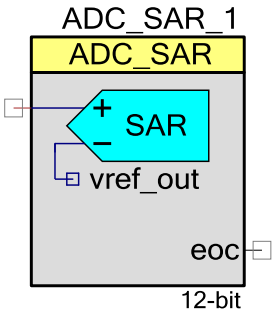
ADC 逐次逼近寄存器 (ADC_SAR)

1.71

特性



- 支持 PSoC 5 器件
- 700 ksps（最大）时为 12 位分辨率
- 四个功耗模式
- 可选分辨率和采样率
- 单端或差分输入

概述

ADC 逐次逼近寄存器 (ADC_SAR) 组件提供中等速度（最大 700-ksps 采样）、中等分辨率（最大 12 位）、模数转换。

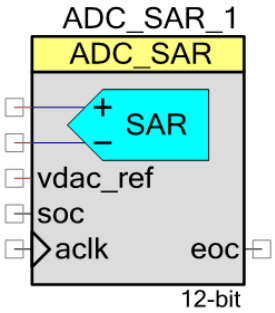
何时使用 ADC_SAR

ADC_SAR 组件的典型应用包括：

- LED 照明控制
- 电机控制
- 磁卡读卡器
- 高速数据收集
- 功率计
- 脉搏血氧计

输入/输出连接

本节介绍 ADC_SAR 的输入和输出连接。I/O 列表中的星号 (*) 表示该 I/O 可能在 I/O 说明中列出的情况下隐藏在符号中。



+输入 – 模拟

此输入是 ADC_SAR 的正向模拟信号输入。转换结果取决于 +输入信号减去电压参考。电压参考是 –输入信号或 V_{SSA}。

–输入 – 模拟 *

显示时，此可选输入是 ADC_SAR 的负向模拟信号（或参考）输入。转换结果取决于 +输入减去 –输入。在将 **Input Range**（输入范围）参数设置为差分模式之一时可看到此引脚。

vdac_ref – Input *

VDAC 参考 (vdac_ref) 是可选引脚。如果选择了 **Vssa to VDAC*2 (Single Ended)**（Vssa 至 VDAC*2（单端））或 **0.0 +/- VDAC (Differential)**（0.0 +/- VDAC（差分））输入范围，则可看到该引脚；否则，此 I/O 会隐藏。只能将此引脚连接到 VDAC 组件输出。请勿将其连接到任何其他信号。

soc – 输入 *

开始转换 (soc) 是可选引脚。如果选择 **Triggered**（触发）采样模式，则可看到该引脚。此输入的上升沿会开始 ADC 转换。如果将 **Sample Mode**（采样模式）参数设置为 **Free Running**（自由运行），则此 I/O 会隐藏。

aclk – 输入 *

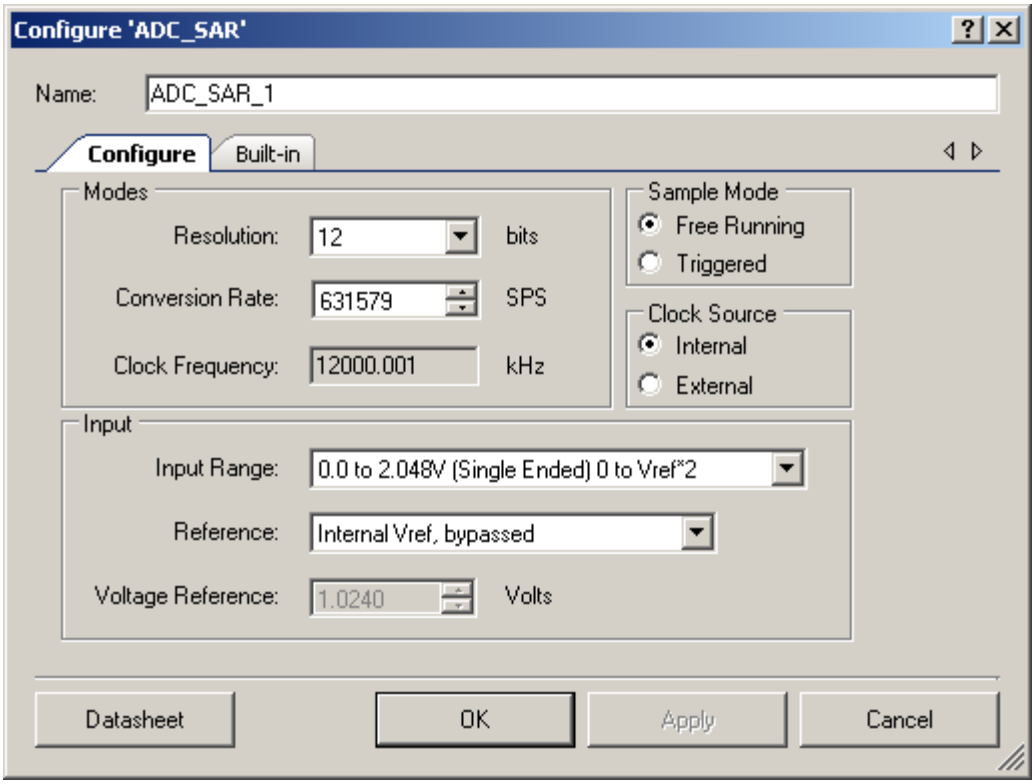
如果将 **Clock Source**（时钟源）参数设置为 **External**（外部），则可看到此可选引脚；否则，此引脚会隐藏。此时钟确定了转换速率取决于转换方法和分辨率。

