8 位电压输出 DAC 数据手册 DAC8 V 2.2

Copyright © 2012 Cypress Semiconductor Corporation. All Rights Reserved.

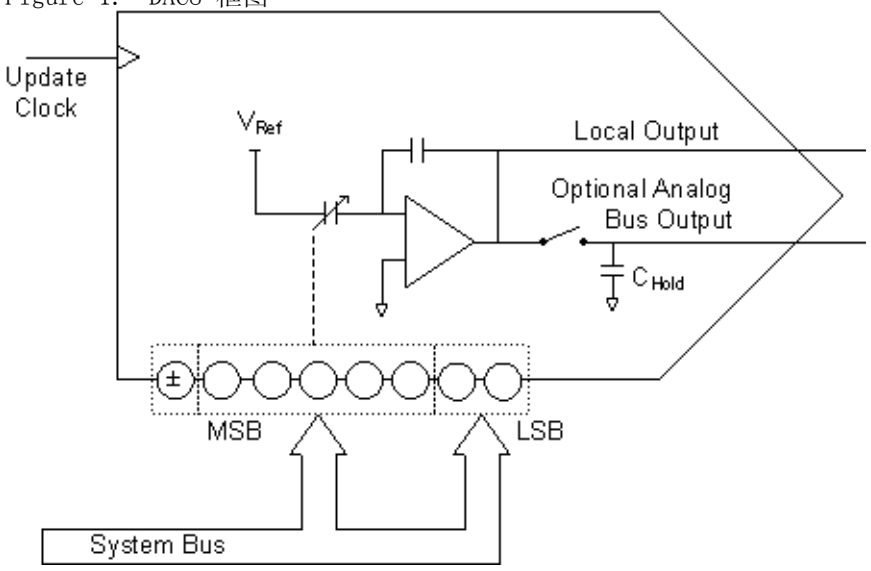
资源	PSoC® 模块			API 存储器 （字节）		引脚 （每个外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52						
	0	0	2	175	0	1

功能和概述

- 8 位分辨率
- 电压输出
- 二进制补码、偏移二进制和符号 / 大小输入数据格式
- 模拟总线和外部输出的采样和保持
- 更新速率达到 125 ksps

DAC8 用户模块将数字代码转换为输出电压。 DAC8 用户模块以高达每秒 125k 次采样的更新速率将数字代码转换为输出电压。 应用程序编程接口 （API） 支持偏移二进制、二进制补码以及寄存器图像数据格式。 偏移补偿用于将错误减到最少。

Figure 1. DAC8 框图

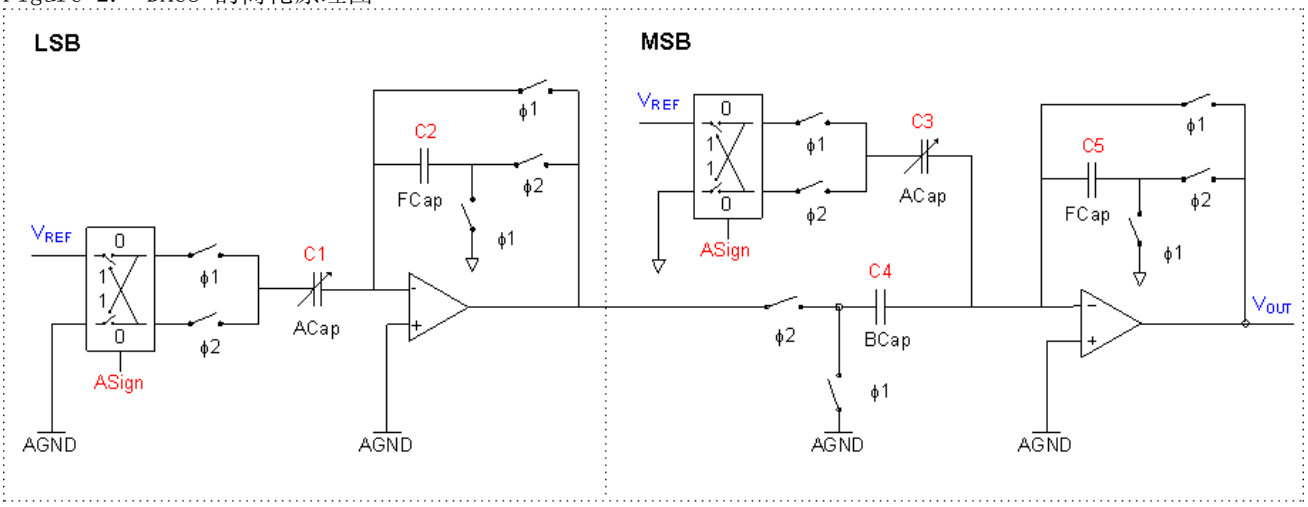


功能描述

DAC8 用户模块将数字代码转换为模拟输出电压。 数字代码使用从 -127 到 +127 的二进制补码形式的数字表示。 或者，输入代码也可使用偏移二进制形式以 0 到 254 范围内的数字表示。 输出电压范围根据针对系统级参数 RefMux 所选择的值，可以有多个选择。

