Please note that Cypress is an Infineon Technologies Company.

The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content

The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers

Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

7- to 13-Bit Variable Resolution Incremental ADC Datasheet ADCINCVR V 4.00

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	flash	RAM	
CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52						
	3	0	1	325	5	1

See AN2239, ADC Selection Guide for other converters.

For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

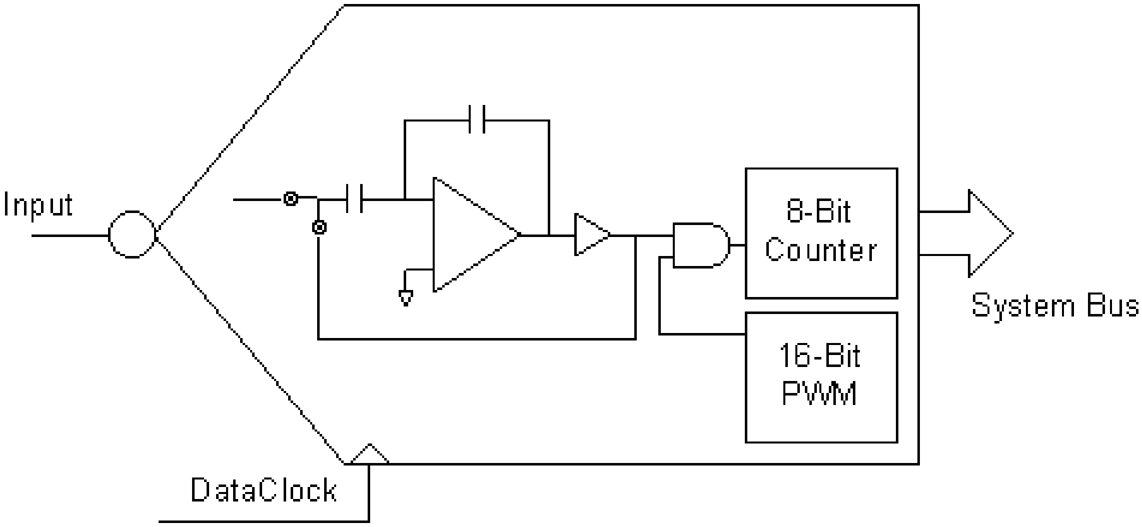
- 7- to 13-bit resolution, 2's complement
- Sample rate from 4 to 5018 sps
- Input range Vss to Vdd
- Integrating converter provides good normal-mode rejection
- Internal or external clock

The ADCINCVR is an integrating ADC with an adjustable resolution between 7 and 13 bits. It can be configured to remove unwanted high frequencies by optimizing the integrate time. Input voltage ranges, including rail-to-rail, may be measured by configuring the proper reference voltage and analog ground. The output is 2's complement based on an input voltage between $–V_{ref}$ and $+V_{ref}$ centered at AGND.

Sample rates from 4 to 5018 sps are achievable depending on the selection of the resolution, DataClock, and CalcTime parameters.

The programming interface allows you to specify the number of sequential samples to be taken or to select continuous sampling. The CPU load varies with the input level. For example, when $V_{in} = +V_{ref}$, there are 5076 CPU cycles (maximum 13 bit). When V_{in} = AGND, there are 2708 CPU cycles (average 13 bit). When $V_{in} = -V_{ref}$, there are 340 CPU cycles (minimum 7-13 bit).

