8-Bit 电压输出乘法 DAC 数据表 MDAC8 V 2.2

资源	PSoC® 模块			API 内存 （字节）		引脚 （每个外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C29/27/26/25/24/22xxx、CY8C23x33、CY8CLED04/08/16、CY8CLED0xD、CY8CLED0xG、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C28x43、 CY8C28x52						
	0	0	2	255	0	1

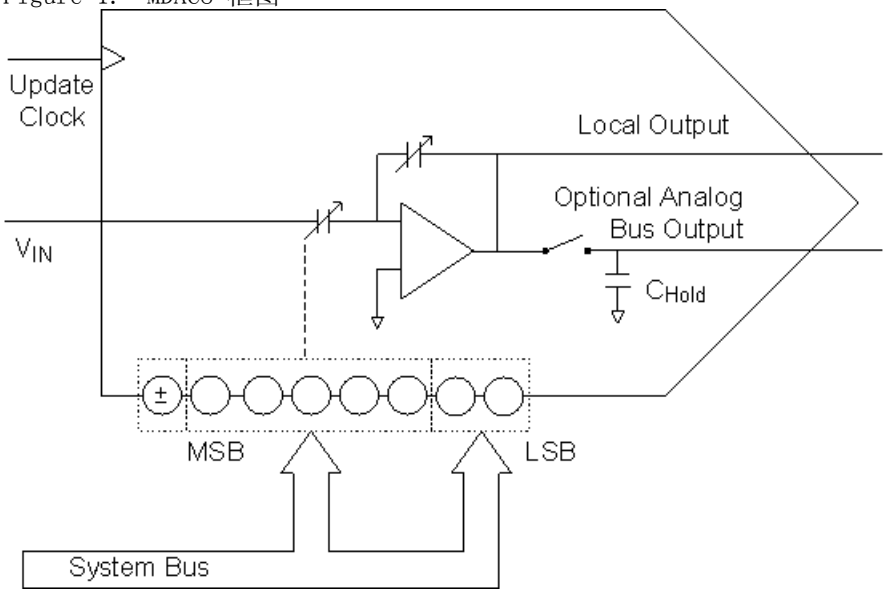
如需一个或多个使用此用户模块的完全配置的功能性示例工程，请转到 www.cypress.com/psocexampleprojects.

特性与概述

- 8-bit 分辨率
- 电压输出
- 四象限乘法
- 2 的补码、偏移二进制和正负 / 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

MDAC8 是使用数字代码标度输入电压的 8-bit、四象限乘法 DAC。 MDAC8 以每秒 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、2 的补码和正负及大小数据格式。 偏移补偿最大程度地减小了转换误差。