DAC8 用户模块可以映射到任意两个相邻的开关式电容 PSoC 模拟模块。 这两个模块名为 LSB 和 MSB。 LSB 模块或段通过 MSB 模块的 “BCap” 电容 C_4 与 MSB 段连接。 在内部进行操作时基于符号与大小格式。 5 个最高有效大小位设置了 C_3 的值，以下简化原理图显示了二进制加权电容的阵列。 两个最低有效大小位设置了 C_1 的值。 C_3 的值取自 0 到 31 个单位范围内，C_1 的值取自电容单位集合 {0，8，16，24}。可由 ASign 位反相的参考电压在每段分别由大小电容 C_1 和 C_3 与反馈电容 C_2 和 C_5 之间的比率缩放。 每个均拥有 32 个单位的标称电容。 LSB 段的输出由耦合电容 C_4 与反馈电容 C_5 之间的比率进一步缩放。

Figure 2. DAC8 的简化原理图



硬件会在每个更新周期执行偏移补偿。 由 Φ_1 和 Φ_2 控制的开关会在 Φ_1 期间将运算放大器配置为单位增益跟随器。 在此配置中，偏移电压显示在求和节点上，为各种 ACaps、BCaps 和 FCaps 充电。 在 Φ_2 中重新配置时，电路会将这些电容上的偏移充电反相，从而有效抵消了偏移电压。

在每个更新周期中，V_{out} 在运算放大器偏移电压 （于 Φ_1 期间设定）与所需电压 （于 Φ_2 期间设定）之间转换；直接结果是出现偏移补偿。 降低高精度代价的一种方法是使用与输出总线相关联的采样与保持电路。 V_{out} 在 Φ_2 后半阶段为负载和保持电容 （DAC8 框图中的 C_{Hold}）充电。 在该周期结束时，C_{Hold} 将会与运算放大器输出分离。 每个模拟输出总线由带有适当较高输入阻抗的模拟输出缓冲区提供支持。

综合缩放参考结果，输出为：

Equation 1

$$V_{Out} = (V_{REF} \pm AGND)\frac{C_1 C_4}{C_2 C_5} + (V_{REF} \pm AGND)\frac{C_3}{C_5} + AGND$$

如果在器件编辑器中将全局参数 RefMux 配置为 (2 BandGap) ± BandGap，AGND 为 2.6V，参考电压为 1.3V，那么相应的输出为：

Equation 2

$$2.6Volts \pm 1.3Volts\left(\frac{C_1}{2^5 \cdot 2^5} + \frac{C_3}{2^5}\right)$$

公式 2 显示为一个具有 10 个大小位的结果；不过注意 C_1 的值限制为使用 8 作为因子标度 2 个最低有效大小位所获取的数值。 约去 C_1 及其分母中的比例因子后，结果可表示为：

Equation 3

$$2.6\text{Volts} \pm 1.3\text{Volts}\left(\frac{C_1}{2^5 \cdot 2^5} + \frac{C_3}{2^5}\right), \qquad C_1 \in \{0,1,2,3\}$$

示例

上面的公式 2 和 3 显示，8 位 DAC 输入代码分布在 2 个数值和符号位中。 输入代码的 MSB 会用作符号位。 第 2 位到第 7 位用于指定 C_3 的 5 位值，最后，两个 LSB 用于指定 C_1。 输入代码值 -74 的符号为负，C3 值为 18，C1 值为 2。 将这些值代入公式 2 中，将得到：

Equation 4

$$V_{Out} = 2.6\text{Volts} - 1.3\text{Volts}\left(\frac{2}{2^5 \cdot 2^5} + \frac{18}{2^5}\right) = 1.86\text{Volts}$$

计算得到的值是理想值，该值很可能会根据系统噪声和芯片偏移而有所不同。

直流和交流电气特性

下列值表示了预计的性能，它们基于初始特性数据。 除非下表中另有规定，否则 T_A = 25° C，Vdd = 5V。除非另有规定，否则 f_{clock} = 125 kHz，外部 AGND 为 2.50V，外部 V_{Ref} 为 1.23V，REFPWR = HIGH，SCPOWER = ON，PSoC 模块功耗为 HIGH。