eoc – 输出

结束转换 (eoc) 输出上的上升沿表示转换已经完成。可以将 DMA 请求连接到此引脚以将转换输出传输到系统 RAM、DFB 或其他组件。可将内部中断也连接到此信号，或者可以连接自己的中断。

元件参数

将一个 ADC_SAR 组件拖放到您的设计上，并双击以打开 **Configure**（配置）对话框。



ADC_SAR 具有以下参数。以粗体显示的选项是默认值。

模式

分辨率

设置 ADC 的分辨率。

ADC_Resolution	值	说明
12	12	将分辨率设置为 12 位。
10	10	将分辨率设置为 10 位。
8	8	将分辨率设置为 8 位。

SAR 始终在 12 位模式下运行。8 位和 10 位选项会保留，但是仅影响 ADC_GetResult16() 和 ADC_GetResult8() API。

转换速率

此参数设置 ADC 转换。转换时间与转换速率成反比。以每秒采样数为单位输入转换速率。转换一个采样需要 19 个 SAR ADC 时钟周期。

时钟频率

该文本框为只读（始终为灰色）区域，用于显示所选工作条件（分辨率和转换速率）所需的时钟速率。当这两个条件中的任一条件更改或两个条件都更改时，会更新该文本框。时钟频率可以为介于 1 MHz 与 14 MHz 之间的任何值。占空比应为 50%。最小脉冲宽度应大于 33 ns。如果时钟不在这些限制范围内，则 PSoC Creator 会在构建过程中生成错误。在这种情况下，在设计范围资源时钟编辑器中更改主控时钟。

Sample Mode（采样模式）

此参数确定 ADC 的工作方式。

Start_of_Conversion	说明
Free Running（自由运行）	ADC 连续运行。
Triggered（触发）	SOC 引脚上的上升沿脉冲开始单个转换。

时钟源

通过此参数可以选择 ADC_SAR 模块内部的时钟或外部时钟。

ADC_Clock	说明
Internal（内部）	使用 ADC_SAR 的内部时钟。
External（外部）	使用外部时钟。时钟源可以是模拟、数字或由其他组件生成。

输入

输入范围

该参数按给定的输入范围配置 ADC。无论使用了什么输入范围设置，连接至 PSoC 的模拟信号都必须在 V_{SSA} 与 V_{DDA} 之间。

输入范围	说明
0.0 to 2.048V (Single Ended)（0.0 至 2.048V（单端）） **0 to Vref*2（0 至 Vref*2）**	当使用内部参考 (1.024 V) 时，可用输入范围为 0.0 至 2.048 V。ADC 配置为单端输入模式下运行，同时 –输入在内部连接到 Vrefhi_out。如果使用外部参考电压，则可用输入范围为 0.0 至 Vref*2。

输入范围	说明
Vssa to Vdda (Single Ended) （Vssa 至 Vdda（单端））	此模式使用 $V_{DDA}/2$ 参考；可用输入范围涵盖完整模拟电源电压。ADC 置于单端输入模式，同时 –输入在内部连接到 Vrefhi_out。
Vssa to VDAC*2 (Single Ended) （Vssa 至 VDAC*2（单端））	此模式使用 VDAC 参考，该参考应连接到 vdac_ref 引脚。可用输入范围为 Vssa 到 VDAC*2 伏。ADC 配置为在单端输入模式下运行，同时 –输入在内部连接到 Vrefhi_out。
0.0 ± 1.024V (Differential) （0.0 ± 1.024V（差分）） –输入 ± Vref	此模式配置为用于差分输入。当使用内部参考 (1.024 V) 时，输入范围为 –输入 ± 1.024 V。 例如，如果 –输入连接到 2.048 V，则可用输入范围为 2.048 ± 1.024 V 或 1.024 至 3.072 V。对于需要扫描单端和差分信号的系统，在扫描单端输入时将 –输入连接到 Vssa。 可以使用外部参考提供较宽工作范围。可以使用相同等式 –输入 ± Vref 计算可用输入范围。
0.0 ± Vdda (Differential) （0.0 ± Vdda（差分）） –输入 ± Vdda	此模式配置用于差分输入，是针对电源电压的比率计。输入范围为 –输入 ± Vdda。对于需要扫描单端和差分信号的系统，在扫描单端输入时，将 –输入连接至 Vssa。
0.0 ± Vdda/2 (Differential) （0.0 ± Vdda/2（差分）） –输入 ± Vdda/2	此模式配置用于差分输入，是针对电源电压的比率计。输入范围为 –输入 ± Vdda/2。对于需要扫描单端和差分信号的系统，在扫描单端输入时，将 –输入连接至 Vssa
0.0 ± VDAC (Differential) （0.0 ± VDAC（差分）） –输入 ± VDAC	此模式配置为用于差分输入，使用 VDAC 参考，该参考应连接到 vdac_ref 引脚。输入范围为 –输入 ±VDAC。对于需要扫描单端和差分信号的系统，在扫描单端输入时，将 –输入连接至 Vssa。