Figure 1. MDAC8 框图

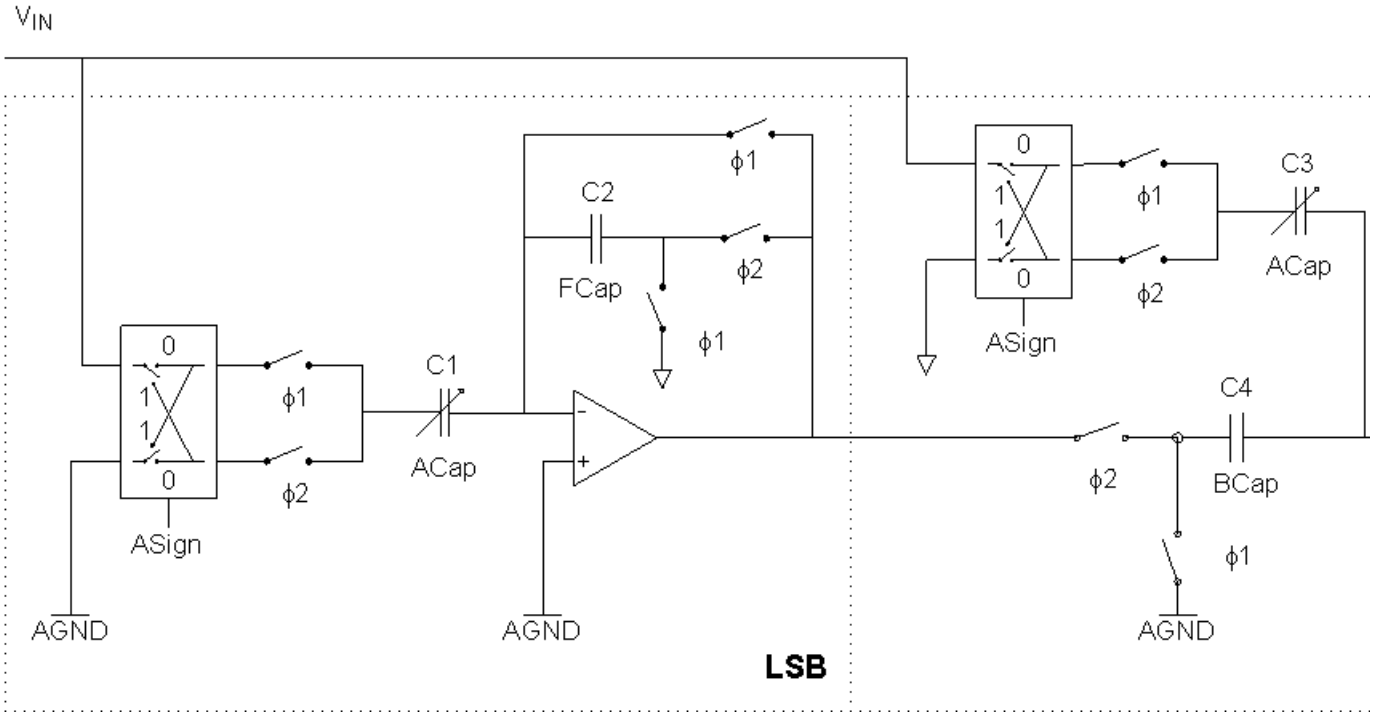


功能说明

MDAC8 用户模块使用数字代码乘以模拟输入电压。 数字代码使用 2 的补码或 -127 到 +127 范围内的正负及大小形式的数字表示。 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。 输入和输出电压相对于 AGND，可通过系统级参数 RefMux 中的值选择 AGND。 输入电压可乘以 1 或者 2 ，具体取决于为 SetOutputRange API 函数选定的值。

MDAC8 用户模块可映射到任意两个模拟 PSoC 模块。 这两个模块名为 LSB 和 MSB。 LSB 模块或 LSB 段通过 “BCap” 电容 C_4 连接到 MSB 段。 在内部，操作基于正负及大小格式。 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 假设为 0 到 31 个单位范围内的值，C_1 假设为电容单位集合 {0，8，16，24} 中的值。 可由 ASign 位反相的输入电压根据大小电容 C_1 和 C_3 分别与反馈电容 C_2 和 C_5 的比率在每个段标度。 每个输入电压均假设为 32 或 16 个单位的值（如果是 32 个单位，则将输入电压增益 1 倍，如果是 16 个单位，则将输入电压增益 2 倍）。 LSB 段的输出通过耦合电容器 C_4 与反馈电容器 C_5 的比率进一步进行标度。 SetOutputRange API 函数将会更改 C_2 和 C_5 的值。

Figure 2. MDAC8 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 过程中将运算放大器配置为单位增益跟随器。 在这种配置中，偏移电压出现在求和节点中，为 ACaps、BCaps 和 FCaps 各电容器充电。 在 Φ_2 中重新配置时，电路会将这些电容器上的偏移充电反相，从而有效抵消了偏移电压。

在每个更新周期内，V_{out} 会在 （Φ_1 过程中的）运算放大器偏移电压和 （在 Φ_2 过程中确定的）偏移补偿的直接结果即所需电压之间倾斜。 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。 在 Φ_2 的后半部分，V_{out} 将对加载和保持电容器 （MDAC8 框图中的 C_{Hold}）进行充电。 在该周期结束时，C_{Hold} 将会与运算放大器输出隔离。 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作用。

输出可由标度输入表示如下。

Equation 1

$$V_{Out} = (V_{IN}-AGND)\frac{C_1C_4}{C_2C_5} + (V_{IN}-AGND)\frac{C_3}{C_5} + AGND$$

当全局参数 RefMux 在器件编辑器中配置为 (2*BandGap) ± BandGap 时，AGND 为 2.6 伏。 设置 $C_F = C_2 = C_1$、$C_4 = 1$ 且 $C_F = 2^5$。 则相应的输出如下。

Equation 2

$$V_{Out} = (V_{IN} - 2.6)\frac{C_1}{2^5 2^5} + (V_{IN} - 2.6)\frac{C_3}{2^3} + 2.6$$

Equation 3

$$V_{Out} = V_{IN}\left(\frac{C_1}{2^{10}} + \frac{C_3}{2^5}\right) - 2.6\left(\frac{C_1}{2^{10}} + \frac{C_3}{2^5}\right) + 2.6$$

回顾一下，C_1 限制为使用 8 作为因子 （左移三位）标度两个最低有效大小位获取的数值。 则约去 C_1 及其分母中的比例因子之后，结果可表示如下。

Equation 4

$$V_{Out} = 2.6Volts + V_{IN}\left(\frac{C_1}{2^8} + \frac{C_3}{2^5}\right) - 2.6\left(\frac{C_1}{2^8} + \frac{C_3}{2^5}\right), C_1 \in 0, 1, 2, 3$$

示例

假设外部源提供的输入电压为 1V ，则等式 3 变成如下所示：

Equation 5

$$V_{Out} = 2.6Volts \pm 1.6Volts\left(\frac{C_1}{2^8} + \frac{C_3}{2^5}\right), C_1 \in 0, 1, 2, 3$$

因此，到 MDAC 的输入代码分布在正负号即输入代码的 MSB 内。 2 - 7 位中包含了 C_3 的值。 最后是由输入代码的 2 个 LSB 位表示的 C_1 的值。 因此，如果值为 -105，则输出电压为 1.3V。

Equation 6

$$V_{Out} = 2.6Volts - 1.6Volts\left(\frac{1}{256} + \frac{26}{32}\right) = 1.3V$$