Table 1. 5.0V DAC8 直流和交流电气特性， CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52 系列 PSoC 器件

参数	典型值	限制	单位	条件和注释
分辨率	—	8	位	
线性度				
DNL	0.5		LSB	
INL	0.3		LSB	
单调性	是			
增益误差				
包括参考增益误差	3.5		%FSR	
不包括参考增益误差[3]	0.5		%FSR	
V_{OS}，偏移电压	±7.5		mV	
输出噪声	4.5		mV rms	0 到 300 kHz
f_{clock}，模拟列时钟[1]				

参数	典型值	限制	单位	条件和注释
低功耗	8 到 500		kHz	
中功耗	4 到 2000		kHz	
高功耗	4 到 3200		kHz	
工作电流 [2]				
低功耗	305		µA	
中功耗	1130		µA	
高功耗	4315		µA	

下列值表示了预计的性能，它们基于初始特性数据。 除非下表中另有规定，否则 T_A = 25° C，Vdd = 3.3V。除非另有规定，否则 f_{clock} = 125 kHz，外部 AGND 为 1.50V，外部 V_{Ref} 为 0.8V，REFPWR = HIGH，SCPOWER = ON，PSoC 模块功耗为 HIGH。

Table 2. 3.3V DAC8 直流和交流电气特性， CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52PSoC 器件系列

参数	典型值	限制	单位	条件和注释
分辨率	—	8	位	
线性度				
DNL	0.5		LSB	
INL	0.4		LSB	
单调性	是			
增益误差				
包括参考增益误差	2.6		%FSR	
不包括参考增益误差 [3]	0.3		%FSR	
V_{OS}，偏移电压	±7.5		mV	
输出噪声	2.5		mV rms	0 到 300 kHz
f_{clock}，模拟列时钟 [1]				
低功耗	8 到 500		kHz	
中功耗	4 到 2000		kHz	
高功耗	4 到 3200		kHz	
工作电流 [2]				
低功耗	290		µA	
中功耗	1090		µA	
高功耗	4175		µA	

下列值表示预计的性能，且基于初始特性数据。 除非下表中另有规定，否则 T_A = 25° C，Vdd = 2.7V。除非另有规定，否则 f_{clock} = 125 kHz，外部 AGND 为 1.50V，外部 V_{Ref} 为 0.8V，REFPWR = HIGH，SCPOWER = ON，PSoC 模块功耗为 HIGH。

Table 3. 2.7V DAC8 直流和交流电气特性， CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52 系列 PSoC 器件

参数	典型值	限制	单位	条件和注释
分辨率	—	8	位	
线性度				
DNL	0.5		LSB	
INL	0.4		LSB	
单调性	是			
增益误差				
包括参考增益误差	2.6		%FSR	
不包括参考增益误差 [3]	0.3		%FSR	
V_{OS}，偏移电压	±7.5		mV	
输出噪声	2.5		mV rms	0 到 300 kHz
f_{clock}，模拟列时钟 [1]				
低功耗	8 到 500		kHz	
中功耗	4 到 2000		kHz	
高功耗	4 到 3200		kHz	
工作电流 [2]				
低功耗	290		µA	
中功耗	1090		µA	
高功耗	4175		µA	

电气特性说明

1. 为宽频带噪声中增加指定的范围上限为 3 dB。 下降的下限 < 1 LSB。
2. 由 DAC 选择的模拟列时钟频率比控制更新周期的相位时钟频率快四倍。 请参阅以下的时序讨论内容。
3. 不包括所有模拟模块均通用的参考模块功耗 （请参阅 PSoC 系列数据手册）。
4. 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

DAC8 用户模块会映射到两个名为 LSB 和 MSB 的 PSoC 模块。 LSB 模块的输出将会馈送到 MSB 模块的输入，因此，这两个模块始终置于相邻位置。 在 CY8C26/25xxx 器件系列中，MSB 模块仅映射到 “A 型 ” 开

关电容器 PSoC 模块中。 在 CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52 器件系列中，MSB 模块仅映射到 “C 型 ” 开关电容 PSoC 模块中。 这有助于最大程度地减小线性度误差，因为这些类型的模块允许自动归零进程消除连接 LSB 和 MSB 模块的 “BCap” 电容 （上图中的 C_4）上的偏移误差。

此外，在选择放置位置时需要考虑 MSB 和 LSB 时钟必须来自相同的源。 如果将其放置在模拟阵列的同一列，则这种情况将会自动发生。 如果将其放置在两个不同的列中，则必须将两个列复用器设置为同源。