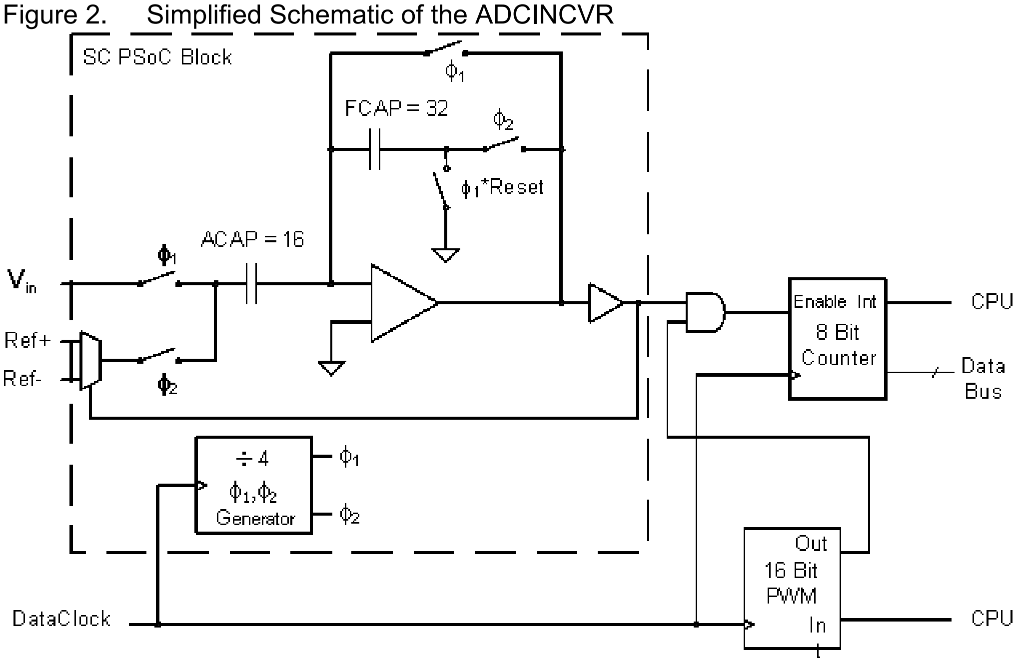
Figure 1. ADCINCVR Block Diagram



Functional Description

The ADCINCVR is formed from a single analog switched capacitor PSoC block and three digital PSoC blocks, as shown in Figure 2.

Figure 2. Simplified Schematic of the ADCINCVR



The analog block is configured as a integrator that can be reset. Depending on the output polarity, the reference control is configured so that the reference voltage is either added or subtracted from the input and placed in the integrator. This reference control attempts to pull the integrator output back towards AGND. If the integrator is operated 2^{Bits} times and the output voltage comparator is positive “n" of those times, the residual voltage (V_{resid}) at the output is calculated using Equation 1:

Equation 1

$$V_{resid} = 2^{Bits} \cdot V_{in} - (n \cdot V_{ref}) + (2^{Bits} - n) \cdot V_{ref}$$

Equation 2

$$V_{in} = \frac{n - 2^{Bits-1}}{2^{Bits-1}} V_{ref} + \frac{V_{resid}}{2^{Bits}}$$

This equation states that the range of this ADC is $\pm V_{ref}$, the resolution (LSB) is $V_{ref}/2^{Bits-1}$, and the voltage on the output at the end of the computation is defined as the residue. Since V_{resid} is always less than V_{ref}, $V_{resid}/2^{Bits}$ is less than half a LSB and can be ignored. The resulting equation is Equation 3:

Equation 3

$$V_{in} = \frac{n - 2^{Bits-1}}{2^{Bits-1}} V_{ref}$$

Example 1

For a V_{ref} of 1.3V and a resolution of 8-bits, you can easily calculate the input voltage based on the value read from the incremental ADC at the time the data is ready. This calculation is done using Equation 4:

Equation 4

$$V_{in} = \frac{n - 128}{128} 1.3$$

The result of the calculation is referenced to AGND. For a ADC data value of 200, the Voltage measured is calculated to be 0.73V using Equation 5:

Equation 5

$$V_{in} = \frac{200 - 128}{128} 1.3 = 0.73\,V$$

The value calculated is an ideal value and may differ based on system noise and chips offsets.

To determine the expected code given a specific input voltage, the equation can be rearranged to give Equation 6:

Equation 6

$$n = \frac{2^{Bits-1} \cdot V_{in}}{V_{ref}} + 2^{Bits-1}$$

Example 2

For a V_{ref} of 1.3V and a resolution of 8-bits, you can easily calculate the expected ADC code based on the input Voltage. The calculation is done using Equation 7:

Equation 7

$$n = \frac{128 \cdot V_{in}}{1.3} + 128$$

For an input voltage of -1V below AGND the code from the ADC can be expected to be 29.53. This is based on Equation 8:

Equation 8

$$n = \frac{128 \cdot (-1)}{1.3} + 128 = 29.53$$

The value calculated is an ideal value and may differ based on system noise and chips offsets.

To make the integrator function as an incremental ADC, the following digital resources are used:

- An 8-bit counter to accumulate the number of cycles that the output is positive.
- A 16-bit PWM to measure the integrate time and gate the clock into the 8-bit counter.

A single DataClock is connected to the 8-bit counter, the 16-bit PWM, and the analog column clock which connects to the analog SC PSoC block. The analog column clock is actually two clocks, φ_1 and φ_2, which are generated from the DataClock. These two additional clocks are exactly one-fourth the frequency of the DataClock. This means that the PWM and counter operate four times faster than required and therefore need to accumulate N+2 bits worth of data (N equal number of bits of resolution).

Note **CAUTION:** It is imperative, when placing this module, that you configure it with the same clock for all three blocks. Failure to do so causes it to operate incorrectly.

The counter is implemented with an 8-bit digital block for the LSB and a software counter for the MSB. Each time the hardware counter overflows, an interrupt is generated and the upper MSB of the counter is incremented. This allows the ADCINCVR module to be implemented with only three digital blocks instead of four.