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。

四象限乘法的意思是输入电压和输入代码均可导致输出电压为正或者为负。 请参见以下 3 幅图。

Figure 3. 输入电压与时间

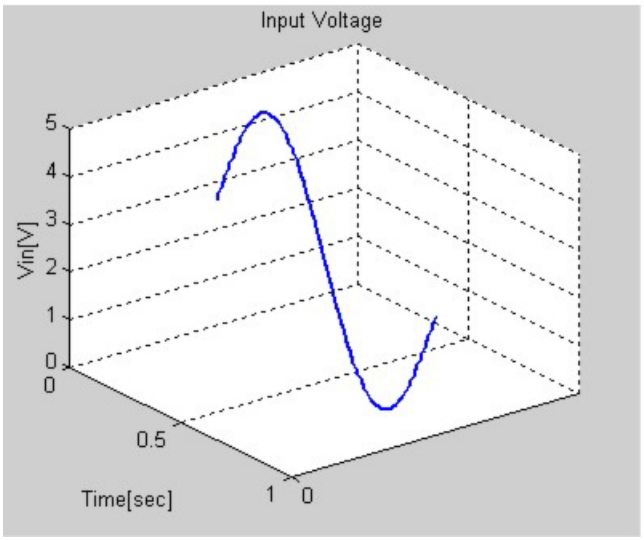


Figure 4. 输出电压与输入代码和时间，FCap=32

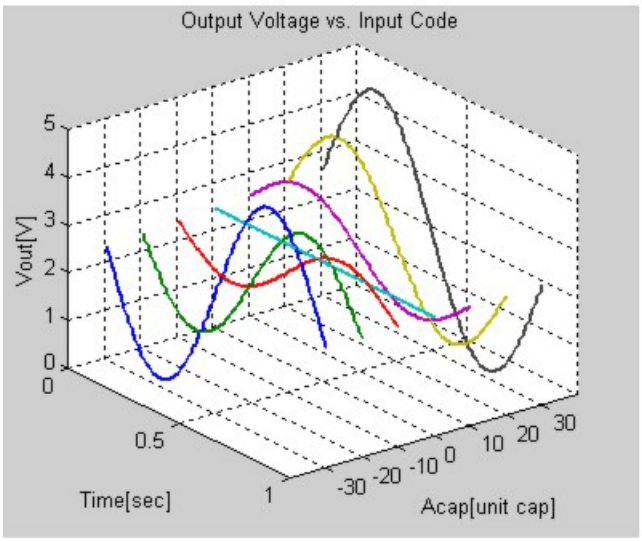
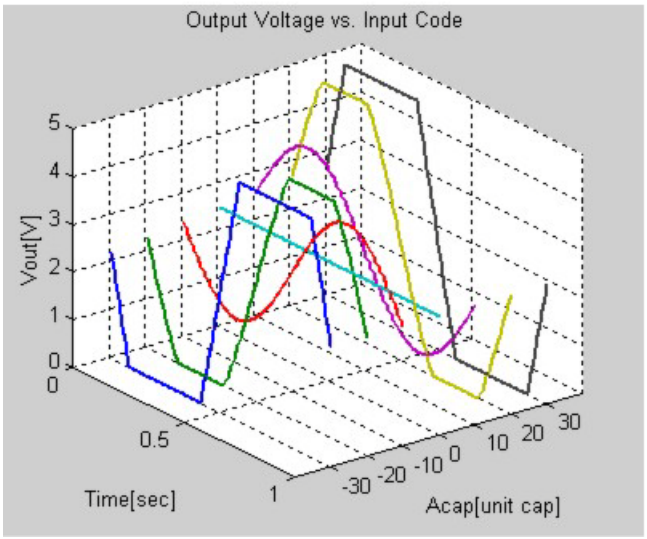


Figure 5. 输出电压与输入代码和时间，FCap=16



输入电压必须减小，以防止输出钳位，如上图所示。

直流和交流电气特性

The following values are indicative of expected performance and based on initial characterization data. 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 MDAC8 直流和交流电气特性

参数	典型值	限值	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	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	

参数	典型值	限值	单位	条件和注释
V_{OS}，偏移电压	±2.1	--	mV	
输出噪声	4.5	--	mV rms	0 到 300 kHz
f_{clock}，内部更新速率 [1]				
低功耗	2 到 125	--	kHz	
中功耗	1 到 500	--	kHz	
高功耗	1 到 800		kHz	
工作电流 [2]				
低功耗	305	--	µA	
中功耗	1130	--	µA	
高功耗	4315	--	µA	

The following values are indicative of expected performance and based on initial characterization data. 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

参数	典型值	限值	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	是	--		
增益误差				
包括参考增益误差	2.6	--	%FSR	

参数	典型值	限值	单位	条件和注释
不包括参考增益误差[3]	0.3	--	%FSR	
V_{OS}，偏移电压	±3.5	--	mV	
输出噪声	2.5	--	mV rms	0 到 300 kHz
f_{clock}，内部更新速率[1]				
低功耗	2 到 125	--	kHz	
中功耗	1 到 500	--	kHz	
高功耗	1 到 800	--	kHz	
工作电流[2]				
低功耗	270	--	µA	
中功耗	1020	--	µA	
高功耗	3900	--	µA	