参数和资源

要创建 DAC8 实例，请选择器件编辑器中的用户模块，如果需要可重新命名，并将其放置到该器件中。 如果信号在片外驱动并与列时钟资源上的其他用户模块相互依存，放置时需注意模拟列输出总线是否可用。 放置后，该用户模块将显示其参数。

DataFormat （数据格式）

DAC8 用户模块 API 可处理以下三种不同的数据格式：偏移二进制、二进制补码及符号与大小。 API 章节 （参见下文）的 WriteBlind 入口点部分对这些规范及每个规范相关值的范围进行了描述。

AnalogBus （模拟总线）

DAC 模块将其输出传播到相邻的模拟 PSoC 模块。 选择其中一个模拟总线选项，将 DAC 输出通过一个模拟输出缓冲区与外界连接。 在特定列中选择该总线，为阵列顶部的 PSoC 模块提供额外的本地连接。 开关电容 PSoC 模块采用了一个采样和保持电路，可在 Φ_2 的后半阶段对 DAC 输出进行采样。 这样可以将外部输出与自动归零操作中产生的电压摆幅分离。

ClockPhase （时钟相位）

此参数决定由列时钟分频器生成的相位时钟 （Φ_1 和 Φ_2）的角色，列时钟分频器将在接下来的时钟与时序各章节中讨论。 选择*正常*时，Φ_1 期间将发生自动归零循环，DAC 的输出在 Φ_2 期间有效。 将 ClockPhase 设置为*交换*时，情况则相反。 当 DAC 连接到另一个在 Φ_1 期间对输入进行采样的外设时，您会用得上这一设置。

Note 将 ClockPhase 设置为 “ 交换 ” 时，将禁用模拟输出总线的采样和保持功能。 如果将 AnalogBus 参数设置为 “ 启用 ”，总线输出将反映本地 PSoC 模块输出的情况，交替显示 Φ_1 期间的 AGND （加偏移电压）和 Φ_2 期间的所需输出。

