# 14-Bit 增量型 ADC 数据表 ADCINC14 V 1.4



| 资源 | PSoC® 模块 | | | API 内存 （字节） | | 引脚 （根据外部 I/O） |
|---|---|---|---|---|---|---|
| | 数字 | 模拟 CT | 模拟 SC | 闪存 | RAM | |
| CY8C29/27/24/22xxx、CY8C23x33、CY8CLED04/08/16、CY8C28x45、CY8C28x43、 CY8C28x52 | | | | | | |
| | 4 | 0 | 1 | 266 | 6 | 1 |

请参阅 AN2239 以了解其他转换器。

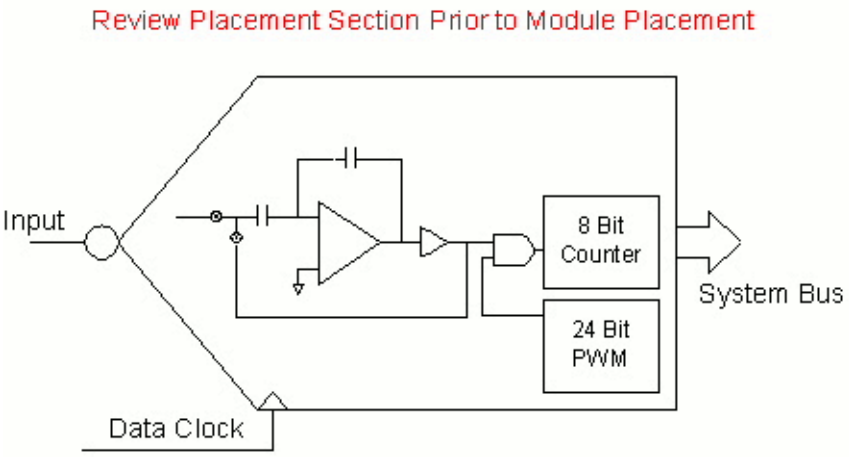
## 功能和概述

- 14-bit 分辨率、2 的补码
- 2 到 120 sps 的采样率
- Vss 到 Vdd 的输入范围
- 积分转换器提供良好的正常模式抑制
- 内部或外部时钟

ADCINC14 是具有 14 位分辨率的积分 ADC。 它可以配置为通过优化积分时间来删除不需要的高频率。 可以通过配置适当的参考电压和模拟接地来测量输入电压范围 （包括轨到轨）。 根据以 AGND 为中心的 -Vref 和 +Vref 之间的输入电压，可以选择有符号和无符号的结果格式。

根据 DataClock 和 CalcTime 参数的选择，采样率可以实现从 2 到超过 120 sps 变化。 编程接口允许您指定要采用的连续样品数或者选择连续采样。 CPU 负荷因输入电平而异。 例如，当 Vin = +Vref 时，有 9832 个 CPU 周期（最大 13 位）。 当 Vin = AGND 时，有 5076 个 CPU 周期。 当 Vin = -Vref 时，有 360 个 CPU 周期。

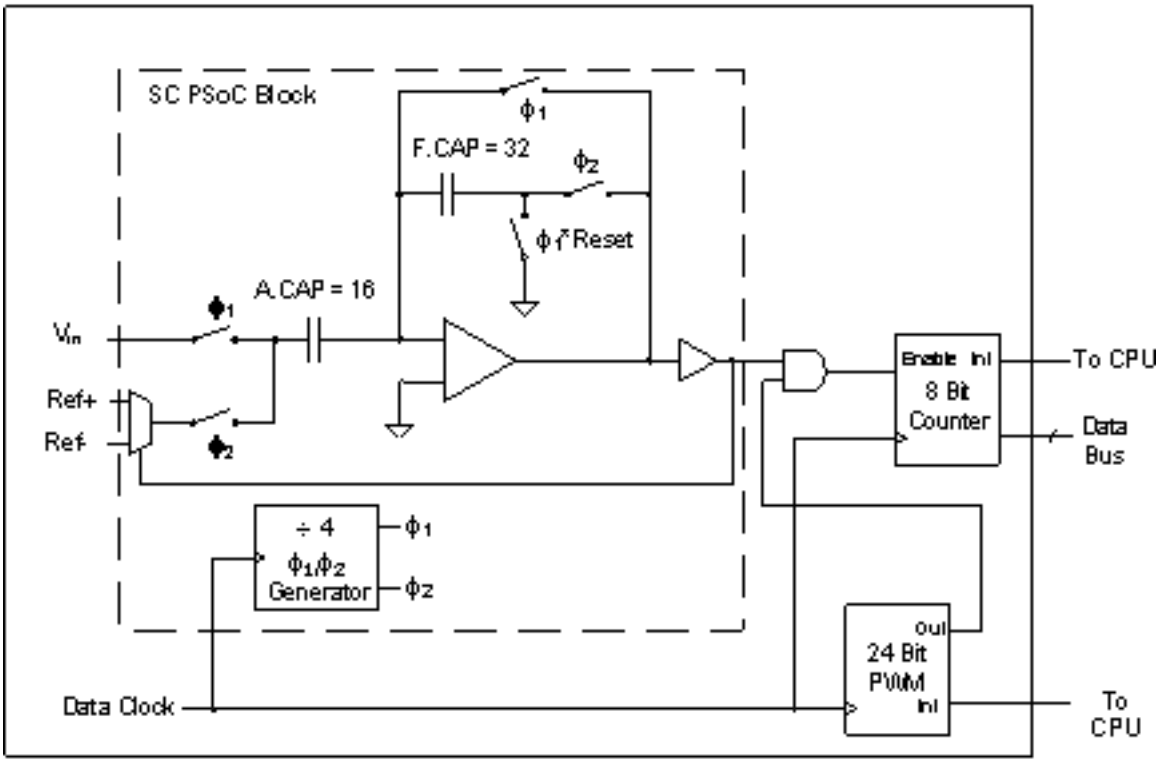
Figure 1. ADCINC14 框图



# 功能说明

ADCINC14 由一个模拟开关电容 PSoC 模块和四个数字 PSoC 模块组成，如图 2 所示：

Figure 2. ADCINC14 的简化图示




模拟模块配置为可复位积分器。 根据输出极性，对参考控制进行了配置，以便可以从输入中增减参考电压，并置于积分器中。 此参考控制尝试将积分器输出拉回到 AGND。 如果积分器运行 $2^{14}$ 次，而输出电压比较器在这些次数中结果为正的次数为 “n”，则输出残余电压 ($V_{resid}$) 为：