电气特性注释

1. 对 Φ_1、Φ_2 的限制；对宽频带噪声中增加 3dB 的具体指定。
2. 不包括参考模块功耗，所有模拟模块（请参见 PSoC 系列数据表）均可适用。

除非下表中另有规定，否则保证在所有情况下均限制为 TA = 25C，Vdd = 5V。除非另有规定，否则 fclock = 125 kHz，外部 AGND 为 2.50V，外部 VRef 为 1.23V，REFPWR = HIGH，SCPOWER = ON，PSoC 模块功耗为 HIGH。

Table 3. 5.0V MDAC8 直流和交流电气特性

参数	典型值[1]	限值[2]	单位	条件和注释
分辨率	—	8	位	
线性度				
DNL	0.10	0.25	LSB	
INL	0.15	0.40	LSB	

参数	典型值[1]	限值[2]	单位	条件和注释
单调性	—	½	位	
增益误差	1.0	2.5	%FSR	
V_{OS}，偏移电压[3]	8	43	mV	
输出噪声				
频带限值	0.3	1	mV rms	0 到 10 kHz
宽频带	7	10	mV rms	0 到 300 kHz
f_{clock}，内部更新速率[4]	—	32 到 333	kHz	
V_{in} 频带宽	40	—	kHz	
工作电流[5]				
低功耗	250	—	µA	
中功耗	560	—	µA	
高功耗	1560	2000	µA	

除非下表中另有规定，否则保证在所有情况下均限制为 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 0.80V，REFPWR = HIGH，SCPOWER = ON，PSoC 模块功耗为 HIGH。

Table 4. 3.3V MDAC8 直流和交流电气特性

参数	典型值[1]	限值[2]	单位	条件和注释
分辨率	—	8	位	
线性度				
DNL	0.10	0.20	LSB	
INL	0.20	0.45	LSB	

参数	典型值[1]	限值[2]	单位	条件和注释
单调性	—	½	位	
增益误差	1.0	2.5	%FSR	
V_{OS}，偏移电压[3]	7	31	mV	
输出噪声				
频带限值	0.3	1	mV rms	0 到 10 kHz
宽频带	7	10	mV rms	0 到 300 kHz
f_{clock}，内部更新速率[4]	—	32 到 333	kHz	
V_{in} 频带宽	40	—	kHz	
工作电流[5]				
低功耗	200	—	µA	
中功耗	500	—	µA	
高功耗	1280	1800	µA	

电气特性注释

1. 典型值表示统计平均值加 1σ。
2. 限值通过测试或统计分析进行保证。
3. 2 的补码对外部 AGND 进行零标度偏移，不包括模拟输出缓冲区偏移误差。
4. 对 Φ_1、Φ_2 的限制；对宽频带噪声中增加 3dB 的具体指定。
5. 不包括参考模块功耗，所有模拟模块 （请参见 PSoC 系列数据表）均可适用。

时序

模拟列时钟电路利用一对 2 分电路为负遮盖相位时钟 Φ_1 和 Φ_2 生成时序信号和适当的占空比。 因此，为 MDAC8 选定的时钟源的运行速率必须是所需最大更新速率的 4 倍。 除在内部用于去除输出短时脉冲的采样与保持信号外，还会生成 “ 就绪 ” 信号。 就绪表示可在不违反建立时间要求的情况下写入控制寄存器 0。 请参见以下更新时序图。

未在同一个更新周期内写入两个寄存器并满足写入这两个寄存器的建立时间要求将会导致该周期生成错误的输出值。 如果在 Φ_2 开始之前写入这两个寄存器但不满足建立时间，则输出值介于之前的输出电压和所需输出之间。 如果在 Φ_2 为高时写入一个或两个寄存器，则输出值可能会在之前输出和所需输出确定的电压区间之外。 为尽可能减小中断等非确定性事件所产生的偏差，应首先写入 MSB 寄存器。

对于一大类的应用程序，上述瞬时 （一个更新周期内的）偏差是可以接受的。 对于其他应用程序，要求必须更为严格。 例如低失真波形发生器。 硬件同步是一种确定寄存器更新时序的方法，此方法可用于避免这样的问题，并且受 API 中以单词 “Stall” 结尾的进入点直接支持。

硬件同步可保证对输出值寄存器进行尽早访问。 写入 ASY_CR 寄存器将会触发硬件同步。 如果 ACLKi 处于活动状态，则会立即继续写入 IO 空间；如果不处于活动状态，则 CPU 时钟将通过 IO 写入中途停顿，直到 ACLKi 被置为有效。 即使所需更新速率慢得多，也可通过更高的频率 （最高可达 f_{MAX}）运行更新时钟，这是将停止期间的 CPU 周期损失降到最低的一种方法。 与快速更新时钟的强制同步显示了在标签 “A” 处的建立时间失败，从而导致 CPU 时钟在最差情况下停止。