参考

此参数选择 ADC_SAR 参考配置的开关。

ADC_Reference	说明
内部 Vref	使用内部参考。此选项允许的最大采样率为 100,000 sps。对于较高采样率，请使用 **Internal Vref, bypassed**（内部 Vref，绕过）选项。
Internal Vref, bypassed **（内部 Vref，绕过）**	使用内部参考；必须将一个绕过电容器置于 SAR1 的引脚 P0[2]* 上或 SAR0 的引脚 P0[4]* 上。
External Vref（外部 Vref）	在 SAR1 的引脚 P0[2]* 上或 SAR0 的引脚 P0[4] 上使用外部参考。

***** 如果数字开关导致的内部噪声超过应用的模拟性能要求，则建议使用外部绕过电容器。若要使用此选项，请将端口引脚 P0[2] 或 P0[4] 配置为模拟 HI-Z 引脚，并连接一个值介于 0.01 µF 与 10 µF 之间的外部电容器。

电压参考

电压参考用于计入应用程序编程接口一节中讨论的电压转换函数的 ADC。当使用内部参考时，此参数为只读。当使用外部参考时，可以编辑此值以匹配外部参考电压。

- 当选择输入范围 **Vssa to Vdda**（Vssa 至 Vdda）、**–输入 +/- Vdda** 或 **–输入 +/- Vdda/2** 时，输入 V_{DDA} 电源电压。
- 当选择输入范围 **Vssa to VDAC*2**（Vssa 至 VDAC*2）或 **–输入 +/- VDAC** 时，输入 VDAC 电源电压值。

注：输入范围和参考电压受 V_{DDA} 电压限制。

放置

ADC_SAR 组件放置在两个可用 SAR 模块之一中。放置信息通过 *cyfitter.h* 文件提供给 API。如果需要更改默认放置，请使用设计范围资源 – 指令编辑器（在项目的 .cydwr 文件中）编辑参数。

资源

ADC_SAR 使用芯片中的固定模块 SAR 和时钟源。

资源	资源类型				API 存储器（字节）		引脚（每个外部 I/O）
	时钟分频器	宏单元	中断	SAR 固定模块	闪存	RAM	
8 到 12 位	1	1	1	1	1106	7	1

应用程序编程接口

应用程序编程接口 (API) 子程序允许您使用软件配置组件。下表列出了每个函数的接口，并进行了说明。以下各节将更详细地介绍每个函数。

默认情况下，PSoC Creator 将实例名称 “ADC_SAR_1” 分配给指定设计中组件的第一个实例。您可以将该实例重命名为符合标识符语法规则的任意唯一值。实例名称会成为每个全局函数名称、变量和常量符号的前缀。为增加可读性，下表中使用了实例名称 “ADC”。

函数	说明
ADC_Start()	对 ADC 加电并复位所有状态
ADC_Stop()	停止 ADC 转换并将功耗减少到最小值
ADC_SetPower()	设置功耗模式

函数	说明
ADC_SetResolution()	设置 ADC 的分辨率
ADC_StartConvert()	开始转换
ADC_StopConvert()	停止转换
ADC_IRQ_Enable()	内部 IRQ 连接到 eoc。此 API 启用内部 ISR。
ADC_IRQ_Disable()	内部 IRQ 连接到 eoc。此 API 禁用内部 ISR。
ADC_IsEndConversion()	如果转换完成，则返回非零值
ADC_GetResult8()	返回有符号 8 位转换结果
ADC_GetResult16()	返回有符号 16 位转换结果
ADC_SetOffset()	设置 ADC 的偏移
ADC_SetGain()	每伏电压的 ADC 增益计数
ADC_CountsTo_Volts()	将 ADC 计数转换为单位为伏的浮点电压值
ADC_CountsTo_mVolts()	将 ADC 计数转换为单位为毫伏的电压值
ADC_CountsTo_uVolts()	将 ADC 计数转换为单位为微伏的电压值
ADC_Sleep()	停止 ADC 操作，并保存用户配置
ADC_Wakeup()	恢复并使能用户配置
ADC_Init()	初始化随自定义程序提供的默认配置
ADC_Enable()	启用 ADC 的时钟，并使其通电
ADC_SaveConfig()	保存当前用户配置
ADC_RestoreConfig()	恢复用户配置