**Equation 1**

$$V_{resid} = 2^{14} \times V_{in} - n \times V_{Ref} + (2^{14} - n) \times V_{Ref}$$

**Equation 2**

$$V_{in} = \frac{n - 2^{14-1}}{2^{14-1}} \times V_{Ref} + \frac{V_{resid}}{2^{14}}$$

此等式说明了此 ADC 的范围为 ± $V_{ref}$，分辨率 (LSB) 为 $V_{ref}/2^{14-1}$，计算结束时输出电压定义为残余电压。 由于 $V_{resid}$ 始终小于 $V_{ref}$，因此 $V_{resid}/2^{14}$ 小于 LSB 的一半，可以忽略。

生成的等式为等式 3：

**Equation 3**

$$V_{in} = \frac{n - 8192}{8192} V_{ref}$$

## 示例 1

如果 $V_{ref}$ 为 1.3V，我们可以根据在数据就绪时从增量型 ADC 读取的值，轻松计算出输入电压。 此计算是使用等式 4 完成的：

**Equation 4**

$$V_{in} = \frac{n - 8192}{8192} 1.3$$

计算结果以 AGND 为参考。 如果 ADC 数据值为 10000，则测量的电压计算为 0.29V。此计算可以在等式 5 中完成：

**Equation 5**

$$V_{in} = \frac{10000 - 8192}{8192} 1.3 = 0.29 V$$

计算出的值为理想值，很有可能因系统噪声和芯片偏移而有所不同。

要给定特定输入电压的情况下确定预计的代码，可以重新排列该等式，计算出下列结果：

**Equation 6**

$$n = \frac{8192 \cdot V_{in}}{V_{ref}} + 8192$$

## 示例 2

如果 $V_{ref}$ 为 1.3V，可以根据输入电压轻松计算出预计的 ADC 代码。 此计算是使用等式 7 完成的：

**Equation 7**

$$n = \frac{8192 \cdot V_{in}}{V_{ref}} + 8192$$

如果输入电压高于 AGND 1V，则 ADC 的代码预计为 14493。 这是根据等式 8 计算的：

**Equation 8**

$$n = \frac{8192 \cdot 1}{1.3} + 8192 = 14493$$

计算出的值为理想值，很有可能因系统噪声和芯片偏移而有所不同。

为了将积分器函数作为增量型 ADC，使用了下列数字资源：

- 8-bit 计数器，用于累计输出为正的周期数。
- 17-bit PWM，用于计算积分时间和控制进入 8-bit 计数器的时钟。

单一 DataClock 连接到 8-bit 计数器、17-bit PWM （从 24-bit PWM 实现）和模拟列时钟 （该时钟连接到模拟 SC PSoC 模块）。 模拟列时钟实际为两个时钟 $\Phi_1$ 和 $\Phi_2$，它们都从数据时钟生成。 这两个附加时钟的频率刚好是数据时钟频率的四分之一。 这意味着 PWM 和计数器的运行速度比所需速度快 4 倍，因此需要累计相当于 16 位的数据。

**Note** 放置此模块时，必须为所有三个模块配置同一个时钟。 不这样做会导致其运行不正常。

计数器是通过用于 LSB 的 8-bit 数字时钟和用于 MSB 的软件计数器实现的。 每当硬件计数器溢出时，都会生成中断，计数器的高 MSB 会递增。 这使得 ADCINC14 模块只用四个数字模块而非五个数字模块来实现。

采样率为 DataClock 除以积分时间再加上它进行结果计算所需的时间 CalcTime。 积分时间是 ADCINC14 对输入信号进行采样的时间。

**Equation 9**

$$SampleRate = \frac{DataClock}{2^{Bits+2} + CalcTime}$$

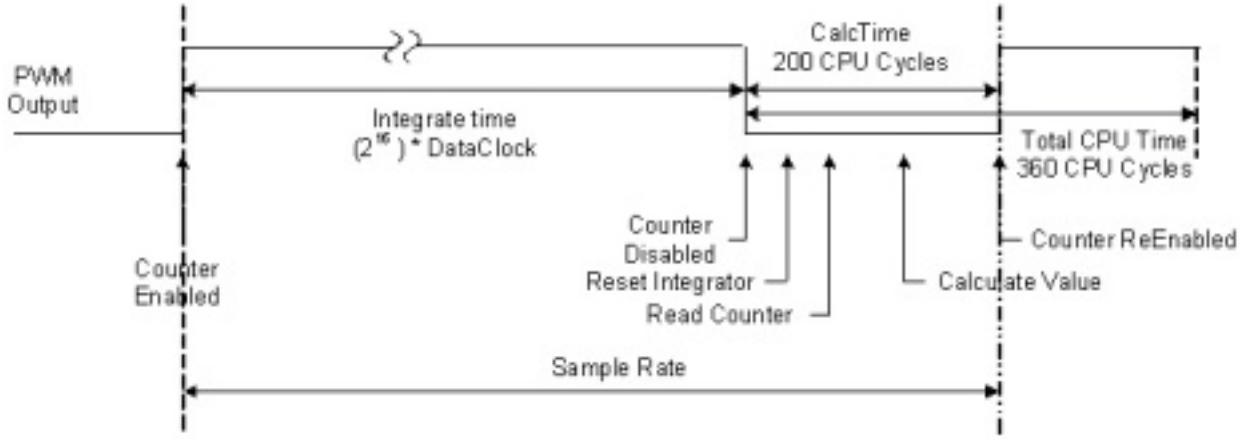
计算 CalcTime 结果所耗费的时间与 CPU 时钟成反比变化。 CalcTime 必须设置为大于计算结果所需的值。 最小 CalcTime 等于 200 个 CPU 周期，且必须以 DataClock 的形式表示。 CalcTime 还可以增加到最小值之上以优化采样率。