Figure 6. 更新时序

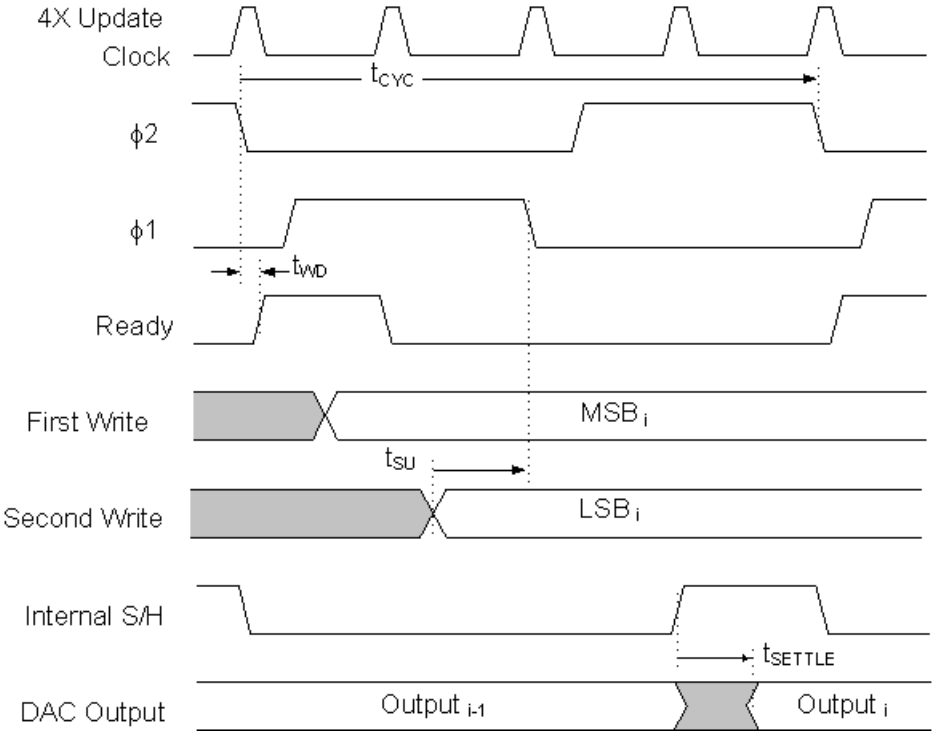
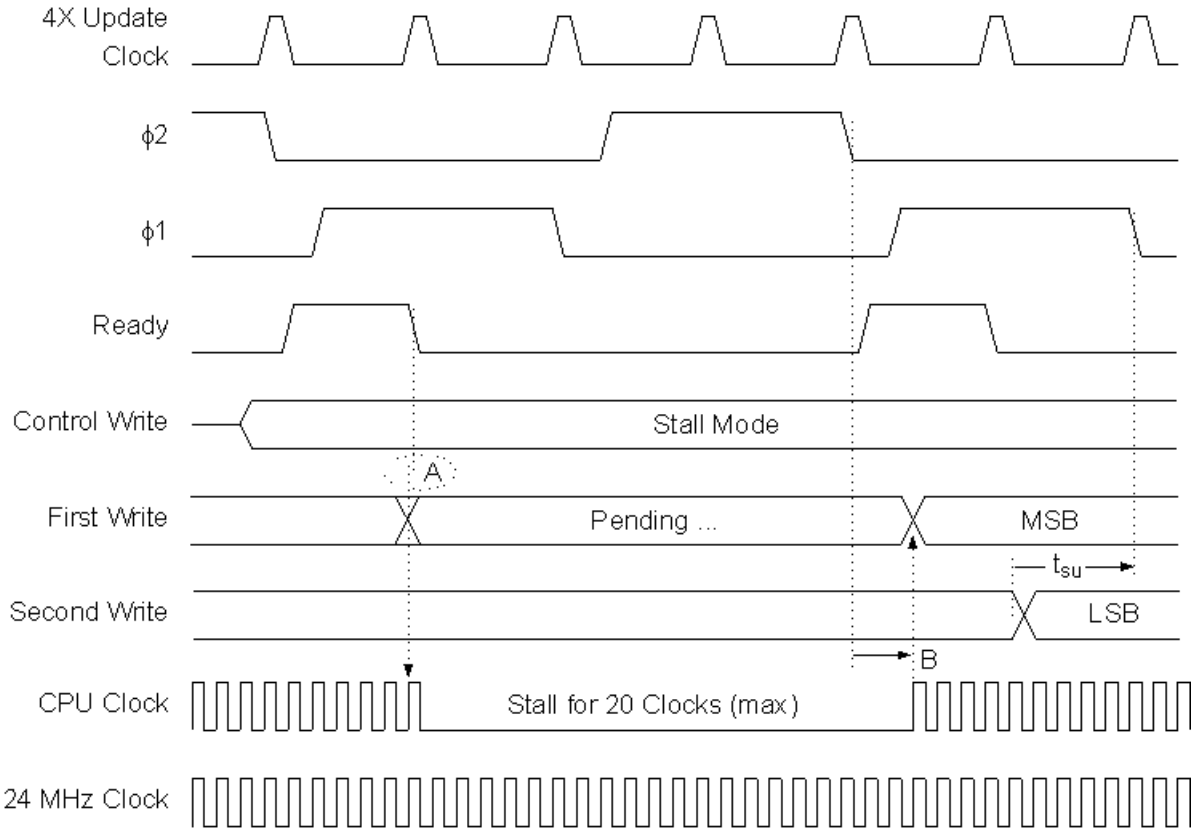