全局变量

变量	说明
ADC_initVar	此变量指示 ADC 是否已初始化。该变量初始化为 0，并在第一次调用 ADC_Start() 时设置为 1。这样，第一次调用 ADC_Start() 子程序后，组件不用重新初始化即可重启。 如需重新初始化组件，可在 ADC_Start() 或 ADC_Enable() 函数前调用 ADC_Init() 函数。
ADC_offset	此变量校准偏移。它在首次调用 ADC_Start() 时设置为 0，可以使用 ADC_SetOffset() 进行修改。通过减去给定偏移，该变量可影响 ADC_CountsTo_Volts()、ADC_CountsTo_mVolts() 和 ADC_CountsTo_uVolts() 函数。

变量	说明
ADC_countsPerVolt	此变量用于校准增益。它在首次调用 ADC_Start() 时以及每次调用 ADC_SetResolution() 时进行计算。值取决于分辨率、输入范围和电压参考。它可以使用 ADC_SetGain() 进行更改。 通过在 ADC 计数和施加的输入电压之间进行正确转换，此变量可仅影响 ADC_CountsTo_Volts、ADC_CountsTo_mVolts 和 ADC_CountsTo_uVolts 函数。
ADC_shift	在差分输入模式中，SAR ADC 以二进制偏移方案输出经过数字转换的数据。此变量用于将 ADC 计数转换为 2 的补码形式。 此变量在首次调用 ADC_Start() 时以及每次调用 ADC_SetResolution() 时进行计算。计算值取决于分辨率和输入模式。 通过减去正确移位值，此变量可影响 ADC_GetResult8() 和 ADC_GetResult16() 函数。

void ADC_Start(void)

说明： 这是开始执行组件操作的首选方法。ADC_Start() 设置 initVar 变量，调用 ADC_Init() 函数，然后调用 ADC_Enable() 函数。

参数： 无

返回值： 无

副作用： 如果已设置 initVar 变量，则该函数仅调用 ADC_Enable() 函数。

void ADC_Stop(void)

说明： 停止 ADC 转换并将功耗减少到最小值。

注：此 API 不会断开 ADC 电源，但是会将功耗减少到最小值。此器件有一个缺陷，导致与某些模拟资源的连接在器件未通电时不可靠。当停止使用该资源的组件时，该不可靠性会在静默失败中表现出来（例如模拟组件中出现不可预见的失败结果）。

参数： 无

返回值： 无

副作用： 无

void ADC_SetPower(uint8 power)

说明： 设置 ADC 的运行功耗。应将较高功耗设置用于较快时钟速度。

参数： uint8 功耗：功耗设置

参数名称	值	说明	时钟频率
ADC__HIGHPOWER	0	正常功耗	14 MHz
ADC__MEDPOWER	1	1/2 功耗	7 MHz
ADC__LOWPOWER	2	1/3 功耗	4.6 MHz
ADC__MINPOWER	3	1/4 功耗	3.5 MHz

返回值： 无

副作用： 功耗设置可能会影响转换精度。

void ADC_SetResolution(uint8 resolution)

说明： 设置 GetResult16() 和 GetResult8() API 的分辨率。此函数不影响实际转换。

参数： uint8 分辨率：分辨率设置

参数名称	值	说明
ADC__BITS_12	12	将分辨率设置为 12 位。
ADC__BITS_10	10	将分辨率设置为 10 位。
ADC__BITS_8	8	将分辨率设置为 8 位。

返回值： 无

副作用： 在转换周期过程中不能更改 ADC 分辨率。建议的最佳实践是使用 ADC_StopConvert() 停止转换，更改分辨率，然后使用 ADC_StartConvert() 重新启动转换。

如果决定在调用此 API 之前不停止转换，请使用 ADC_IsEndConversion() 等到转换完成，然后再更改分辨率。

如果在转换过程中调用 ADC_SetResolution()，则在当前转换完成之前，分辨率不会更改。对于另外 6 +“新分辨率（以位为单位）”个时钟周期，数据在新分辨率中不可用。在调用 ADC_SetResolution() 之后可能需要添加此时钟周期数的延迟，数据才能再次有效。

通过在 ADC 计数和施加的输入电压之间计算正确转换，可影响 ADC_CountsTo_Volts()、ADC_CountsTo_mVolts() 和 ADC_CountsTo_uVolts()。计算取决于分辨率、输入范围和电压参考。

void ADC_StartConvert(void)

说明： 强制 ADC 开始进行转换。在自由运行模式中，ADC 连续运行。在触发模式中，该函数还充当软件版本的 SOC，并且必须通过 ADC_StartConvert() 触发每个转换。

参数： 无

返回值： 无

副作用： 调用 ADC_StartConvert() 会禁用外部 SOC 引脚。

void ADC_StopConvert(void)