**Note** $2^{16}$ 与 CalcTime 的和不得超过 $2^{24}$-1 或 16,777,215。

**Equation 10**

$$CalcTime \geq \frac{DataClock * 200}{CPUClock}$$

Figure 3. 与 PWM 输出相关的 ADCINC14 时序




24-bit PWM 编程为输出高电平信号，该信号为 $2^{16}$ 乘以 DataClock。 在进行最小结果计算和复位积分器所耗费的时间过程中，PWM 输出为低电平。 还可以调整此低电平时间以帮助提供更准确的采样率和 DataClock。 PWM 总周期是积分时间与 CalcTime 之和。

当启动第一个读数时，将计算 PWM 配置，对积分器进行复位，并且计数器都复位为 FFh。 初始延迟始终至少等于计算的延迟时间。 PWM 仅在第一次读取之前初始化。 比较寄存器和周期寄存器一旦设置以后，就不需要对其进行重新初始化。 当 PWM 计数小于或等于积分值时，输出变为高电平，使得 8-bit 计数器倒计数。 PWM 将保持高电平输出直到计数器达到零。 此时，禁用 8-bit 计数器的时钟，并生成 PWM 中断。

此 8-bit 软件计数器的初始值设置为最大负数的 $2^{14}/64$ 倍。 8-bit 计数器每次溢出时，将执行 8-bit 计数器的中断，软件计数器 (MSB) 按 1 递增。

如果 ADC 的输入大于或等于最大正值，8-bit 计数器将在 DataClock 每次正跃变时递增。 如果 ADC 的输入小于或等于最大负输入值，8-bit 计数器将永远不再递减，因此不再生成中断。 理想情况下，接近模拟接地的输入将允许计数器递增一半时间。 很容易看出，根据输入电压电平的不同，8-bit 计数器中的中断数将在 0 到 $(2^{16})/256$ 个之间变化。 这意味着在积分周期内，处理器最多可以中断 65536/256 （即 256）次。

由于获取更高分辨率结果的采样时间较长，因此在处理采样的过程中，期望处理器等待是不切实际的。 ADC 子程序与主程序之间的主要通信是一个可轮询的标志。 当 ADCINC14_bfStatus 的最低有效位为非零值时，ADCINC14_iResult. API 中的新数据可用于检查数据标志和检索数据。

此数据处理程序设计基于轮询。 如果需要基于中断的数据处理程序，您可以将自己的数据处理程序代码插入到中断子程序 ADCINC14_CNT_ISR，此子程序位于程序集文件 *ADCINC14INT.asm*. 最佳插入代码的位置已做明显标记。

## CPU 使用率

ADCINC14 需要 CPU 时间来计算结果，并在每次硬件计数器溢出时递增软件计数器。 CPU 开销取决于三个变量： CPU 时钟、DataClock 和输入电压。 注意，最初输入电压影响 ADC 的 CPU 开销似乎很奇怪。 接近或低于 -Vref 的输入电压需要很少的 CPU 开销。 接近或大于 +Vref 的输入电压需要较多的 CPU 开销。 对于给定输入，请使用下列等式计算 CPU 周期：

**Equation 11**

$$CPUcycles = PWM16\_IRQ\_CPUcycles + \left(\frac{2^{14}}{64} * \left(\frac{Vref + Vin}{2 * Vref}\right) * Counter\_IRQ\_CPUcycles\right)$$

**Equation 12**

$$CPUcycles = 360 + \left(\frac{2^{14}}{64} * \left(\frac{Vref + Vin}{2 * Vref}\right) * 37\right)$$

要计算最大 CPU 周期，请将 Vin 设置为 Vref。

**Equation 13**

$$CPUcycles = 360 + \left(\frac{2^{14}}{64} * \left(\frac{Vref + Vref}{2 * Vref}\right) * 37\right) = 360 + (256 * 1 * 37) = 9832$$

要计算 ADCINC14 的 CPU 使用率百分比，可以使用等式 14：

**Equation 14**

$$Percent\_CPU\_Utilization = \frac{Sample\_Rate * CPUcycles}{CPU\_freqency} * 100$$

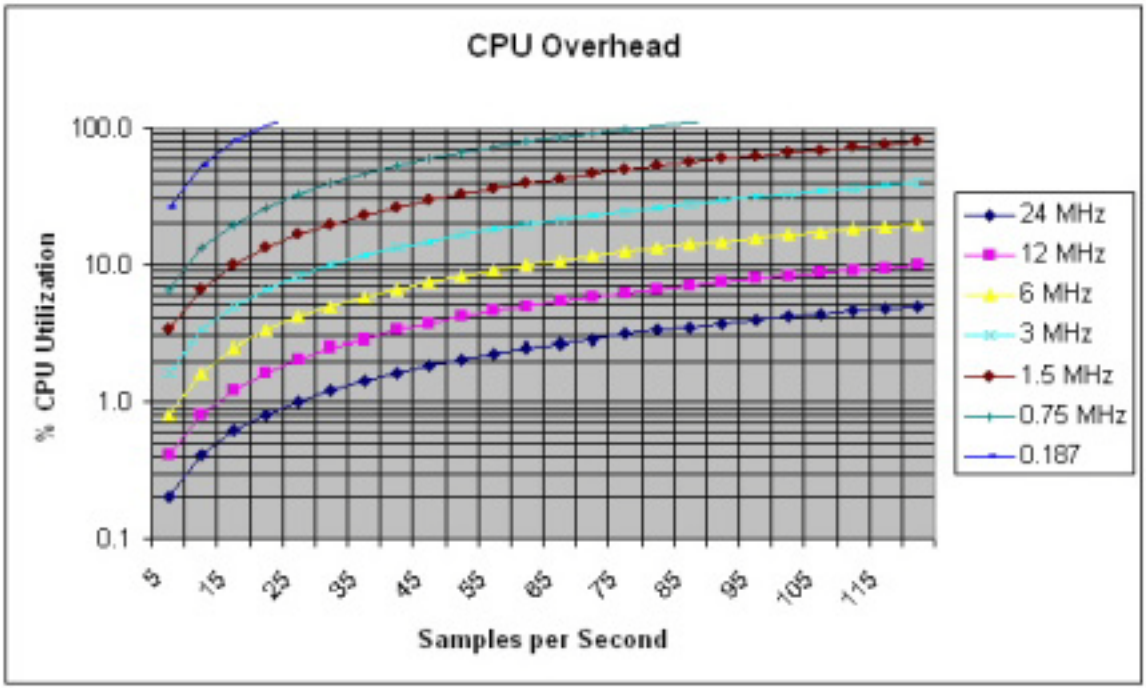
将采样率设置为 50 次采样 / 秒，CPU 时钟设置为 12 MHz，并使用 4% 的 CPU （等式 15）：

