

Table 13. CRSTx

Value	Description
0b	Reset by writing a '0' to the CMP_RDC register's CMPxL bit.
1b	Reset by rising edge of Comparator x LUT output.

CDSx – This bit selects the data output for the comparator x channel, for routing to the capacitive sense logic and comparator x interrupt.

Table 14. CDSx

Value	Description
0b	Select the Comparator x LUT output.
1b	Select the Comparator x Latch output.

Table 15. Resource CMP_LUT: Bank 0, reg[7Ch] Comparator LUT Register

Bit	7	6	5	4	3	2	1	0
CMP_LUT	LUT1				LUT0			

This register selects the logic function.

LUTx – Selects 1 of 16 logic functions for output of comparator bus x. A = Comp1 output, B = Comp0 output.

Table 16. LUTx

Value	Description	Value	Description
0h	FALSE	8h	A NOR B
1h	A AND B	9h	A XNOR B
2h	A AND ~B	Ah	~B
3h	A	Bh	A OR ~B
4h	~A AND B	Ch	~A
5h	B	Dh	~A OR B
6h	A XOR B	Eh	A NAND B
7h	A OR B	Fh	TRUE

Version History

Version	Originator	Description
1.1	DHA	Added Version History

Note PSoC Designer 5.1 introduces a version history in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.

Copyright © 2012 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.