说明： 强制 ADC 停止转换。如果转换当前正在执行，则该转换会完成，但是不会进行进一步转换。

参数： 无

返回值： 无

副作用： 在触发模式中，此函数将软件版本的 SOC 设置低级别并将 SOC 源切换为硬件 SOC 输入。

void ADC_IRQ_Enable(void)

说明： 转换结束后，启用中断。还必须启动全局中断，以实现 ADC 中断。若要启用全局中断，请在启用任何中断之前，在 *main.c* 文件中启用全局中断宏 "`CYGlobalIntEnable;`"。

参数： 无

返回值： 无

副作用： 启用中断。读取该结果清除中断。

void ADC_IRQ_Disable(void)

说明： 转换结束后，禁用中断。

参数： 无

返回值： 无

副作用： 无

uint8 ADC_IsEndConversion(uint8 retMode)

说明： 立即返回转换状态或是在转换完成之前不返回（封锁），具体取决于 retMode 参数。

参数： uint8 retMode：检查转换返回模式。有关选项，请参见下表。

选项	说明
ADC_RETURN_STATUS	立即返回状态。如果返回值为零，则转换未完成，并且应重试此函数，直至返回非零结果。
ADC_WAIT_FOR_RESULT	在 ADC 转换完成之前不返回结果。

返回值： uint8：如果返回非零值，则最后一次转换已完成。如果返回值为零，则 ADC 仍在计算最后的结果。

副作用： 此函数可读取在读取后清除的转换结束状态。

int8 ADC_GetResult8(void)

说明： 返回 8 位转换的结果。如果设置的分辨率大于 8 位，则函数返回结果的最低有效位。当设置的分辨率小于 12 位时，此函数返回移位值。应调用 ADC_IsEndConversion() 以验证数据采样是否就绪。

参数： 无

返回值： int8：最后一次 ADC 转换的最低有效位。

副作用： 将 ADC 计数转换为 2 的补码形式。

int16 ADC_GetResult16(void)

说明： 返回转换的 16 位结果，以及分辨率为 8 至 12 位的结果。当设置的分辨率小于 12 位时，此函数返回移位值。应调用 ADC_IsEndConversion() 以验证数据采样是否就绪。

参数： 无

返回值： int16：最后一次 ADC 转换的 16 位结果

副作用： 将 ADC 计数转换为 2 的补码形式。

void ADC_SetOffset(int16 offset)

说明： 设置 ADC_CountsTo_Volts()、ADC_CountsTo_mVolts() 和 ADC_CountsTo_uVolts() 所用的 ADC 偏移，以便在计算电压转换前，从给定读数中减去该偏移量。

参数： int16 offset：当输入短接或连接到相同输入电压时，会测量此值。

返回值： 无

副作用： 通过减去给定偏移，可影响 ADC_CountsTo_Volts()、ADC_CountsTo_mVolts() 和 ADC_CountsTo_uVolts()。

void ADC_SetGain(int16 adcGain)

说明： 为以下电压转换函数设置每伏电压的 ADC 增益计数。默认情况下，该值由参考和输入范围设置设定。该值仅可用于进一步校准具有已知输入的 ADC，或仅在 ADC 使用外部参考的情况下使用。

参数： int16 adcGain：每伏电压的 ADC 增益计数

返回值： 无

副作用： 通过在 ADC 计数和施加的输入电压之间进行正确转换，可影响 ADC_CountsTo_Volts()、ADC_CountsTo_mVolts()、ADC_CountsTo_uVolts() 函数。

float ADC_CountsTo_Volts(int16 adcCounts)

说明： 将 ADC 输出转换为单位为伏的浮点电压值。例如，如果测得的 ADC 输出为 0.534 伏，则返回值为 0.534。

参数： int16 adcCounts：ADC 转换的结果

返回值： Float：单位为伏的结果

副作用： 无

int16 ADC_CountsTo_mVolts(int16 adcCounts)

说明： 将 ADC 输出转换为单位为毫伏的 16 位整数电压值。例如，如果测得的 ADC 输出为 0.534 伏，则返回值为 534。

参数： int16 adcCounts：ADC 转换的结果

返回值： int16：单位为 mV 的结果

副作用： 无

int32 ADC_CountsTo_uVolts(int16 adcCounts)

说明： 将 ADC 输出转换为单位为微伏的 32 位整数电压值。例如，如果测得的 ADC 输出为 0.534 伏，则返回值为 534000。

参数： int16 adcCounts：ADC 转换的结果

返回值： int32：单位为 µV 的结果

副作用： 无

void ADC_Sleep(void)

说明： 这是准备组件睡眠的首选子程序。ADC_Sleep() 子程序保存当前组件的状态。然后调用 ADC_Stop() 函数并调用 ADC_SaveConfig() 以保存硬件配置。

在调用 CyPmSleep() 或 CyPmHibernate() 函数之前调用 ADC_Sleep() 函数。有关电源管理函数的更多信息，请参考 PSoC Creator *System Reference Guide*（《系统参考指南》）。

