比较器数据手册 CMP V 1.1

Copyright © 2014 Cypress Semiconductor Corporation. All Rights Reserved.

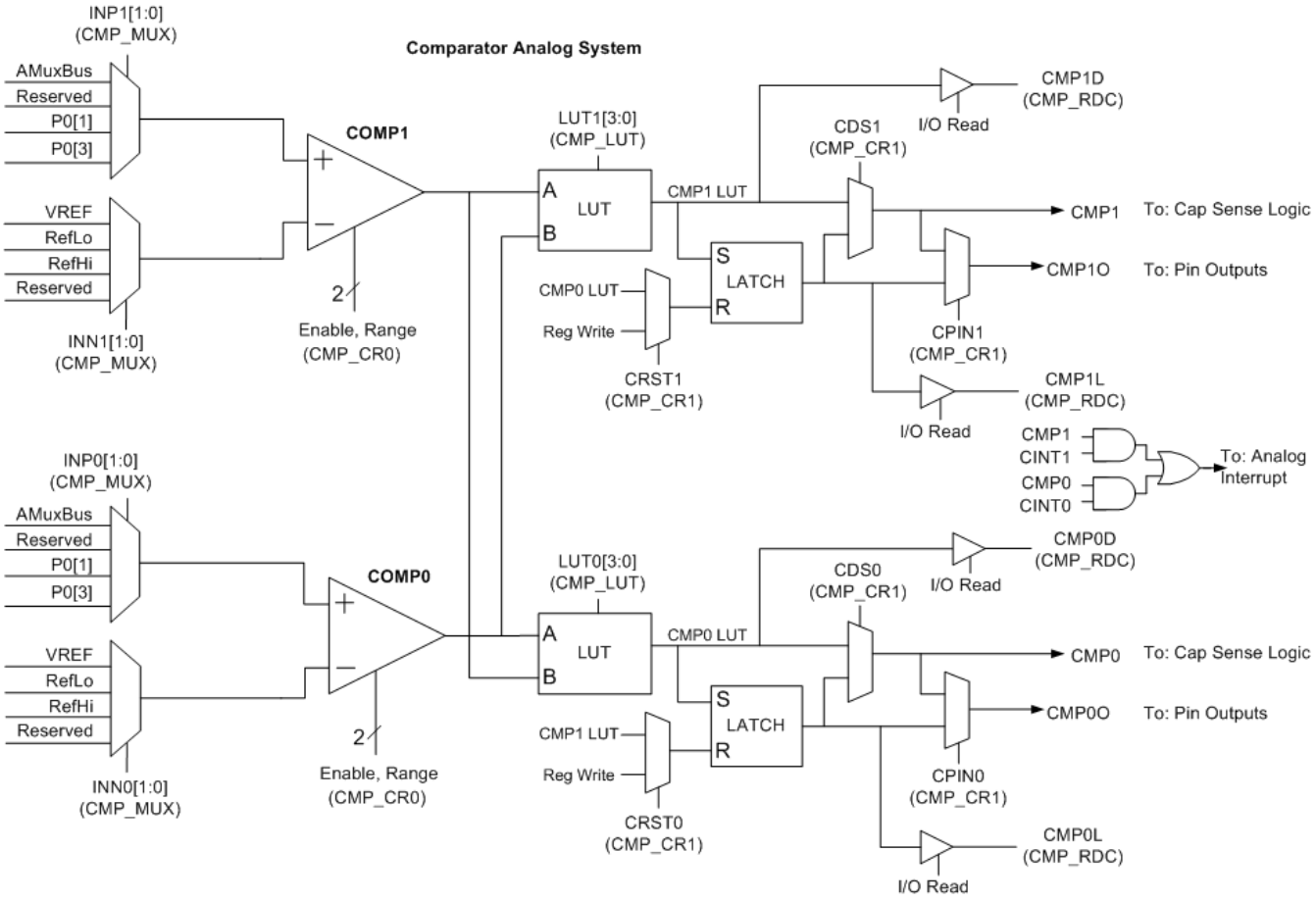
资源	PSoC® 模块			API 存储器 （字节）		引脚数量 （每个外部 I/O）
	CapSense®	I2C/SPI	定时器	闪存 ±3%	RAM	
CY8C20xx7、 CYRF89435、 CYRF89535						
	–	–	–	141	0	0 至 2