**Equation 15**

$$Percent\_CPU\_Utilization = \frac{50*9832}{12MHz}*100 = 4.1\%$$

图 4 显示了受支持采样率的 CPU 使用率和 CPU 时钟：

Figure 4. CPU 使用率、采样率和 CPU 时钟




## 频率抑制

通过选择适当的积分时间，可以抑制特定噪声源。 要抑制噪声源及其谐波，请选择等于噪声信号积分周期的积分时间。 如果抑制多个信号，请选择与这两个信号的积分周期相等的积分时间。

### 示例 1

如果需要抑制由 50 Hz 和 60 Hz 信号导致的噪声，请选择同时包含 50 Hz 和 60 Hz 信号的积分数字的周期。

**Equation 16**

$$IntegrateTime = 6*\frac{1}{60} = 5*\frac{1}{50} = 100mSec$$

积分时间为 100 ms 将同时抑制 50 Hz 和 60 Hz 以及这些信号的任何谐波。 接下来，计算生成适当的 IntegrateTime 所需的 DataClock。

**Equation 17**

$$DataClock = \frac{2^{16}}{IntegrageTime}$$

请注意，虽然 CalcTime 影响采样率，但是此计算中不使用它。 IntegrateTime 是 ADCINC14 实际对输入电压进行采样的周期。 采样率基于 IntegrateTime 以及计算结果需要的时间。

### 示例 2

对于给定应用场合，所需的 IntegrateTime 为 100 ms。 对于 100 ms 的 IntegrateTime，数据时钟必须为：

**Equation 18**

$$DataClock = \frac{2^{16}}{IntegrateTime} = \frac{65536}{100msec?} = 655.4kHz$$

与数据时钟相关的 CalcTime 必须根据 DataClock 和 CPU 时钟计算得出。 如果 CPU 时钟为 12 MHz，则最短计算时间为：

**Equation 19**

$$CalcTime = \frac{DataClock \cdot 200}{CPU\_Clock} = \frac{655.4kHz \cdot 200}{12000kHz} \cong 10.9\,DataClocks$$

此 CalcTime 应当四舍五入到最近的整数值，在此示例中为 “11”。 要确定采样率，请使用等式 20：

**Equation 20**

$$SampleRate = \frac{DataClock}{2^{16} + CalcTime} = \frac{655.4kHz}{65536 + 11} = 10Samples/Second$$

如果需要较长的采样率，可以增加 CalcTime，直到 CalcTime + $2^{16}$ 小于或等于 $2^{24}$ - 1 (16,777,215)。 这不应当有太多限制。

## 直流和交流电气特性

以下值均为基于初始特性数据的预期性能指标。除非下表中另有指定，否则 $T_A$ =25° C，Vdd = 5.0V，功耗 = 高，运算放大器偏压 = 低，输出在 P2[6] 上有 1.25 外部 Vref 的情况下参考 P2[4] 上的 2.5V 外部模拟接地，分辨率设置为 14 位。

Table 1. 5.0V ADCINC14 直流和交流电气特性

| 参数 | 典型值 | 限制 | 单位 | 条件和注释 |
|---|---|---|---|---|
| 输入 | | | | |
| 输入电压范围 | --- | Vss 到 Vdd | | Ref Mux = Vdd/2 ± Vdd/2 |
| 输入电容[1] | 3 | --- | pF | |
| 输入阻抗 | 1/(° C*clk) | --- | Ω | |
| 分辨率 | --- | 14 | 位 | |
| 采样率 | | 2 到 60 | sps | |
| 信噪比 | 80 | --- | dB | |
| 直流精度 | | | | |
| 微分非线性误差 | .25 | --- | 最低有效位 | 列时钟 2.0 MHz |
| 积分非线性误差 | 1.5 | --- | 最低有效位 | |
| 偏移误差 | 4 | --- | mV | |
| 增益误差 | | | | |
| 包括参考增益误差 | 3.0 | -- | % FSR | |
| 不包括参考增益误差[2] | 0.1 | -- | % FSR | |
| 工作电流 | | | | |
| 低功耗 | 370 | --- | uA | |
| 中等功耗 | 750 | --- | uA | |
| 高功耗 | 2130 | --- | uA | |
| 数据时钟 | --- | 0.125 到 8 | MHz | 数字模块和模拟列时钟的输入 |

以下值均为基于初始特性数据的预期性能指标。除非下表中另有指定，否则 $T_A$ = 25° C，Vdd = 3.3V，功耗 = 高，运算放大器偏压 = 低，输入在 P2[6] 上有 1.25 外部 Vref 的情况下参考 P2[4] 上的 1.64V 外部模拟接地，分辨率设置为 14 位。