参数： 无

返回值： 无

副作用： 无

void ADC_Wakeup(void)

说明： 这是将组件恢复到调用 ADC_Sleep() 时状态的首选子程序。ADC_Wakeup() 函数调用 ADC_RestoreConfig() 函数以恢复配置。如果组件在调用 ADC_Sleep() 函数前已启用，则 ADC_Wakeup() 函数还将重新启用组件。

参数： 无

返回值： 无

副作用： 调用 ADC_Wakeup() 函数前未调用 ADC_Sleep() 或 ADC_SaveConfig() 函数可能会产生意外行为。

void ADC_Init(void)

说明： 根据自定义程序“配置”对话框设置来初始化或恢复组件。无需调用 ADC_Init()，因为 ADC_Start() 子程序会调用该函数并是开始组件操作的首选方法。

参数： 无

返回值： 无

副作用： 所有寄存器将设置为自定义程序“配置”对话框中的值。

void ADC_Enable(void)

说明： 激活硬件并开始执行组件操作。会根据时钟速度自动设置较高功耗。ADC_SetPower() API 说明包含功耗与时钟速率的关系。无需调用 ADC_Enable()，因为 ADC_Start() 子程序会调用该函数，这是开始组件操作的首选方法。

参数： 无

返回值： 无

副作用： 无

void ADC_SaveConfig(void)

说明： 此函数会保存组件配置和非保留寄存器。它还保存 Configure（配置）对话框中定义的或通过相应 API 修改的当前组件参数值。该函数由 ADC_Sleep() 函数调用。

参数： 无

返回值： 无

副作用： 会保留所有 ADC 配置寄存器。此函数没有实现，旨在供将来使用。此处提供它是为了使 API 在组件间保持一致。

void ADC_RestoreConfig(void)

说明： 此函数会恢复组件配置和非保留寄存器。它还将组件参数值恢复为在调用 ADC_Sleep() 函数之前的值。

参数： 无

返回值： 无

副作用： 调用此函数前未调用 ADC_Sleep() 或 ADC_SaveConfig() 函数可能会产生意外行为。此函数没有实现，旨在供将来使用。此处提供它是为了使 API 在组件间保持一致。

DMA

可以使用 DMA 组件将转换的结果从 ADC_SAR 寄存器传输到 RAM。应将 DMA 数据请求信号 (DRQ) 从 ADC 连接至 EOC 引脚。可以使用 DMA 向导按如下所示配置 DMA 操作：

DMA 源的名称	长度	方向	DMA 请求信号	DMA 请求类型	说明
ADC_SAR_WRK0_PTR	2	源	EOF	上升沿	接收转换的 2 字节结果，以及分辨率始终为 12 位的结果。 请注意，此寄存器未进行符号扩展；结果始终是无符号的。0-V 差分输入返回半量程代码。完整负向输入返回 0 代码，完整正向输入返回满量程代码。

固件源代码示例

PSoC Creator 在 Find Example Project（查找示例项目）对话框中提供了许多包括原理图和示例代码的示例项目。要获取组件特定的示例，请打开组件目录中的对话框或原理图中的组件实例。要获取通用的示例，请打开开始页或 **File**（文件）菜单中的对话框。根据需要，使用对话框中的 **Filter Options**（滤波器选项） 可缩小可选项目的列表。

有关更多信息，请参考 PSoC Creator 帮助中的“查找示例项目”主题。

中断服务子程序

ADC_SAR 在文件 *ADC_SAR_1_INT.c*（其中“ADC_SAR_1”为实例名称）中包含空白中断服务子程序。可以将自定义代码置于指定区域中以执行转换结束时所需的任何函数。下面显示了空白中断服务子程序的副本。将自定义代码置于“/* `#START MAIN_ADC_ISR` */”与“/* `#END` */”注释之间。这可确保在重新生成项目时保留代码。

```
CY_ISR( ADC_SAR_1_ISR )
{
    /* Place user ADC ISR code here. This can be a good place */
    /* to place code that is used to switch the input to the  */
    /* ADC. It may be good practice to first stop the ADC     */
    /* before switching the input then restart the ADC.       */

    /* `#START MAIN_ADC_ISR` */
    /* Place user code here. */
    /* `#END`  */
}
```

第二个指定区域可用于放置变量定义和常量定义。

```
/* System variables */
```

```
/* `#START ADC_SYS_VAR`  */
    /* Place user code here. */
/* `#END`  */
```

下面是使用中断捕获数据的代码示例。

```
#include <device.h>

int16  result = 0;
uint8 dataReady = 0;
void main()
{
    int16 newReading = 0;
    CYGlobalIntEnable;           /* Enable Global interrupts */
    ADC_SAR_1_Start();           /* Initialize ADC */
    ADC_SAR_1_IRQ_Enable();      /* Enable ADC interrupts */
    ADC_SAR_1_StartConvert();    /* Start ADC conversions */
    for(;;)
    {
        if (dataReady != 0)
        {
           dataReady = 0;
           newReading = result;
           /* More user code */
        }
    }
}
```