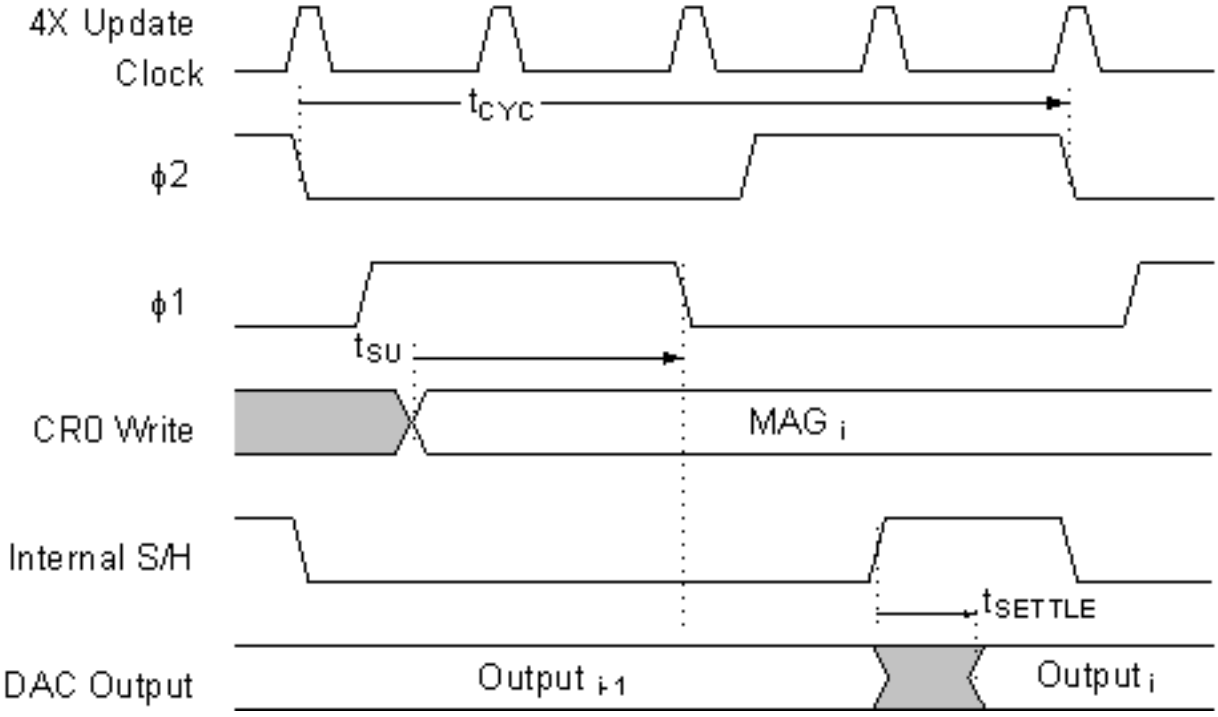
模拟列时钟

不论 DAC 是否获得指令，指示其通过调用相应的 WriteBlind 和 WriteStall API 函数 “ 写入 ” 新值，其都会持续更新输出。 模拟列时钟复用器会选择用于生成相位时钟 Φ_1 和 Φ_2 的源时钟，相位时钟可控制此更新操作。 相位时钟发生器将列时钟四分频，生成 Φ_1 和 Φ_2，因此列时钟频率比实际的模拟输出更新速率要快四倍。 两个复用等级为包括所有数字模块和系统时钟分频器在内的列时钟提供了不同选择。 上面的 “ 电气特性 ” 章节对列时钟频率的上限和下限进行了说明。

对于正输出电压 (Vout > AGND) 和正常相位， Φ_1 期间参考电压会存储在 A Cap 中，如上面的简化原理图所示。 为了对 A Cap 完全充电，对其值所作的任何更改都必须符合 Φ_1 下降沿的建立时间 t_{SU}。 此建立时间 （如下图所示）取决于参考功耗水平。 虽然建立时间未特性表征化，但硬件停止机理可用来保证符合建立时间的要求。 如果由于建立时间未满足要求而造成 A Cap 未完全充电，输出

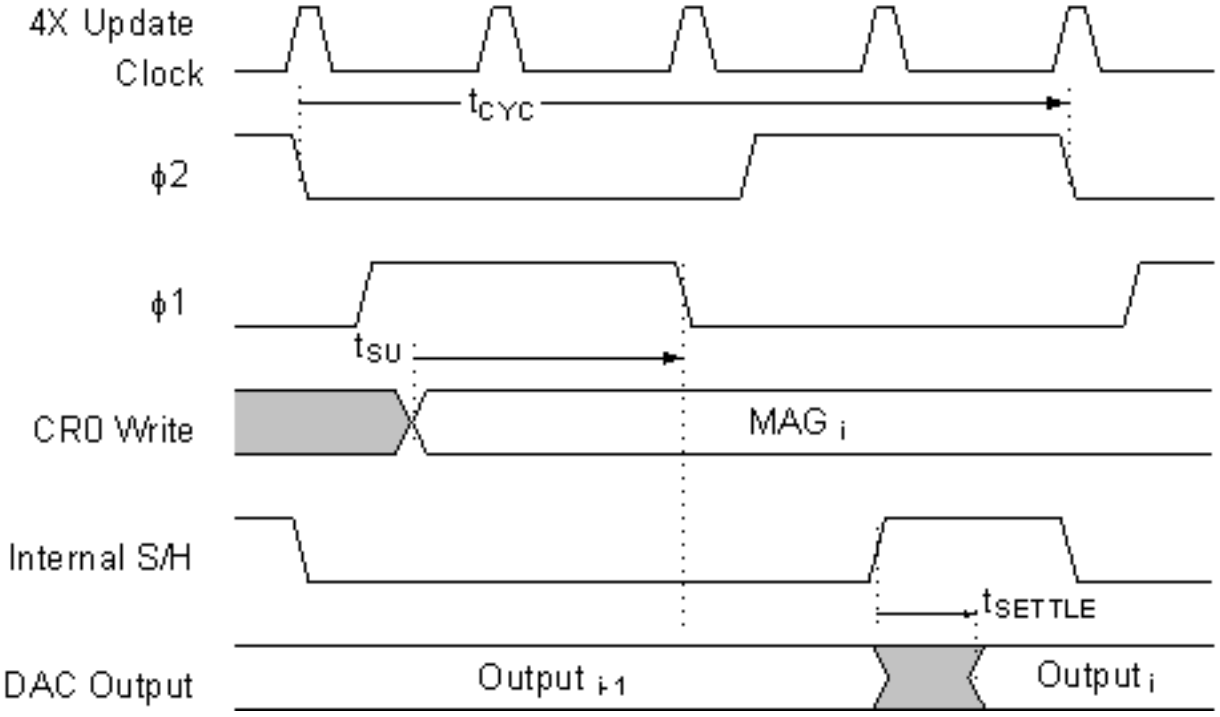
将会出错，直到到达整个 f1 周期的下一个相位时钟周期时才会得到更正。 类似的，f2 下降沿的建立时间会控制负输出电压 (Vout < AGND) 的行为。