特性和概述

- 可选的正向和负向输入
- 直接连接至中断控制器
- 可编程的功耗

CMP 用户模块具有两个比较器，每个比较器都能对两个可选模拟输入信号进行比较。将两个输入中的一个作为参考电压。输入和输出的功能与布线都是灵活的。比较器的输出和相邻比较的输出器可以与任意两个输入的逻辑函数进行组合。此外，您可以将该组合输出与锁存值结合起来，并将它路由到某个引脚的输出或中断控制器。 CMP 用户模块控制了输入多路复用器和比较器。放置用户模块后，组合逻辑会自动进行配置，但是可以通过器件编辑器更改。

图 1. CMP 框图



功能说明

CMP 用户模块包括：

- 两个具有双端口输入的比较器
- 全局模拟总线输入
- 正向输入 （PMux）或一个 CMP （或两个）可选低通滤波器
- 负向输入 （NMux）端的三个可选参考电压
- 输出端可配置的组合逻辑

它们被禁用时，比较器没有功耗。两个活动模式提供了一个轨至轨的全输入范围以及一个有限输入范围稍微小的电源选项。正向输入电压高于负向输入电压时，比较器的输出为高电平。

当放置用户模块时，可配置的组合逻辑会自动设置一个默认的配置。默认情况下不会使能锁存。通过配置 CMP 用户模块，可以将比较器的结果传输到输出端。如果需要其他功能，可以通过 GUI 接口来配置组合逻辑。

输出组合逻辑是由 CMP LUT 和锁存逻辑组件的。每个 LUT 包括：两个输入、比较器 （CMPx）的输出以及相邻比较器 （CMPy）的输出。可通过配置某个 CMP LUT 实现一个两输入逻辑功能 （共十六个）。有关的可能配置详细信息，请参见下表。锁存逻辑在 SysClk 的上升时钟沿上锁存 CMPx 的输出。通过使用其他 CMP LUT 输出的上升沿或者将 0 写入 CMP_RDC 寄存器的 CMP0L 位，可以复位锁存器。您可以使用 CMP_ClearLatchx() API 函数。请参考 PSOC Designer 文件。

这两种比较器均都可用于生成单一的中断。寄存器的 CINT0 和 CINT1 位是 AND 门的输入端，由此结合为输出到模拟中断的 OR 门组合。将在器件编辑器中设置输出中断逻辑。有关器件编辑器中数字互连视图使用情况的详细信息，请参考 *“PSoC Designer 集成开发环境用户指南 ”*。

表 1. 16 个双输入布尔逻辑函数

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

直流和交流电的电气特性

表 2. CY8C20xx7 的比较器电气规范

参数	说明	最小值	典型值	最大值	单位	注意
T_{COMP}	比较器响应时间		70	100	ns	过压值为 50 mV
V_{OS}	输入偏移电压		2.5	30	mV	
I_{DD}	供电电流		20	80	µA	平均直流电流，过压值为 50 mV
PSRR	供电电压 > 2 V		80		dB	电源抑制比
	供电电压 < 2 V		40		dB	
V_{CM}	正常模式的输入电压范围	0		1.5	V	

放置

当用户模块实例化以后，用户模块所用的各模块将自动放置；而没有提供备选放置方式。

参数和资源

PMux0 和 PMux1

通过该参数可控制比较器 0 （PMux0）和比较器 1 （PMUX1）的正输入选择。可选择模拟复用器总线 （AnalogMuxBus）或两个引脚 （Port_0_1 或 Port_0_3）中的一个作为输入。可单独选择每个复用器的连接，所以如果需要，可将 PMux0 和 PMux1 输入互连。