Table 2. 3.3V ADCINC14 直流和交流电气特性

| 参数 | 典型值 | 限制 | 单位 | 条件和注释 |
|---|---|---|---|---|
| 参数 | 典型值 | 限制 | 单位 | 条件和注释 |
| 输入 | | | | |
| 输入电压范围 | --- | Vss 到 Vdd | | Ref Mux = Vdd/2 ± Vdd/2 |
| 输入电容[1] | 3 | --- | pF | |
| 输入阻抗 | 1/(° C*clk) | --- | Ω | |
| 分辨率 | --- | 14 | 位 | |
| 采样率 | | 2 到 60 | sps | |
| 信噪比 | 80 | --- | dB | |
| 直流精度 | | | | |
| 微分非线性误差 | .25 | --- | 最低有效位 | 列时钟 2.0 MHz |
| 积分非线性误差 | 1.5 | --- | 最低有效位 | |
| 偏移误差 | 4 | --- | mV | |
| 增益误差 | 2.8 | --- | % FSR | |
| 包括参考增益误差 | 3.0 | -- | % FSR | |
| 不包括参考增益误差[2] | 0.4 | -- | % FSR | |
| 工作电流 | | | | |
| 低功耗 | 170 | --- | uA | |
| 中等功耗 | 540 | --- | uA | |
| 高功耗 | 1950 | --- | uA | |
| 数据时钟 | --- | 0.125 到 8 | MHz | 数字模块和模拟列时钟的输入 |

**Note** 电气特性说明

1. 包括 I/O 引脚。
2. 参考增益误差测量方法是将 $V_{RefHigh}$ 外部参考与通过测试复用器并返回至引脚的 $V_{RefLow}$ 进行比较。

## 放置

ADC 模块可以放置在任何开关电容 PSoC 模块中。 它们每个必须能够单独驱动其所在特定列的比较器总线。

计数器模块可以放置在任何可用数字模块中，但是 PWM24 只能放置在特定位置。 请参见下表以获得有效放置。

Table 3. 有效放置

| 部件系列 | 有效 PWM24 放置 LSB/ISB/MSB |
|---|---|
| CY8C24/22xxx | DBB00/DBB01/DCB02 |
| CY8C27xxx | DBB00/DBB01/DCB02、DCB03/DBB10/DBB11、<br>DBB10/DBB11/DCB12 |
| CY8C27X66 | DBB00/DBB01/DCB02, DBB01/DCB02/DCB03, DCB02/DCB03/DBB10,<br>DCB03/DBB10/DBB11, DBB10/DBB11/DCB12, DBB11/DCB12/DCB13 |
| CY8C29xxx | DBB00/DBB01/DCB02, DBB01/DCB02/DCB03, DCB02/DCB03/DBB10,<br>DCB03/DBB10/DBB11, DBB10/DBB11/DCB12, DBB11/DCB12/DCB13,<br>DCB12/DCB13/DBB20, DCB13/DBB20/DBB21, DBB20/DBB21/DCB22,<br>DBB21/DCB22/DCB23, DCB22/DCB23/DBB30, DCB23/DBB30/DBB31,<br>DBB30/DBB31/DCB32, DBB31/DCB32/DCB33 |
| CY8C26/25xxx | DBA00/1/2 和 DCA04/5/6 |

两个数字模块都有中断服务子程序。 希望 （但不是需要）计数器模块比 PWM24 模块具有更高的中断优先级。 因此，建议将计数器模块置于比 PWM24 模块更低的数字模块位置。

## 参数和资源

**输入**

在完成模拟 PSoC 模块放置后，再进行输入选择。 八个开关电容模块具有不同的输入选择。 每个模块都可以连接到最近的相邻模块，某些模块可以直接连接到外部输入引脚。 放置模拟模块时必须考虑如何获取其输入信号。 部分放置允许输入从封装引脚直接路由到输入。 这些直接连接允许精确测量供电轨 40 mV 范围内的输入。 信号还可以通过列复用器之一、CT 模块测试复用器之一路由，以及路由至 ADCINC14 还可以测量电源轨附近信号的模拟列上。

**ClockPhase**

时钟相位的选择用于将一个开关电容模拟 PSoC 模块的输出与另一个模块的输入同步。开关电容模拟 PSoC 模块使用两相时钟 （$\Phi_1$、$\Phi_2$）获取和传输信号。 通常，ADCINC14 的输入是在 $\Phi_1$ 上采样的，这是正常设置。 这便产生一个问题：许多用户模块在 $\Phi_1$ 期间将其输出自动归零，仅在 $\Phi_2$ 期间给出有效输出。 如果此类模块的输出馈送到 ADCINC14 的输入，则 ADCINC14 获取自动归零的输出而不是有效信号。 时钟相位选择允许交换相位，因此输入信号是在 $\Phi_2$ 期间获取的，这是交换设置。

### CalcTime

CalcTime 是下一积分周期前 CPU 用于计算中间积分结果所耗费的时间。 计算 “CalcTime” 结果所耗费的时间与 CPU 时钟成反比变化。此值必须是数据时钟的形式。最小 CPU 计算时间为 200 个 CPU 时钟。 还可以增加 CalcTime 以优化采样率。 等式 21 确定了 CalcTime 应设置为何值：

**Equation 21**

$$CalcTime \geq \frac{DataClock * 200}{CPUClock}$$

#### 示例

如果 DataClock 设置为 1.5 MHz，CPU 在 6 MHz 下运行，则 CalcTime 应当设置为大于或等于 25。如等式 22 所示：

**Equation 22**

$$CalcTime \geq \frac{DataClock * 200}{CPUClock} = \frac{1.50MHz * 200}{12.0MHz} = 25DataClocks$$

### 时钟和积分器列时钟

数据时钟确定采样率和信号采样窗口。 此时钟必须路由到计数器模块、24-bit PWM 模块和包含积分器列的列时钟的时钟输入。

此参数设置仅设置计数器模块和 PWM 模块的时钟。

Note 必须将积分器开关电容模块的列时钟手动设置为 “ 相同 ” 时钟。 所有三个模块必须使用相同时钟，否则此用户模块工作不正常。

此时钟可以是时钟频率在 125 kHz 到 8 MHz 之间的任意源：

$$SampleRate = \frac{DataClock}{2^{16} + CalcTime}$$

### 数据格式

此选择确定返回结果的格式。 如果选择 “ 有符号 ”，则数据范围介于 -8182 和 8191 之间。 如果选择无符号格式，则数据范围介于 0 和 16,383 之间。