Figure 3. V_{OUT} > AGND 的更新时序



对于一大类应用程序，允许出现瞬时 （一个更新周期）偏差。 其他应用程序可能有更为严格的要求。 可以利用硬件同步来控制对 A Cap 值进行更改的寄存器写入时序。 此操作在 WriteStall API 中由入口点直接支持。 调用时，基础硬件对向 PSoC 模块寄存器的写入操作加以识别，并冻结 CPU 时钟，保持写入完成状态，直至出现 Φ_1 的上升沿为止。 ASY_CR 寄存器会进行相应控制。

Figure 4. 硬件同步与 CPU 时钟停止



CPU 停止期间，所有模拟与数字 PSoC 模块功能正常运行。 延迟期间，用于指示写入 DAC 的 CR0 寄存器的 MOV 指令仅暂停，所有中断会进入或保持待处理状态。 CPU 周期的数量在同类停止期间的相应时间内丢失，相应周期可以使用以下关系进行计算。

Equation 5

$$CPU\,Cycles \le \frac{F_{CPU}}{F_{\phi_1}} = \frac{4 \times F_{CPU}}{F_{ColClock}}$$

要将停止期间丢失的 CPU 周期的数量减至最少，无疑应以实际最高频率运行列时钟。 由于额外的输出周期只重复之前的输出周期，所以，相比对输出电压进行实际更改所需的频率，这一频率要高得多。 较快的列时钟也会最大程度地减少从输出函数调用到输出更改时的延迟。

不过，对列时钟频率存在实际的限制。 由于 DAC 必须从 AGND 转化为每个相位时钟周期的输出电压，因此，列时钟的频率被限定为允许设定输出的大小。 当使用模拟输出总线的采样和保持功能时，运算放大器输出将在 Φ_2 的后半阶段的采样窗口中驱动总线。 如果列时钟过快，导致运算放大器仍在转化并设定输出电压，则会观察到运算放大器输出信号的噪声。 极端情况下，在采样窗口关闭前该输出仅部分转化为最终输出电压。 这时可观察到输出中出现的严重非线性增益压缩，在输出范围全量程末端最为明显。 模拟列时钟可防止此现象的发生，其上限在上面的 “ 电气特性 ” 表中给出。

应用程序编程接口

提供的应用程序编程接口 (API) 子程序作为用户模块的一部分，允许设计人员能够采用更高级的方式处理模块。 本节指定每个函数的接口，以及 “ 引用 ” 文件所提供的相关常量。

Note

在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可能通过调用 API 函数发生更改。 如果在调用后需要 A 和 X 的值，则调用函数负责在调用前保留 A 和 X 的值。 选择此 “ 寄存器易失 ” 策略是

为了提高效率，自 PSoC Designer 1.0 版起已强制使用此策略。 C 编译器自动遵循此要求。 汇编语言程序员也必须确保其代码遵守这一策略。 虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来也会如此。

对于大型储存器模块驱动， 保存 CUR_PP、IDX_PP、MVR_PP 以及 MVW_PP 寄存器中的所有值也是调用程序的职责。 尽管部分寄存器现在可能不可修改，但是无法保证在将来的版本中也会如此。

所提供的入口点用来初始化 DAC8 用户模块、写入更新值，以及禁用用户模块。

DAC8_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  DAC8_Start(BYTE bPowerSetting)
```

汇编程序：

```
mov   A, bPowerSetting
lcall  DAC8_Start
```

参数：

bPowerSetting： 1 个用于指定功耗水平的字节。 在复位和配置之后，会关闭分配给 DAC 模块的 PSoC 模块的电源。 下表给出了在 C 语言程序和汇编语言程序中提供的符号名称及其相关数值。

符号名	值
DAC8_OFF	0
DAC8_LOWPOWER	1
DAC8_MEDPOWER	2
DAC8_FULLPOWER	3

返回值：

None

副作用：

DAC 输出将被驱动。 默认情况下，初始值为 AGND。 如果通电时要求其他输出值，在调用 “ 启动 ” 前先调用一个写入例程。 此函数可能会更改 A 和 X 寄存器。

DAC8_SetPower

说明：

设置 DAC 开关电容 PSoC 模块的功耗水平。 可用于打开和关闭模块。

C 原型：

```
void  DAC8_SetPower(BYTE bPowerSetting)
```

汇编程序：

```
mov   A, bPowerSetting
lcall  DAC8_SetPower
```

参数：

bPowerSetting： 与 “DAC8_Start” 入口点使用的 bPowerSetting 参数相同。

返回值：

None

副作用：

DAC 输出将被驱动。 默认情况下，初始值为 AGND。 如果通电时要求其他输出值，在调用 “ 启动 ” 前先调用一个写入例程。 此函数可能会更改 A 和 X 寄存器。

DAC8_WriteBlind

说明：

立即将输出电压更新为指定值。

C 原型：