PMux0 和 PMux1 的可能正向输入值：

- AnalogMuxBus （默认）
- Port_0_1
- Port_0_3

NMux0 和 NMux1

通过该参数可控制比较器 0 （NMux0）和 1 （NMux1）的可选负向输入。负向输入可能来自三个不同的参考电压：参考电压 （VREF， 1.3 V）、参考电压 Lo （Ref Lo，约 0.9 V）或参考电压 Hi （Ref Hi，约 1.8 V）。您可以单独连接每个复用器，由此如果需要可以连接 NMux0 和 NMux1 输入。

NMux0 和 NMux1 的可能负向输入值：

- VREF （默认）
- RefLo
- RefHi

中断生成控制

以下两项参数 InterruptAPI 和 IntDispatchMode 仅在 PSoC Designer 内将使能**中断生成控制**复选框选中时进行访问。该选框位于 **Project > Settings > Chip Editor** 路径下。

InterruptAPI （高级，默认 = Enable （使能））

使用 InterruptAPI 参数时，能够有条件地生成用户模块的中断处理程序和中断矢量表条目。选择 **Enable** （使能）以生成中断处理程序和中断矢量表条目。选择 **Disable** （禁用）以取消生成中断处理程序和中断矢量表条目。如果必须生成一个中断调度代码，那么请选择 **Enable**。不需要减少开销时，请禁用它。对于项目有多个覆盖层，其中一个模块资源由不同的覆盖层使用，这个尤其重要。

IntDispatchMode （高级，默认情况下为 ActiveStatus）

IntDispatchMode 参数用于指定中断请求的处理方式，这些中断由同一模块内不同覆盖层的多个用户模块所共享。选择 **ActiveStatus** 会导致固件在处理共享的中断请求之前，会测试哪一个外覆层正处于活动状态。每次共享的中断发出请求时，都会进行该测试操作。这样会加长延迟，还会造成为共享中断请求提供服务的过程不确定，但不需要使用任何 RAM。如果选择了 **OffsetPreCalc** 参数，则固件只有在初始加载一个外覆层时，才会计算共享中断请求的来源。这种计算可减少中断延迟，并产生为共享中断请求提供服务的确定过程，但会占用一个字节的 RAM 空间。

应用编程接口 （API）

通过 API 函数，设计人员能够使用一个更高层次交互用户模块。本节指定每个函数的接口，以及 “include” 文件所提供的相关常量。

注意：

在所有用户模块 API 中，通过调用 API 函数可以更改 A 和 X 寄存器的值。如果调用后需要 A 和 X 的值，则调用函数要在执行调用前保留 A 和 X 的值。选择此 “ 寄存器易失 ” 策略是为了提高效率。 C 编译器会自动遵循该要求。汇编语言程序员也必须确保其代码遵循该策略。虽然一些用户模块 API 函数可以保持 A 和 X 不变，但是无法保证它们将来也会如此。

对于大型存储器模型器件，调用程序必须保留 CUR_PP、 IDX_PP、 MVR_PP 以及 MVW_PP 寄存器中的所有值。尽管一些寄存器现在未被修改，但是在将来的版本中会改变这种情况。