The sample rate is the DataClock divided by the integrate time plus the time it takes to do the result calculations (CalcTime). The integrate time is the period when the input signal is being sampled by the ADCINCVR.

Equation 9

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

The time it takes to calculate the result, CalcTime, varies inversely proportional with the CPU clock. The CalcTime must be set to a value greater than what is required to calculate the result. The minimum CalcTime is equivalent to 180 CPU cycles and must be expressed in terms of the DataClock. You can also increase the CalcTime beyond the minimum to optimize the sample rate.

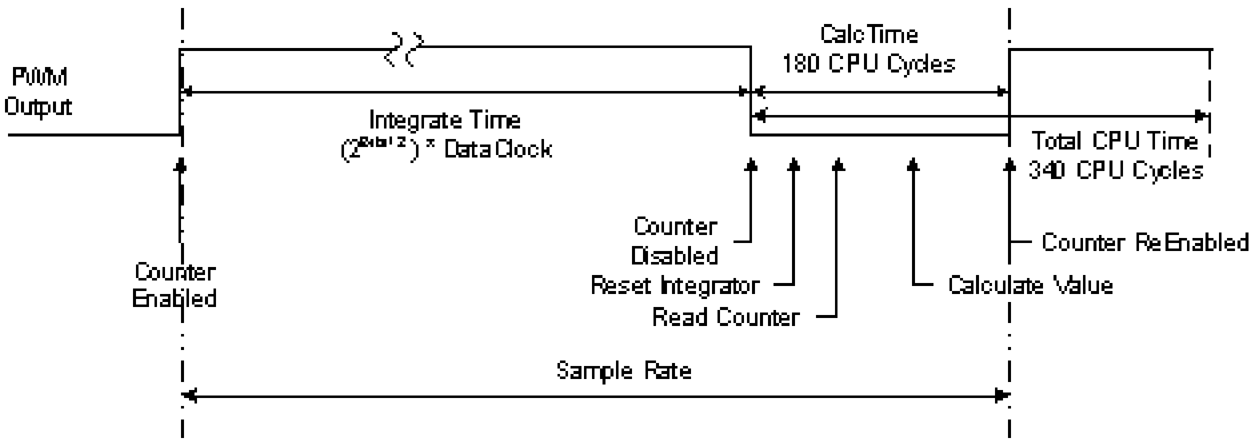
Note The total of 2^{Bits+2} plus the CalcTime must not exceed 2^{16}-1 or 65,535.

Equation 10

$$CalcTime \geq \frac{180 \cdot DataClock}{CPU\_Clock}$$

The 16-bit PWM is programmed to output a high signal that is 2^{Bits+2} times the DataClock. For example, if the resolution is set to 10 bits, the PWM output remains high for 4096 (2^{10+2}) DataClock periods. The PWM output is low for the time it takes to do the minimum result calculations and to reset the integrator. This low time can also be adjusted to help provide a more exact sample rate in combination with the DataClock. The total period of the PWM is the sum of the integrate time and the CalcTime.

Figure 3. Timer Cycles for the ADCINCVR with Respect to PWM Output



When the first reading is initiated, the PWM configuration is calculated, the integrator is reset, and the counter is reset to FFh. The initial delay is always at least that of the calculation time. The PWM is initialized only before the first reading. After the Compare and Period registers are set once, they do not have to be reinitialized unless resolution or calculation time is changed. When the PWM count is less than or equal to the integrate value, the output goes high, enabling the 8-bit counter to count down. The output of the PWM stays high until the counter reaches zero. At this point, the clock to the 8-bit counter is disabled and the PWM interrupt is generated.

The initial value of this 8-bit software counter is set to 2^{Bits}/64 times the most negative value. Each time the 8-bit counter overflows, the interrupt for the 8-bit counter is executed and/or the software counter is incremented by one.

When the input to the ADC is greater than or equal to the most positive value, the 8-bit counter increments on every positive transition of the DataClock. If the input to the ADC is less than or equal to the most negative input value, the 8-bit counter never decrements and therefore, never generates an interrupt. An input near analog ground under ideal conditions allows the counter to increment half the time. It is easy to see that, depending on the input voltage level, the amount of interrupts from the 8-bit counter varies from 0 to (2^{Bits+2})/256. For example, if the resolution is set to 10 bits, the PWM compare value is set to 2^{10+2} (4096). This means that it is possible that the processor could be interrupted a maximum of 4096/256 or 16 times during the integrate period.