```
// For OffsetBinary:
void  DAC8_WriteBlind(BYTE bOutputValue)
// For TwosComplement:
void  DAC8_WriteBlind(CHAR cOutputValue)
// For TwoByteSignAndMagnitude:
void  DAC8_WriteBlind2B(BYTE bMSB, BYTE bLSB)
```

汇编程序：

```
; for OffsetBinary and TwosComplement
mov   A, bOutputValue
lcall  DAC8_WriteBlind
; for TwoByteSignAndMagnitude format:
mov   A, bLSB
mov   X, bMSB
lcall  DAC8_WriteBlind
; for OffsetBinary and TwosComplement
mov   A, cOutputValue
lcall  DAC8_WriteBlind
; for TwoByteSignAndMagnitude format:
mov   A, bLSB
mov   X, bMSB
lcall  DAC8_WriteBlind
```

参数：

cOutputValue： 用于指定输出电压的一个字节。 允许的数值范围对应于 DataFormat （数据格式）的选择数值，如下表所示。

数据格式	最小值	最大值
偏移二进制 (OffsetBinary)	0	254
二进制补码 (TwosComplement)	-127	127
两个字节符号与大小 (TwoByteSignAndMagnitude)	3F18h	1F38h

偏移二进制值为正数，用 0 代表最低输出电压，254 代表最高输出电压。 二进制补码为 M8C 处理器的原始带符号格式。 在两个字节符号与大小 (TwoByteSignAndMagnitude) 格式中，高字节的表示形式为 $00smmmmm_2$，低字节的表示形式为 $00tmm000_2$，其中 “s” 为标记，“t” 为反相标记，“m” 代表大小位；对于正数，s=0，t=1。

返回值：

None

副作用：

输出时可能出现短时脉冲的原因将在本用户模块的 “ 时序 ” 章节中讨论。 此函数可能会更改 A 和 X 寄存器。

注意： 当您选择 OFFSET_BINARY 时，超出范围的输入值将自动转换成 API 内的二进制补码数据。 这表明，超过最大偏移二进制容许值的超范围值将转换为一个较小的正输出值 （接近 Agnd）。

DAC8_WriteStall

说明：

Φ_1 开始前可能会停止微处理器，然后更新输出电压到指定值。 请注意，API 假设已禁用中断，或者最大中断延迟小于 ACLKi （请参见 “ 快速更新时钟 ” 图的 “ 强制同步 ”）。

C 原型：

```
// For OffsetBinary:
void  DAC8_WriteStall(BYTE bOutputValue)
// For TwosComplement:
void  DAC8_WriteStall(CHAR cOutputValue)
// For TwoByteSignAndMagnitude:
void  DAC8_WriteStall2B(BYTE bMSB, BYTE bLSB)
```

汇编：

```
; for OffsetBinary and TwosComplement
mov   A, cOutputValue
lcall  DAC8_WriteStall
; for TwoByteSignAndMagnitude format:
mov   A, bLSB
mov   X, bMSB
lcall  DAC8_WriteStall
```

参数：

cOutputValue： 与 WriteBlind 入口点所述参数的格式和值范围相同。

bMSB 和 bLSB： 与 WriteBlind 入口点所述参数的格式和值范围相同。

返回值：

None

副作用：

如果 ACLKi 处于非活动状态（“i” 是模拟 PSoC 模块映射到的列），微处理器的 CPU 时钟已禁用，直到 Φ_2 变为非活动状态，约需四分之三个更新周期 （加上两个 CPU 时钟）。 请注意，停止间隔期间不会识别中断。 此函数可能会更改 A 和 X 寄存器。

DAC8_Stop

说明：

断开用户模块的电源。

C 原型：

```
void  DAC8_Stop(void)
```

汇编：

```
lcall  DAC8_Stop
```

参数：

None

返回值：

None

副作用：

将不会驱动输出。 此函数可能会更改 A 和 X 寄存器。

固件源代码示例

此示例汇编代码创建了一个周期性下降的锯齿波。

```
;-------------------------------------------------------------------------
; This sample code for the DAC8 user module generates a periodic signal that
; ramps down
;-------------------------------------------------------------------------

include "m8c.inc"            ; part specific constants and macros
include "memory.inc"         ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"        ; PSoC API definitions for all User Modules

export _main

DAC_MAX: equ 254             ; This is the maximum DAC value

area bss (RAM, REL, CON)
    bDACValue: blk 1         ; Variable to hold the DAC value


area text (ROM, REL, CON)
_main:
 mov A, DAC8_HIGHPOWER    ; Start DAC with HIGH power setting
    call DAC8_Start