CMP_Start0

说明：

设置比较器 0 的状态 （关闭、正常功耗或低功耗）。

C 原型：

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

汇编程序：

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

参数：

bMode：用于设置的模块值。

符号名称	数值
CMP_CMP0MODE_OFF	0x00
CMP_CMP0MODE_NORMALPOWER	0x01
CMP_CMP0MODE_LOWPOWER	0x03

返回值：

无

其他影响：

此函数可能会更改 A 和 X 寄存器。

CMP_Start1

说明：

设置比较器 1 的状态 （关闭、正常功耗或低功耗）。

C 原型：

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

汇编程序：

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

参数：

bMode：用于设置的模块值。

符号名称	数值
CMP_CMP1MODE_OFF	0x00
CMP_CMP1MODE_NORMALPOWER	0x10
CMP_CMP1MODE_LOWPOWER	0x30

返回值：

无

其他影响：

此函数会更改寄存器 A 和 X。

CMP_Stop0

说明：

使比较器 0 掉电。不驱动输出端。

C 原型：

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

汇编程序：

```
lcall CMP_Stop0
```

参数：

无

返回值：

无

其他影响：

此函数会更改 A 和 X 寄存器。

CMP_Stop1

说明：

使比较器 1 掉电。不驱动输出端。

C 原型：

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

汇编程序：

```
lcall CMP_Stop1
```

参数：

无

返回值：

无

其他影响：

此函数会更改 A 和 X 寄存器。

CMP_EnableInt

说明：

使能模拟中断，可以使用比较器 0 或比较器 1 来驱动中断，但不能同时使用这两个比较器。

C 原型：

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

汇编程序：

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

参数：

bComparator — 比较器 0 或比较器 1 （MP_CR1 寄存器的 CINT0 或 CINT1）的掩码用于设置驱动模拟中断的比较器。如果其他任意值 （除下面列出的数值外，例如 0x88 （CMP_CINT0_MASK | CMP_CINT1_MASK）或 0x00）被传输给该函数，则将使能模拟中断，但任意比较器都不会驱动中断。

符号名称	数值
CMP_CINT0_MASK	0x08
CMP_CINT1_MASK	0x80

返回值：

无

其他影响：

此函数会更改寄存器 A 和 X。

CMP_DisableInt

说明：

禁用模拟中断和比较器中断选择，因此 CINT0 或 CINT1 都不能驱动中断。

C 原型：

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

汇编程序：

```
lcall CMP_DisableInt
```

参数：

无

返回值：

无

其他影响：

此函数会更改 A 和 X 寄存器。

CMP_ClearInt

说明：

清除已发布的模拟中断。

C 原型：

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

汇编程序：

```
lcall CMP_ClearInt
```

参数：

无

返回值：

无

其他影响：

此函数会更改 A 和 X 寄存器。

CMP_ClearLatch0

说明：

通过分别清除 CMP_CR1 寄存器的 CRST0 位和 CMP_RDC 寄存器的 CMP0L 位来清除比较器 0 的锁存输出。有关比较器锁存输出的详细信息，请参见 “ 技术参考手册 （TRM）”。

C 原型：

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

汇编程序：

```
lcall CMP_ ClearLatch0
```

参数：

无

返回值：

无

其他影响：

此函数会更改 A 和 X 寄存器。

CMP_ClearLatch1

说明：

通过清除 CMP_CR1 寄存器的 CRST1 位，然后清除 CMP_RDC 寄存器的 CMP1L 位，清零比较器 0 的锁存输出。有关比较器锁存输出的详细信息，请参见 “ 技术参考手册 （TRM）”。

C 原型：

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

汇编程序：

```
lcall CMP_ ClearLatch1
```

参数：

无

返回值：

无

其他影响：

此函数会更改寄存器 A 和 X。

CMP_ChangeMuxes0

说明：

仅更改了比较器 0 负向输入 （NMux0）和正向输入 （PMux0）选择。

C 原型：

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

汇编程序：

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

参数：

bMuxSetting：对 PMux0 位和 NMux0 位进行 OR （或）运算。如果没有对 PMux0 和 NMux0 的值进行 OR （或）运算，则未被传入的值为 0x00。例如，如果仅传输 CMP_PMUX0_PIN_0_1，将 NMux0 的比较器输入设置为 0x00，既为 CMP_NMUX0_VREF_1P3V。

PMux0 符号名称	数值	NMux0 符号名称	数值
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

返回值：

无

其他影响：

此函数会更改 A 和 X 寄存器。

CMP_ChangeMuxes1

说明：

仅改变比较器 0 的负向 （NMux1）和正向 （PMux1）输入选择。

C 原型：

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

汇编程序：

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

参数：

bMuxSetting：进行 PMux1 位和 NMux1 位的 OR （或）运算以设置该参数。如果 PMux1 和 NMux1 的值未进行 OR （或）运算，然后没有被传入的值将设置为 0x00。例如：如果仅 CMP_PMUX-0_PIN_0_1 传入， NMux1 的比较器输入被设置为 0x00，这就是 CMP_NMUX1_VREF_1P3V。