### 参考复用器全局设置

器件编辑器 “ 全局资源 ” 部分中的 “ 参考复用器 ” 选项选择决定了可用输入电压。 参考复用器的选择决定模拟接地以及有关模拟接地的输入电压可用范围。 下表显示的是 Vdd 为 5V 和 3.3V: 时的有效范围。

Table 4. 每个参考复用器的输入电压范围

| 参考复用器设置 | Vdd = 5 伏特 | Vdd = 3.3 伏特 |
|---|---|---|
| (Vdd/2) ± 带隙 | $1.2 < V_{in} < 3.8$ | $0.35 < V_{in} < 2.95$ |
| (Vdd/2) ± (Vdd/2) | $0 < V_{in} < 5$ | $0 < V_{in} < 3.3$ |
| 带隙 ± 带隙 | $0 < V_{in} < 2.6$ | $0 < V_{in} < 2.6$ |
| (1.6* 带隙 ) ± (1.6* 带隙 ) | $0 < V_{in} < 4.16$ | NA |
| (2* 带隙 ) ± 带隙 | $1.3 < V_{in} < 3.9$ | NA |
| (2* 带隙 ) ± P2[6] | $(2.6 - V_{P2[6]}) < V_{in} < (2.6 + V_{P2[6]})$ | NA |
| P2[4] ± 带隙 | $(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$ | $(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$ |
| P2[4] ± P2[6] | $(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$ | $(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$ |

## 中断生成控制

当在 PSoC Designer 中选中 **启用中断生成控制** 复选框时，会有一个附加参数变为可用。 可以在以下菜单下找到此复选框： **项目 > 设置 > 芯片编辑器** . 当外覆层的多个用户模块所共享的中断用于多个外覆层时，中断生成控制非常重要：

#### IntDispatchMode

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不同外覆层中的多个用户模块共享。 选择 “ActiveStatus” 会导致固件在为共享的中断请求提供服务之前测试哪一个外覆层正处于活动状态。 每次请求共享中断时，都会进行此测试。 这会增加延迟，还会产生为共享中断请求提供服务的不确定过程，但是不需要任何 RAM。 选择 “OffsetPreCalc” 参数会导致固件仅在最初加载重叠层时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。

# 应用程序编程接口

提供 API 子程序以初始化、配置、启动采样、停止和读取 ADC 的结果数据。 在所有情况下，模块的 “ 实例名称 ” 会替换在下列进入点中显示的 “ADCINC14” 前缀。 未能使用正确的名称是常见的语法错误原因。

## ADCINC14_Start

**说明：**

对此用户模块执行所有必需的初始化，并设置开关电容 PSoC 模块的功耗水平。

**C 原型：**

```
void ADCINC14_Start (BYTE bPowerSetting)
```

**汇编：**

```
mov   A, ADCINC14_HIGHPOWER
lcall  ADCINC14_Start
```

**参数：**

PowerSetting： 1 个用于指定功耗水平的字节。 在复位和配置之后，分配给 ADCINC14 的模拟 PSoC 模块关闭电源。 下表给出了以 C 语言和汇编语言提供的符号名称及其相关数值：

| 符号名称 | 值 |
|---|---|
| ADCINC14_OFF | 0 |
| ADCINC14_LOWPOWER | 1 |
| ADCINC14_MEDPOWER | 2 |
| ADCINC14_HIGHPOWER | 3 |

功耗水平会影响模拟性能。 正确的功耗设置与数据时钟的采样率密切相关，必须为每个应用场合确定正确的功耗设置。 建议在开始开发时选择满功率。 可以稍后进行测试，以确定可以将功耗设置设置到多低的程度。

**返回值：**

无

**副作用：**

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

## ADCINC14_SetPower

**说明：**

设置开关电容 PSoC 模块的功率水平。

**C 原型：**

```
void ADCINC14_SetPower (BYTE bPowerSetting)
```

**汇编：**

```
mov   A, [bPowerSetting]
lcall  ADCINC14_SetPower
```

**参数：**

PowerSetting: 与用于 “ 启动 ” API 子程序的 PowerSetting 参数相同。 允许在运行 ADC 时更改功耗水平。

**返回值：**

无

**副作用：**

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

## ADCINC14_Stop

**说明：**

将开关电容积分器模块的功耗水平设置为 “0”。 当未使用 ADCINC14 且您想要省电时，进行此设置。 此子程序关闭模拟开关电容模块的电源，禁用数字模块。 要实现最低功耗水平，还应当从数字模块中删除时钟。

**C 原型：**

```
void  ADCINC14_Stop()
```

**汇编：**

```
lcall  ADCINC14_Stop
```

**参数：**

无

**返回值：**

无

**副作用：**

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

## ADCINC14_GetSamples

**说明：**

初始化并启动 ADC 算法以收集指定的采样数。 请记住通过调用以下位置中的 M8C_EnableGInt 宏调用以启用全局中断： *M8C.inc* 或 *M8C.h*.

**C 原型：**

```
void ADCINC14_GetSamples (BYTE bNumSamples)
```

**汇编：**

```
mov   A, [bNumSamples]
lcall  ADCINC14_GetSamples
```

**参数：**

NumSamples： 8-bit 值，用于设置要检索的采样数。 值 “0” 会导致 ADC 连续运行。

**返回值：**

无

**副作用：**

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

## ADCINC14_StopAD

**说明：**

立即停止 ADC。

**C 原型：**

```
void ADCINC14_StopAD()
```

**汇编：**

```
lcall  ADCINC14_StopAD
```

**参数：**

无

**返回值：**

无

**副作用：**

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

## ADCINC14_fIsDataAvailable 和 ADCINC14_fIsData

**说明：**

若已完成数据转换，而且数据可读，则返回非零值。

**C 原型：**

```
CHAR ADCINC14_fIsDataAvailable()
CHAR ADCINC14_fIsData()
```

**汇编：**

```
lcall  ADCINC14_fIsDataAvailable
```

**参数：**

无

**返回值：**

当数据可用时返回非零值。

**副作用：**

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

## ADCINC14_iGetData

**说明：**

返回上次转换的数据。在获取数据之前应调用 fIsDataAvailable()，以确保数据有效。 必须在下次转换周期完成之前检索数据，否则数据将被覆盖。 如果对此函数的调用正好在积分周期结束时完成，则返回的数据有可能损坏。 因此强烈建议以高于采样率的频率检索数据。 如果无法保证，请在调用此函数之前关闭中断。