Init:
    mov [bDACValue], DAC_MAX ; Initialize DAC value to hold the maximum
    mov X, 0xFF              ; Initialize X register to hold 0xFF

RampDown:
    mov A, [bDACValue]       ; Move DAC value into A register
    call DAC8_WriteStall     ; Write the value in A to the DAC

Delay:
```

```
    dec X                   ; Decrement X register
    jnz Delay               ; Keep delaying if it hasn't reached zero yet

 dec [bDACValue]          ; Decrement DAC value variable
    jnz RampDown            ; If it is not zero, keep ramping down
    jmp Init                ; If it is zero, restart the ramp down
```

以下 C 语言示例代码与之前的汇编示例代码功能相同：

```
//-------------------------------------------------------------------------
// This C sample code for the DAC8 user module creates a periodic signal
// that ramps down.
//-------------------------------------------------------------------------

#include <m8c.h>                // part specific constants and macros
#include "PSoCAPI.h"            // PSoC API definitions for all User Modules


#define DAC_MAX (254)           // Define max DAC value as 254

unsigned char bDACValue = 0;    // Variable for the DAC value
unsigned char i;                // Variable for an index

void main(void)
{
    DAC8_Start(DAC8_HIGHPOWER); // Start DAC8 in HIGH power mode

 while(1)                    // Repeat forever
    {
        if(bDACValue == 0)
     {
           // Reset DAC value to the max if it reached zero
            bDACValue = DAC_MAX;
     }
        // Write value to DAC and decrement
        DAC8_WriteStall(bDACValue--);
     // Delay loop
     for(i = 0xFF; i != 0; i--);
 }
}
```

配置寄存器

API 为 DAC 提供了完整的接口。 直接写入到配置寄存器是更新输出的另一种方式。 不论采用哪种方法，都存在时序注意事项，必须了解这些情况以防止出现输出短时脉冲。 下面的寄存器用于 DAC8 开关电容 DAC 模块。

Table 4. 模块 LSB：寄存器 CR0

位	7	6	5	4	3	2	1	0
值	1	0	符号	大小		0	0	0

符号用 “1” 表示正值 （从 AGND 到最高为 RefHi），用 “0” 表示负值 （从 RefLow 到最高为 AGND）。默认值为 “1”。 请注意，这与 MSB 模块中使用的含义相反。 使用 API 中的一个写入函数可更改符号值。 大小是一个 5 位的值 (0..31)，默认值是 “0”。 使用 API 中的一个写入函数可更改大小值。

Table 5. 模块 LSB： 寄存器 CR1

开关电容类型 A								
位	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
值	0	1	0	0	0	0	0	0
开关电容类型 B								
位	7	6	5	4	3	2	1	0
值	1	0	0	0	0	0	0	0

Table 6. 模块 LSB： 寄存器 CR2

位	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
值	模拟总线	0	1	0	0	0	0	0

在器件编辑器中进行配置时可启用或禁用模拟总线 (AnalogBus)。

Table 7. 模块 LSB： 寄存器 CR3

开关电容类型 A								
位	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
值	0	0	1	1	0	0	功耗	
开关电容类型 B								
位	7	6	5	4	3	2	1	0
值	0	0	1	1	1	0	功耗	

电源： 默认值为 “ 关 ”。 使用 API 中的 “ 启动 ”(Start) 调用可设置该值。

Table 8. 模块 MSB： 寄存器 CR0

位	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
值	1	0	符号	大小				

“ 符号 ”(Sign) 由 API 写入函数设置。 “ 大小 ”(Magnitude) 可使用 API 中的一个写入函数来更改。

Table 9. 模块 MSB： 寄存器 CR1

位	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
值	0	1	0	0	0	0	0	1

Table 10. 模块 MSB: 寄存器 CR2

位	7	6	5	4	3	2	1	0
值	模拟总线	0	1	0	0	0	0	0

在器件编辑器中进行配置时可启用或禁用模拟总线 (AnalogBus)。

Table 11. 模块 MSB: 寄存器 CR3

位	7	6	5	4	3	2	1	0
值	0	0	1	1	BMux		功耗	

配置 BMux，以便从 LSB PSoC 模块中选择连接。 电源: 0 = 关，1 = 低，2 = 中，3 = 满。 默认值为 “ 关 ”。 使用 API 中的 “ 启动 ”(Start) 调用可设置该值。

版本历史记录

版本	创作者	说明
2.2	DHA	添加了版本历史

Note PSoC Designer 5.1 在所有用户模块数据手册中都引入了 “ 版本历史 ”。 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。

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.