Figure 7. 与快速更新时钟的强制同步



在 CPU 停止期间，所有模拟和数字 PSoC 模块正常运行。 写入 DAC CR0 寄存器的第一个 MOV 指令仅仅处于暂停状态，并且任何中断将会在此期间变为或保持待处理状态。 解除停止且完成停止写入之后，待处理的中断在启用后将会得到处理。 这可能存在问题，也可能不存在问题，具体取决于最大总中断延迟以及 Φ_1 和 Φ_2 的时钟周期。

放置

MDAC8 用户模块将会映射到两个 PSoC 模块，名为 LSB 和 MSB。 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。 在 CY8C27/24/22xxx 器件系列中，MSB 模块仅映射到 “C 型” 开关电容器 PSoC 模块。 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” 电容器 （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。 某些 MDAC8 放置不允许为 MSB 和 LSB 模块选择同一输入源。

Note 为 MSB 和 LSB 选择同一输入电压源。

参数和资源

为使数模转换器工作，可在器件编辑器的 “ 选择用户模块 ” 模式下创建 MDAC8 用户模块的一个实例。 然后可将 MSB 和 LSB 模块映射到模拟阵列中的开关电容器 PSoC 模块。 其他任务包括配置适合衍生 Φ_1 和 Φ_2 的更新时钟资源、指定数据格式、选择 “ 增益 ” 范围、选择输入电压源以及分配与 PSoC 模块相关联的输出总线。

配置更新时钟资源包括三个步骤。 第一，配置模拟列时钟发生器的时钟源。 列时钟发生器通过将输入除以 4 生成 Φ_1 和 Φ_2，因此，时钟源的运行速率必须比所需模拟输出更新速率快 4 倍。 “ 时序 ” 章节讨论了与选择更新时钟频率相关的问题。 时钟源的选择包括 V1 和 V2 分频器以及任一数字 PSoC 模块。 当使用外部源或者由于其他目的必须使用 V1 和 V2 时，所有定时器、计数器和脉冲宽度调制器 （PWM）用户模块都应该为速率发生器进行合理的选择。 Φ_1 和 Φ_2 活动周期间的死区时间与 24 MHz 系统时钟同步发生。 因此，外部时钟源未同步到系统时钟可能会产生无法预知的结果。

第二，通过在器件编辑器中设置时钟复用器将时钟源连接到列时钟发生器。 数字 PSoC 模块输出还必须通过 ACLK0 或 ACLK1 复用器进行连接。 有关其他信息，请参见 *PSoC Designer: 集成开发环境用户指南和 PSoC 数据表* 。

最后，通过选择 “ 标准 ” （默认值）或 “ 交换 ” 选择 MDAC8 用户模块参数 ClockPhase 的值。 这样可将 MDAC8 的输出与另一个 PSoC 模块的输入同步。 开关电容器模拟 PSoC 模块使用 Φ_1 和 Φ_2 获取并传输信号。 由于 MSB 模块的输出仅在 Φ_2 中有效，因此当它连接到在 Φ_1 中对其输入进行取样的另一个用户模块时将会出现问题。 将 ClockPhase 参数设置为 “ 交换 ” 将会互换 MSB 和 LSB 模块内 Φ_1 和 Φ_2 的作用，从而当下游用户模块对其输入进行取样时输出将会有效。 （请注意，在 “ 正常 ” 模式下，输入电压在 Φ_1 中进行取样。）

Note 为 MSB 和 LSB 选择同一输入电压源。

InputMSB

MSB 模块的输入电压源。 选择 REFHI 作为输入电压源将会导致 MDAC8 的作用与 DAC8 相似。 其他输入电压选择需要所选模块中相应用户模块的配置。 某些 MDAC8 放置不允许为 MSB 和 LSB 模块选择同一输入源。

Note 为 MSB 和 LSB 选择同一输入电压源。

InputLSB

LSB 模块的输入电压源。 选择 REFHI 作为输入电压源将会导致 MDAC8 的作用与 DAC8 相似。 其他输入电压选择需要所选模块中相应用户模块的配置。 某些 MDAC8 放置不允许为 MSB 和 LSB 模块选择同一输入源。

Note 为 MSB 和 LSB 选择同一输入电压源。

AnalogBus

MDAC8 输出可通过本地互连的模拟 PSoC 模块阵列网络以及 / 或者通过模拟输出总线进行路由。 将 MDAC8 用户模块 AnalogBus 参数设置为 “ 禁用 ”（默认值）将会限制到本地网络的可能连接集合。 选择 “ 启用 ” 会将可能的连接集合扩展到能够驱动引脚的相关模拟输出缓冲区和不由本地网络提供的某些其他输入复用器。

每个开关电容器 PSoC 模块均采用在 Φ_2 中对总线启用信号进行采样的电路。 这样可消除在自动归零操作过程中出现的电压摆幅。

Note 将 AnalogBus 设置为 “ 启用 ” 并将 ClockPhase 设置为 “ 交换 ” 将会禁用采样和保持功能。 在这种情况下，总线输出将会在 Φ_1 中的 AGND （加上偏移电压）和 Φ_2 中的所需输出之间进行交替，从而映射本地 PSoC 模块输出。