PMux1 符号名称	数值	NMux1 符号名称	数值
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

返回值：

无

其他影响：

此函数会更改寄存器 A 和 X。

CMP_bStatus

说明：

通过读取 CMP_READ_CLEAR_REG （CMP_RDC）寄存器，返回比较器数据信号和 “ 锁存状态 ” 信号的状态。数据信号便是框图中的 CMPxD 信号。锁存状态是同一个图形中的 CMPxL 信号。根据比较器输出逻辑的设置情况，该状态与中断状态不一样。

C 原型：

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

汇编程序：

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

参数：

无

返回值：

包含了表示框图中 CMPxD 和 CMPxL 信号输出的位。这些位在它们的本地位置上。返回值在累加器中返回。

掩码名称	数值	说明
CMP_CMP0D_MASK	0x10	比较器 0 数据信号
CMP_CMP0L_MASK	0x01	比较器 0 的锁存状态
CMP_CMP1D_MASK	0x20	比较器 1 的数据信号
CMP_CMP1L_MASK	0x02	比较器 1 的锁存状态
CMP_STATUS_MASK	0x33	两个比较器的数据信号和 “ 锁存状态 ” 信号的掩码。

其他影响：

此函数会更改寄存器 A 和 X。

固件源代码示例

该示例代码首先会清除已生成的模拟中断，然后使能模拟和全局中断，并将比较器 0 设置为模拟中断的输入。然后将这两个比较器设为正常功耗模式并启动它们。在环路中，它会检查比较器的状态，同时检查中断计数器的值，以确定该值是否到达极限；如果该值等于极限值，那么会将引脚输出驱动为高电平，然后将其降低为低电平。

注意： 请将芯片编辑器中所需输出引脚重新命名为 “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();
}
```

以汇编语言编写的相同代码如下：