文件 *ADC_SAR_1_INT.c* 中的中断代码段。

```
    /*********************************
    *      System variables
    *********************************/
    /* `#START ADC_SYS_VAR`  */
     extern int16 result;
     extern uint8 dataReady;
     /* `#END`  */

CY_ISR(ADC_SAR_1_ISR )
{
    /*********************************************/
    /* Place user ADC ISR code here.             */
    /* This can be a good place to place code    */
    /* that is used to switch the input to the   */
    /* ADC. It may be good practice to first     */
    /* stop the ADC before switching the input   */
    /* then restart the ADC.                     */
    /*********************************************/
    /* `#START MAIN_ADC_ISR`  */
        result = ADC_SAR_1_GetResult16();
        dataReady = 1;
    /* `#END`  */
}
```

正确设置 Conversion Rate（转换速率）和 Master Clock（主控时钟）参数十分重要。

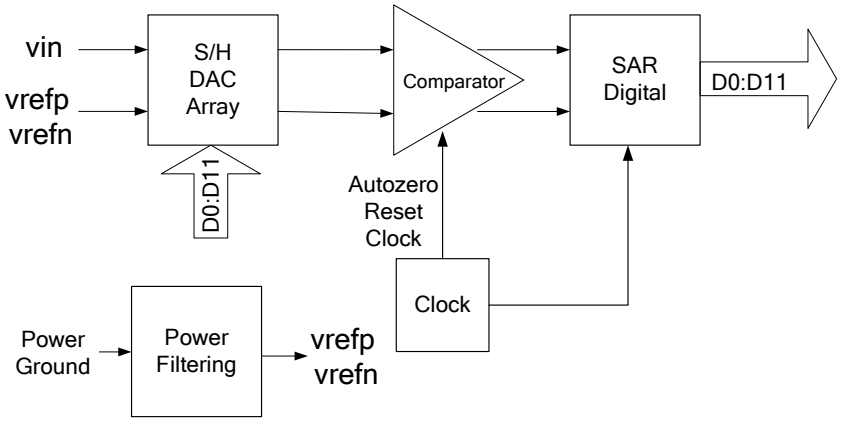
例如，对于最大转换速率（12 位时为 700 ksps），在设计范围资源时钟编辑器中将 Master Clock（主控时钟）设置为 53 MHz，并优化 ISR 子程序。否则，处理器将无法以足够快的速度处理 ISR。如果选择较低 Master Clock（主控时钟），则 ISR 的运行时间会长于 ADC_SAR 转换时间。

可以通过直接读取采样寄存器优化 ISR：

```
CY_ISR(ADC_SAR_1_ISR )
{
    /*********************************************/
    /* Place user ADC ISR code here.             */
    /* This can be a good place to place code    */
    /* that is used to switch the input to the   */
    /* ADC. It may be good practice to first     */
    /* stop the ADC before switching the input   */
    /* then restart the ADC.                     */
    /*********************************************/
    /* `#START MAIN_ADC_ISR`  */
        result = CY_GET_REG16(ADC_SAR_1_SAR_WRK0_PTR);
        dataReady = 1;
    /* `#END`  */
}
```

功能描述

下图显示框图。对输入模拟信号进行采样，并使用二进制搜索算法将其与 DAC 的输出进行比较以确定从 MSB 到 LSB 的输出位。



寄存器

采样寄存器

ADC 结果可能在 8 至 12 位分辨率之间。输出分为两个 8 位寄存器。CPU 或 DMA 可访问这些寄存器，以读取 ADC 结果。

ADC_SAR_WRK0_REG（SAR 工作寄存器 0）

位	7	6	5	4	3	2	1	0
值	Data[7:0]							

ADC_SAR_WRK1_REG（SAR 工作寄存器 1）

位	7	6	5	4	3	2	1	0
值	overrun_det	NA			Data[11:8]			

- Data[11:0]：ADC 结果
- overrun_det：数据过速检测标志。默认情况下，禁用此功能。

直流和交流电气特性

下列值表示期望的性能，它们基于初始特性数据。除非另有指定，否则运行条件为：

- 在连续采样模式下运行
- Fclk = 14 MHz
- 输入范围 = ±V_{REF}
- 10 µF 的绕过电容器

SAR ADC 直流规范

参数	说明	条件	最小值	典型值	最大值	单位
	分辨率		8	–	12	位
	通道数量 – 单端		–	–	GPIO 数量	–
	通道数量 – 差分	差分对由一对 GPIO 组成。	–	–	GPIO 数量/2	–
	单调性[1]		是	–	–	
Ge	增益误差	外部参考	–	–	±0.2	%
V_{OS}	输入偏移电压	V_{CM} = 0 V	–	–	±2	mV
		V_{CM} = V_{DD}/2	–	–	±6	
I_{DD}	电流消耗		–	–	1	mA
	输入电压范围 – 单端 [1]		V_{SSA}	–	V_{DDA}	V
	输入电压范围 – 差分 [1]		V_{SSA}	–	V_{DDA}	V
	外部参考输入电压范围		1.0	–	V_{DDA}	V
PSRR	电源抑制比 [1]		70	–	–	dB
CMRR	共模抑制比		35	–	–	dB
INL	积分非线性度 [1]	来自 V_{BG} 的内部参考	–	–	±2	LSB
DNL	微分非线性度 [1]	来自 V_{BG} 的内部参考	–	–	±2	LSB