ClockPhase

时钟相位的选择用于将一个模拟 PSoC 模块的输出与另一个模块的输入同步。 开关电容器模拟 PSoC 模块使用两相位时钟（Φ_1、Φ_2）获取并传输信号。 将 ClockPhase 参数设置为 “ 交换 ” 将会互换 MSB 和 LSB 模块内 Φ_1 和 Φ_2 的作用，从而当下游用户模块对其输入进行取样时输出将会有效。（请注意，在 “ 正常 ” 模式下，输入电压在 Φ_1 中进行取样。）

GainRange

对于指定输入代码、输入电压和 AGND，从低到高更改增益范围将会相应增大输出电压。 请注意，高增益范围内的输入电压范围实际上是低增益范围内输入电压范围的一半。

DataFormat

MDAC8 用户模块 API 处理三种不同的数据格式：偏移二进制、2 的补码和符号及值。 本用户模块 API 部分中的 WriteBlind 入口点描述了这些规范和各种格式相关的值范围。

应用程序编程接口

应用程序编程接口 (API) 子程序作为用户模块的一部分提供，使设计人员能够在较高的层级处理模块。 本部分具体说明了每个函数对应的接口以及 “include” 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 寄存器的值可能因调用 API 函数而更改。 发起调用 API 的函数有责任在调用之前保留 A 和 X 的值 （如果调用后需要再次用到它们）。 选择这种 “ 寄存器易变 ” 策略是为了提高效率，并且自从 PsoC Designer 的 1.0 版本起使用。 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。 虽然有些用户模块的 API 函数可能保留 A 和 X 不变，但并不保证在未来也将如此。

入口点用于初始化 MDAC8 用户模块、写入更新值和禁用用户模块。

MDAC8_Start

说明：

执行此用户模块所需的所有初始化，并设置开关电容 PSoC 模块的功耗水平。

C 原型：

```
void  MDAC8_Start(BYTE bPowerSetting)
```

汇编程序：

```
mov   A, bPowerSetting
lcall  MDAC8_Start
```

参数：

bPowerSetting: 用于指定功率电平的一个字节。 在复位和配置之后，关闭分配给 DelSigMulti 的模拟 PSoC 模块的电源。 下表给出了 C 语言和汇编语言中提供的符号名及其相关值。

符号名	值
MDAC8_OFF	0
MDAC8_LOWPOWER	1
MDAC8_MEDPOWER	2
MDAC8_FULLPOWER	3

返回值：

无

副作用：

将会运行 MDAC 输出。 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 ” 前先调用一个写入子程序。 此函数可能更改 A 和 X 寄存器。

MDAC8_SetPower

说明：

设置 DAC 开关电容 PSoC 模块的功率电平。 可用于打开和关闭模块。

C 原型：

```
void  MDAC8_SetPower(BYTE bPowerSetting)
```

汇编程序：

```
mov   A, bPowerSetting
lcall  MDAC8_SetPower
```

参数：

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

返回值：

无

副作用：

将会运行 MDAC 输出。 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 ” 前先调用一个写入子程序。 此函数可能更改 A 和 X 寄存器。

MDAC8_SetOutputRange

说明：

通过将 FCap 设置为 32 （低范围：gain=1）或 16 （高范围：gain=2），为 MDAC 开关电容 PSoC 模块设置其中一个范围。

C 原型：

```
void  MDAC8_SetOutputRange(BYTE bRangeSetting)
```

汇编程序：

```
mov   A, bRangeSetting
lcall  MDAC8_SetOutputRange
```

参数：

RangeSetting： 用于指定范围设置的一个字节。

符号名	值
MDAC8_LOWRANGE	0
MDAC8_HIGHRANGE	1

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

MDAC8_SetPhase

说明：

设置内部 Φ1 和 Φ2 时钟为交换或正常 （默认）。

C 原型：

```
void  MDAC8_SetPhase(BYTE bPhaseSetting)
```

汇编程序：

```
mov   A, bPhaseSetting
lcall  MDAC8_SetPhase
```

参数：

bPhaseSetting： 用于指定正常或交换相位的一个字节。

符号名	值
MDAC8_NORMALPHASE	0
MDAC8_SWAPPEDPHASE	1

返回值：

无

副作用：

此函数可能更改 A 和 X 寄存器。

MDAC8_WriteBlind

说明：

即时更新输出电压到指示值。

C 原型：

```
// For OffsetBinary: (BYTE = unsigned char)
void  MDAC8_WriteBlind(BYTE bOutputValue)
// For TwosComplement: (CHAR = signed char)
void  MDAC8_WriteBlind(CHAR cOutputValue)
// For TwoByteSignAndMagnitude: (BYTE of bit flags)
void  MDAC8_WriteBlind2B(BYTE bLSB, BYTE bMSB)
```

汇编：

```
; for OffsetBinary
mov   A, bOutputValue
lcall  MDAC8_WriteBlind
; for TwosComplement
mov   A, cOutputValue
lcall  MDAC8_WriteBlind
; for TwoByteSignAndMagnitude format:
mov   A, bLSB
mov   X, bMSB
lcall  MDAC8_WriteBlind2B
```

参数：

b/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 的选择数值，如下表所示。

数据格式	最小值	最大值
OffsetBinary	0	254
TwosComplement	-127	127
TwoByteSignAndMagnitude	3F18h	1F38h