```
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
```

配置寄存器

比较器的基本拓扑结构为已选的模拟 CT 模块设置配置寄存器中的位。

表 3. CMP_RDC 资源：组 0， reg[78h] 比较器读取 / 清除寄存器

位	7	6	5	4	3	2	1	0
数值	保留		CMP1D	CMP0D	保留		CMP1L	CMP0L

该寄存器用于读取比较器数据信号的状态和比较器 “ 锁存状态 ” 信号的状态。

CMP1D — 为只读寄存器，它会返回比较器 1 的动态变化状态的位。当比较器被禁用时，读取该位得到零。

CMP0D — 为只读寄存器，它会返回比较器 0 的动态变化状态的位。当比较器被禁用时，读取该位得到零。

CMP1L — 该位读取比较器 1 的锁存输出。请根据 CMP_CR1 寄存器的 CRST1 位状态，通过向该位写入 ‘0’ 或通过比较器 0 LUT 的上升沿来清除该位。

CMP0L — 该位读取比较器 0 的锁存输出。请根据 CMP_CR1 寄存器的 CRST0 位状态，通过向该位写入 ‘0’ 或通过比较器 1 LUT 的上升沿来清除该位。

表 4. 资源 CMP_RDC：组 0， reg[79h] 比较器复用器寄存器

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

该寄存器包括各控制位，这样可以选择比较器 0 和 1 的输入端。

表 5. INPx — 比较器 x 的正向输入选择

INPx	数值
模拟全局复用器总线	00b
P0[1]	10b
P0[3]	11b

表 6. INPx — 比较器 x 的负向输入选择

INNx	数值
VREF （1.3 V）	00b
Ref Lo （约 0.9 V）	01b
Ref Hi （约 1.8 V）	10b

表 7. CMP_CR0 资源：组 0， reg[7Ah] 比较器控制寄存器 0

位	7	6	5	4	3	2	1	0
CMP_CR0	保留		CMP1R	CMP1EN	保留		CMP0R	CMP0EN

使用该寄存器可使能和配置比较器的输入范围。

表 8. CMPxEN

数值	说明
0b	比较器被禁用和掉电。
1b	使能比较器。

表 9. CMPxR

数值	说明
0b	将比较器的电压设置为轨至轨输入电压范围内，槽电流约为 20 µA。使用于正常功耗模式。
1b	将比较器的电压设置为优先输入电压范围内 （Vss 至 Vdd - 1V），槽电流约为 10µA。使用于低功耗模式。

表 10. 资源 CMP_ CR1：组 0， reg[7Bh] 比较器控制寄存器 1

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

该寄存器用于配置比较器的输出选项。

CINTx — 通过该位选择比较器 x，以便输入到模拟中断。请注意，如果 CINT1 和 CINT0 均被设置为高电平，则所有比较器输出上的上升沿会导致一个中断。

表 11. CINTx

数值	说明
0b	比较器 x 没有连接模拟中断。
1b	比较器 x 连接至模拟中断。如果在 INT_MSK0 寄存器中使能了该中断，则上升沿会激活它。

CPINx — 该位选择比较器 x 信号，以便能够连接至 GPIO 引脚。与引脚相连的操作也取决于 OUT_P1 寄存器的配置情况。

表 12. CPINx

数值	说明
0b	选择比较器 x 的 LUT 输出。
1b	选择比较器 x 的锁存输出。

CRSTx — 通过该位选择用于重新设置比较器 x 锁存器的源。

表 13. CRSTx

数值	说明
0b	通过向 CMP_RDC 寄存器的 CMPxL 位写入 ‘0’，复位该寄存器。
1b	通过比较器 x LUT 输出的上升沿进行复位。

CDSx — 使用该位可选择比较器 x 通道的数据输出，用于路由到电容式传感逻辑和比较器 x 中断。

表 14. CDSx

数值	说明
0b	选择比较器 x 的 LUT 输出。
1b	选择比较器 x 的锁存输出。

表 15. 资源 CMP_CR0：组 0， reg[7Ch] 比较器 LUT 寄存器

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

该寄存器可选择逻辑函数。

LUTx — 为比较器总线 x 的输出选择 16 个逻辑函数中的一个。 A = Comp1 输出， B = Comp0 输出。

表 16. LUTx

数值	说明	数值	说明
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

版本历史记录

版本	创作者	说明
1.1	DHA	添加了版本历史

注意： PSoC Designer 版本 5.1 在所有用户模块数据手册中都介绍了 “ 版本历史记录 ”。本数据手册详细介绍当前和先前用户模块版本之间的区别。

Copyright © 2014 赛普拉斯半导体公司。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品内嵌的电路外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不会根据专利权或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于合理预计会发生运行异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯将不批准将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

PSoC Designer™ 和 Programmable System-on-Chip™ 是赛普拉斯半导体公司的商标， PSoC® 是赛普拉斯半导体公司的注册商标。此处引用的所有其他商标或注册商标归其各自所有者所有。

所有源代码 （软件和 / 或固件）均归赛普拉斯半导体公司 （赛普拉斯）所有，并受全球专利法规 （美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。赛普拉斯据此向获许可者授予适用于个人的、非独占性、不可转让的许可，用以复制、使用、修改、创建赛普拉斯源代码的派生作品、编译赛普拉斯源代码和派生作品，并且其目的只能是创建自定义软件和 / 或固件，以支持获许可者仅将其获得的产品依照适用协议规定的方式与赛普拉斯集成电路配合使用。除上述指定用途外，未经赛普拉斯的明确书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。

免责声明：赛普拉斯不针对该材料提供任何类型的明示或暗示保证，包括 （但不仅限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不另行通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于合理预计可能发生运转异常和故障，并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而导致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

产品使用可能受适用于赛普拉斯软件许可协议的限制。