[1] 基于器件特性表征（未经过生产测试）。

SAR ADC 交流规范

参数	说明	条件	最小值	典型值	最大值	单位
	采样率 [2]	具有绕过电容器	–	–	700	ksps
		没有绕过电容器	–	–	100	
	启动时间 [2]		–	–	10	µs
SINAD	信噪比 [2]	$V_{DDA} \le 3.6$ V，$V_{REF} \le 3.6$ V	57	–	–	dB
		3.6 V < $V_{DDA} \le 5.5$ V $V_{REF} < 1.3$ V 或 $V_{REF} > 1.8$ V	57	–	–	
THD	总谐波失真 [2]	$V_{DDA} \le 3.6$ V，$V_{REF} \le 3.6$ V	–	–	0.1	dB
		3.6 V < $V_{DDA} \le 5.5$ V $V_{REF} < 1.3$ V 或 $V_{REF} > 1.8$ V	–	–	0.1	

[2] 基于器件特性表征（未经过生产测试）。

组件更改

本节介绍组件与以前版本相比的主要更改。

版本	更改说明	更改/影响原因
1.71	修正了 ADC_GetResult8() 和 ADC_GetResult16() API 以执行一个 16 位读取操作而非两个 8 位读取。	如果在读取了一个字节之后 SAR ADC 更新输出采样寄存器，则生成的数据可能会损坏。
	修正了 ADC_IsEndConversion() API 以等待释放 EOF 状态位。	此函数可能会在快速连续调用之后返回意外转换完成状态。
1.70	更正了 SampleRate 错误提供程序消息中的最小值。	
	在选择“VDAC”作为 Input Range（输入范围）时，会在 Reference（参考）下拉列表中隐藏“External Vref（外部 Vref）”项目。	当选择 VDAC 范围时，不可使用外部参考。
	当选择了 External Vref（外部 Vref）选项时，将外部引脚重命名为“ExtVref”。当选择了 Internal reference with Bypass（带绕过的内部参考）选项时，保留名称“Bypass”。	用于将引脚名称与功能匹配。
	数据手册校正	
1.60	从自定义程序中删除了“Power（功耗）”参数。	会根据时钟速度自动设置较高功耗。 ADC_SetPower() API 说明包含功耗与时钟速率的关系。

版本	更改说明	更改/影响原因
	SAR 在 12 位模式下运行。8 位和 10 位选项会保留，但是仅影响 ADC_GetResult16() API。	在 8 位和 10 位模式下，SAR ADC 仅将 ODD 计数显示为输出。
	将默认 SAR 转换速率从 1 Msps 更改为 631579 sps（12-MHz 时钟）。	SAR 应能够用默认设置进行放置和构建。
	ADC_Stop() API 不会关闭 ADC 的电源，但是会将功耗减少到最小值。	PSoC 5 芯片有一个缺陷，导致与某些模拟资源的连接在未通电时不可靠。
	将转换时间从 18 个周期更改为 19 个周期。	用于提高 SAR 性能。
1.50.a	添加了时钟频率验证。	此更改提供一种方式避免使用带有不规范时钟的 SAR ADC。 如果从版本 1.10 的 SAR ADC 组件更新并使用工作范围之外的时钟，请选择正确时钟频率。
	向组件中添加了信息，以说明它与芯片修订版的兼容性。	如果组件在不兼容的芯片上使用，该工具将报告错误/警告。如果发生此情况，请更新到支持您的目标器件的修订版。
	对数据表进行了少量编辑和更新	
1.50	添加了睡眠/唤醒和初始化/启用 API。	用于支持低功耗模式并提供常用接口，以单独控制大多数组件的初始化和启用。
	添加了 ADC_CountsTo_Volts 和 ADC_CountsTo_uVolts API。	扩展功能。此 API 以伏和毫伏为单位返回转换结果。
	向组件中添加了 DMA 功能文件。	此文件允许 ADC_SAR 在 PSoC Creator 中受 DMA 向导工具的支持。
	在 ADC_GetResult8 和 ADC_GetResult16 API 中实现了 ADC 计数到 2 的补码形式的转换。从 ADC_CountsTo_mVolts 函数中删除了相同内容。	进行此更改是为了与 ADC DelSig 一致。