**C 原型：**

```
INT ADCINC14_iGetData()
```

**汇编：**

```
lcall  ADCINC14_iGetData
```

**参数：**

无

**返回值：**

返回转换结果。 在汇编程序中，MSB 在 X 中返回，LSB 在累加器中返回。

**副作用：**

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

## ADCINC14_ClearFlag

**说明：**

清除数据可用标志。

**C 原型：**

```
void ADCINC14_ClearFlag()
```

**汇编：**

```
lcall  ADCINC14_ClearFlag
```

**参数：**

无

**返回值：**

无

**副作用：**

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

## ADCINC14_iGetDataClearFlag"

**说明：**

返回上次转换的数据并清除数据可用标志。在获取数据之前应调用 fIsDataAvailable()，以确保数据有效。 必须在下次转换周期完成之前检索数据，否则数据将被覆盖。 如果对此函数的调用正好在积分周期结束时完成，则返回的数据有可能损坏。 因此强烈建议以高于采样率的频率检索数据。如果无法保证，请在调用此函数之前关闭中断。

**C 原型：**

```
INT ADCINC14_iGetDataClearFlag()
```

**汇编：**

```
lcall  ADCINC14_iGetDataClearFlag
```

**参数：**

无

**返回值：**

返回转换结果。 在汇编程序中，MSB 在 X 中返回，LSB 在累加器中返回。

**副作用：**

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器也会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。 当前，仅修改 CUR_PP 页面指针寄存器。

## 固件源代码示例

此示例代码将启动连续转换、轮询数据可用标志以及将转换的字节发送至用户函数。

```
;;; Sample Code for the ADCINC14
;;; Continuously Sample and call a user routine with the converted data
;;; sample.
;;;
;;; NOTE: The User Routine must complete operation within one conversion
;;; cycle in order to retrieve the next converted sample data.
;;;

include "m8c.inc"       ; part specific constants and macros
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

_main:
    M8C_EnableGInt                  ;Enable interrupts

    mov   a, ADCINC14_HIGHPOWER     ;Set Power and Enable A/D
    call  ADCINC14_Start

    mov   a, 00h                    ;Start A/D in continuous sampling mode
    call  ADCINC14_GetSamples

;A/D conversion loop
loop1:

    wait:                           ;Poll until data is complete
    call  ADCINC14_fIsDataAvailable
    jz    wait

    call  ADCINC14_ClearFlag        ;Reset flag
    call  ADCINC14_iGetData         ;Get Data - X=MSB A=LSB
    ;; Place User code here

    jmp   loop1
```

下面是以 C 语言编写的相同项目：

```
//-------------------------------------------------------------------------
//  Sample C Code for the ADCINC14
//  Continuously Sample and call a user function with the data.
//
//-------------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void main(void)
{
INT iData;
    M8C_EnableGInt;                         // Enable global interrupts
    ADCINC14_Start(ADCINC14_HIGHPOWER);     // Turn on Analog section
    ADCINC14_GetSamples(0);                 // Start ADC to read continuously
    for(;;)
    {
      while(ADCINC14_fIsDataAvailable() == 0); // Wait for data to be ready
      iData = ADCINC14_iGetData();             // Get Data
      ADCINC14_ClearFlag();                    // Clear data ready flag
      // Place user code here

    }
}
```

## 配置寄存器

这些寄存器通过初始化和 API 库进行配置。 您不必直接更改或读取这些寄存器。 此部分仅供参考。

ADC 是开关电容 PSoC 模块。 它配置为生成模拟调制器。 要生成调制器，需要将该模块配置为带有参考反馈的积分器，该积分器将输入值转换为数字脉冲流。 输入复用器确定将对哪些信号数字化。

Table 5.  模块 ADC、寄存器： CR0

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

Table 6.  模块 ADC： 寄存器 CR1

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | ACMux、AMux | | | 0 | 0 | 0 | 0 | 0 |

当模块放置在 “A” 型模块中时，使用 ACMux。 字段值取决于连接输入的方式。 当模块放置在 “B” 型模块中时，使用 AMux。 字段值取决于连接输入的方式。

Table 7.  模块 ADC： 寄存器 CR2

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

Table 8.  模块 ADC： 寄存器 CR3

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 1 | 1 | FSW1 | FSW0 | 0 | 0 | PWR | |

FSW0 供 TMR 中断处理程序和各种 API 使用。 如果值为 “0”，则 ADC 成为禁用的积分器。 如果值为 “1”，则 ADC 成为启用的积分器。

PWM24 是用于控制 ADC 积分时间的数字 PsoC 模块。 比较值设置为 $2^{Bits+2}$，周期设置为 CalcTime 加上比较值。

Table 9. 模块 PWM24_MSB： 寄存器函数

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 1 | Compare Type | Interrupt Type | 0 | 0 | 1 |

Compare Type 标志指示捕获比较是设置为 “ 等于或小于 ” 或 “ 小于 ”。 Interrupt Type 标志指示是基于捕获事件还是基于终端条件来触发中断。 Compare Type 和 Interrupt Type 都是在器件编辑器中设置的。

Table 10. 模块 PWM24_ISB： 寄存器函数

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 0 | Compare Type | 0 | 0 | 0 | 1 |

Compare Type 标志指示比较函数是设置为 “ 等于或小于 ” 或 “ 小于 ”。 此参数在器件编辑器中设定。

Table 11. 模块 PWM24_LSB： 寄存器函数

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 0 | Compare Type | 0 | 0 | 0 | 1 |