偏移二进制的值为正数，最低输出电压以 0 表示，最高输出电压以 254 表示。 2 的补码为 M8C 处理器的原始带符号格式。 在 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 且 t=1。

返回值：

无

副作用：

输出可能由于某些原因发生故障，原因如本用户模块的 “ 时序 ” 部分所述。 此函数可能更改 A 和 X 寄存器。

MDAC8_WriteStall

说明：

Phi1 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 假设中断已禁用或最大中断延迟小于 ACLKi （请参见 “ 快速更新时钟 ” 时序表的 “ 强制同步 ”）。

C 原型：

```
// For OffsetBinary: (BYTE = unsigned char)
void  MDAC8_WriteStall(BYTE bOutputValue)
// For TwosComplement: (CHAR = signed char)
void  MDAC8_WriteStall(CHAR cOutputValue)
// For TwoByteSignAndMagnitude: (BYTE of bit flags)
void  MDAC8_WriteStall2B(BYTE bLSB, BYTE bMSB)
```

汇编：

```
; for OffsetBinary
mov   A, bOutputValue
lcall  MDAC8_WriteStall
; for TwosComplement
mov   A, cOutputValue
lcall  MDAC8_WriteStall
; for TwoByteSignAndMagnitude format:
mov   A, bLSB
mov   X, bMSB
lcall  MDAC8_WriteStall2B
```

参数：

b/cOutputValue: 与 Write Blind 入口点所述参数的格式和值范围相同。bMSB 和 bLSB: 与 Write Blind 入口点所述参数的格式和值范围相同。

返回值：

无

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。 此函数可能更改 A 和 X 寄存器。

MDAC8_Stop

说明：

断开用户模块的电源。

C 原型：

```
void  MDAC8_Stop(void)
```

汇编：

```
lcall  MDAC8_Stop
```

参数：

无

返回值：

无

副作用：

不会驱动输出。 此函数可能更改 A 和 X 寄存器。

固件源代码示例

此示例代码创建了周期性缓慢下降的锯齿波。

```
;;-------------------------------------------------------------------
;; Sample Code for the MDAC8
;; Generate a falling sawtooth wave
;;-------------------------------------------------------------------

export _main
include "m8c.inc"
include "MDAC8.inc"

      area bss (RAM)
bVal: blk  1                ; RAM for loop iteration variable
bMAXVAL: equ  255           ; Top of ramp plus 1
      area text (ROM, REL)

_main:                      ; (contains infinite loop; never returns)
      mov  A, MDAC8_LOWPOWER ; specify DAC's amplify power
      call MDAC8_Start      ; and turn it on.
Init:
      mov  [bVal], bMAXVAL  ; Start ramp from the top
RampDown:
      mov  A, [bVal]        ;
      dec  A
```

```
      call MDAC8_WriteStall
      dec  [bVal]            ; Bottom of ramp?
      jnz  RampDown          ; No, not yet.
      jmp  Init              ; Yes, re-initialize ramp and loop
                             ; forever


//-------------------------------------------------------------------
// C main line
//-------------------------------------------------------------------

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

    BYTE cVal;
    #define cMax 255


void main(void)
{
    // Insert your main routine code here.
    MDAC8_Start(MDAC8_LOWPOWER);
    while(1) //infinite loop
    {
        cVal = cMax;
        while(cVal > 0)
        {
            MDAC8_WriteStall(cVal--);
        }
    }
}
```

配置寄存器

API 为 MDAC8 用户模块提供完整接口。 直接写入到配置寄存器是更新输出的另一种方式。 无论用哪种方式，都必须了解时序注意事项以防止输出故障。 以下寄存器用于 MDAC8 开关电容 LSB 和 MSB 模块。

Table 5. 模块 LSB：寄存器 CR0

位	7	6	5	4	3	2	1	0
值	1	0	标记	大小		0	0	0

“标记”(Sign) 使用 “1” 表示正值 （AGND 到 RefHi），使用 “0” 表示负值 （RefLow 到 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 中的一个写入函数来更改标记值。 “大小”(Magnitude) 的默认值为 “0”。 使用 API 中的一个写入函数来更改该值。

Table 6. 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 7. 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

模拟总线已禁用。

Table 8. 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	电源	
开关电容类型 B								
位	7	6	5	4	3	2	1	0
值	0	0	1	1	1	0	电源	

电源: 默认为 “ 关闭 ”。 使用 API 中的 “ 启动 ”(Start) 调用来设置该值。

Table 9. 模块 MSB: 寄存器 CR0

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

“ 标记 ”(Sign) 由 API 写入子程序设置。 默认值为 “0”。 “ 大小 ”(Magnitude) 可通过使用 API 中的一个写入函数来更改。 默认值也是 “0”。

Table 10. 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 11. 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，模拟总线在配置时间启用或禁用。

Table 12. 模块 MSB: 寄存器 CR3

位	7	6	5	4	3	2	1	0
值	0	0	1	1	BMux		电源	

BMux 配置用于从 LSB PSoC 模块中选择连接。 电源: 0= 关闭 （默认），1= 低，2= 中，3= 满。 使用 API 中的 “ 启动 ”(Start) 调用来设置该值。