Compare Type 标志指示比较函数是设置为 “ 等于或小于 ” 或 “ 小于 ”。 此参数在器件编辑器中设定。

Table 12. 模块 PWM24_MSB： 寄存器输入

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 1 | 1 | 时钟 | | | |

Clock 从 16 个源中的一个源选择时钟输入。 此参数在器件编辑器中设定。

Table 13. 模块 PWM24_ISB： 寄存器输入

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 1 | 1 | 时钟 | | | |

Clock 从 16 个源中的一个源选择时钟输入。 此参数在器件编辑器中设定。

Table 14. 模块 PWM24_LSB： 寄存器输入

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 启用 | | | | 时钟 | | | |

Enable 从 16 个源中的一个源选择数据输入，Clock 从 16 个源中的一个源选中时钟输入。 这两个参数都是在器件编辑器中设置的。

Table 15. 模块 PWN24_MSB: 寄存器输出

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 0 | 0 | 0 | Output Enable | Output Sel | |

Output Enable 标志表示启用输出。 Output Sel 标志指示 PWN24 的输出将路由到何处。 这两个参数都是在器件编辑器中设置的。

Table 16. 模块 PWM24_ISB: 寄存器输出

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 17. 模块 PWM24_LSB: 寄存器输出

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 18. 模块 PWM24_MSB: 计数寄存器 DR0

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Count(MSB) | | | | | | | |

计数: PWM24 MSB 下降 PWM。 可以使用 PWM24 API 读取它。

Table 19. 模块 PWM24_ISB: 计数寄存器 DR0

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Count(ISB) | | | | | | | |

计数: PWM24 ISB 下降 PWM。 可以使用 PWM24 API 读取它。

Table 20. 模块 PWM24_LSB: 计数寄存器 DR0

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Count(LSB) | | | | | | | |

计数: PWM24 LSB 下降 PWM。 可以使用 PWM24 API 读取它。

Table 21. 模块 PWM24_MSB: 周期寄存器 DR1

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Period(MSB) | | | | | | | |

Period 用于保存依据启用或终端计数条件加载到计数器寄存器中的周期值的 MSB。 可以用器件编辑器和 PWM24 API 对其进行设置。

Table 22. 模块 PWM24_ISB: 周期寄存器 DR1

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Period(ISB) | | | | | | | |

Period 用于保存依据启用或终端计数条件加载到计数器寄存器中的周期值的 ISB。 可以用器件编辑器和 PWM24 API 对其进行设置。

Table 23. 模块 PWM24_LSB: 周期寄存器 DR1

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Period(LSB) | | | | | | | |

Period 用于保存依据启用或终端计数条件加载到计数器寄存器中的周期值的 LSB。 可以用器件编辑器和 PWM24 API 对其进行设置。

Table 24. 模块 PWM24_MSB: 脉冲宽度寄存器 DR2

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Pulse Width(MSB) | | | | | | | |

PulseWidth 保存用于生成比较事件的脉冲宽度值的 MSB。 可以用器件编辑器和 PWM24 API 对其进行设置。

Table 25. 模块 PWM24_ISB: 脉冲宽度寄存器 DR2

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Pulse Width(ISB) | | | | | | | |

PulseWidth 保存用于生成比较事件的脉冲宽度值的 ISB。 可以用器件编辑器和 PWM24 API 对其进行设置。

Table 26. 模块 PWM24_LSB: 脉冲宽度寄存器 DR2

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Pulse Width(LSB) | | | | | | | |

PulseWidth 保存用于生成比较事件的脉冲宽度值的 LSB。 可以用器件编辑器和 PWM24 API 对其进行设置。

Table 27. 模块 PWM24_MSB: 控制寄存器 CR0

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Start/ Stop(0) |

Start/Stop 由 LSB 控制寄存器值控制，设置为零。

Table 28. 模块 PWM24_ISB: 控制寄存器 CR0

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | | | | | | | | Start/ Stop(0) |

Start/Stop 由 LSB 控制寄存器值控制，设置为零。

Table 29. 模块 PWM24_LSB: 控制寄存器 CR0

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | | | | | | | | Start/Stop |

若设置了 Start/Stop，则表示 PWM24 处于启用状态。 可以使用 PWM24 API 修改它

CNT 是配置为计数器的数字 PSoC 模块。 当 DR0 中的值倒数到终端计数时，将调用中断以降低软件计数器和 CNT 从 DR1 重新加载的较高值。 数据通过 DR2 输出。

Table 30. 模块 CT： 寄存器函数

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

Table 31. 模块 CT： 寄存器输入

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Data | | | | 时钟 | | | |

Data 用于选择放置了 ADC 模块的列比较器。 Clock 从 16 个源中的一个源选择时钟输入，Clock 是在器件编辑器中设置的。

Table 32. 模块 CT： 寄存器输出

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 33. 模块 CT： 寄存器 DR0

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 计数器值 | | | | | | | |

Table 34. 模块 CT： 寄存器 DR1

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Table 35. 模块 CT： 寄存器 DR2

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | Data Out | | | | | | | |

Data Out 由 API 用于获取计数器值。

Table 36. 模块 CT： 寄存器 CR0

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 启用 |

如果设置了 Enable，则 CNT 处于启用状态。 它由 ADCINC14 API 修改和控制。

Table 37. 寄存器 INT_MSK1

| 位 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 值 | | | | | | | | |

此处用于设置对应于 TMR 模块和 CNT 模块的掩码位，以启用其各自的中断。 实际掩码值取决于各个模块的放置位置。

## 版本历史记录

| 版本 | 创作者 | 说明 |
|---|---|---|
| 1.4 | DHA | 增加了 DRC 以检查是否出现下列情况：<br>1. 数字和模拟资源的源时钟不同。<br>2. ADC 时钟大于 CPU 时钟。<br><br>添加了当两个 ADCINC14 或 ADCINCVR 模块添加到项目中时的 DRC 警告。 |

**Note** PSoC Designer 5.1 在所有的用户模块数据表中提供版本历史记录。 本数据表详细介绍了当前和先前用户模块版本之间的区别。