Due to the ADCINCVR control being interrupt based and the length of the time period for a high resolution result, it is unreasonable to expect the processor to wait while a sample is processed. The primary communication between the ADC routine and the main program is a flag that may be polled using an API function, ADCINCVR_IsDataAvailable(). When a value is returned, the API ADCINCVR_iGetData() can be called to retrieve the data.

The data handler was designed to be poll based. If an interrupt based data handler is desired, you can insert your own data handler code into the interrupt routine ADCINCVR_PWM16_ISR, located in the assembly file *adcincvrINT.asm*. The point to best insert code is clearly marked.

CPU Usage

The ADCINCVR requires CPU time to calculate the result and to increment the software counter each time the hardware counter overflows. The CPU overhead is dependent on three variables: CPU clock, DataClock, and input voltage. At first it may seem odd that input voltage affects the CPU overhead for an ADC. Input voltages that are near or lower than $-V_{ref}$ require very little CPU overhead. Input voltages that are near or greater than $+V_{ref}$ require the most CPU overhead. To calculate the CPU cycles required for a given input:

Equation 11

$$\text{CPU\_Cycles} = \text{PWM\_IRQ\_CPU\_Cycles} + \left(\frac{2^{Bits}}{64}\left(\frac{V_{ref} + V_{in}}{2 \cdot V_{ref}}\right)\text{Counter\_IRQ\_CPU\_Cycles}\right)$$

Equation 12

$$CPUcycles = 340 + \left(\frac{2^{Bits}}{64} * \left(\frac{Vref + Vin}{2 * Vref}\right) * 37\right)$$

To calculate the maximum CPU cycles at 10-bits resolution, set V_{in} to V_{ref}:

Equation 13

$$CPUcycles = 340 + \left(\frac{2^{10}}{64} * \left(\frac{Vref + Vref}{2 * Vref}\right) * 37\right) = 340 + (16 * 1 * 37) = 932$$

To calculate the percent CPU usage of the ADCINCVR, use Equation 14:

Equation 14

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

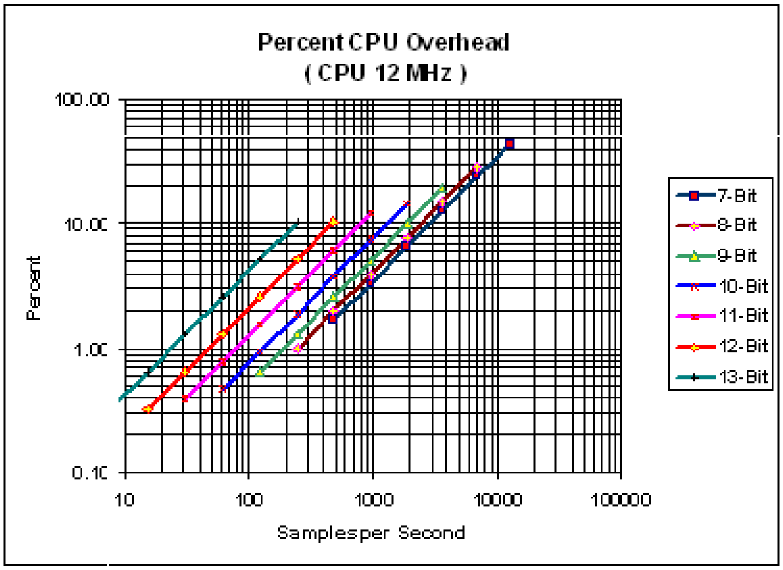
Setting the resolution to 10 bits (as in the previous example), sample rate to 1000 sps, and the CPU clock to 12 MHz, in the equation below, shows that less than eight percent of the CPU is used.

Equation 15

$$Percent\_CPU\_Utilization = \frac{1000 * 932}{12MHz} * 100 = 7.8\%$$

Figure 4 shows CPU usage for the supported sample rates and resolutions. The default CPU speed is set to 12 MHz.

Figure 4. CPU Usage



Frequency Rejection

By selecting the proper integrate time, some noise sources may be rejected. To reject a noise source and its harmonics, select an integrate time that is equal to an integral cycle of the noise signal. If more than one signal is to be rejected, select an integrate time that is equal to an integral cycle of both signals.

For example, to reject noise caused by 50 Hz and 60 Hz signals, select a period that contains an integral number of both the 50 Hz and 60 Hz signals.

An IntegrateTime of 100 ms rejects both 50 Hz and 60 Hz, and any harmonics of these signals. Next, calculate the DataClock required to generate the proper IntegrateTime.

Note that the CalcTime is not used in this calculation, although it affects the sample rate. The IntegrateTime is the period when the ADCINCVR is actually sampling the input voltage. The sample rate is based on the IntegrateTime and the time it takes to calculate the result.

Example

An IntegrateTime of 100 ms and an A/D resolution of 13 bits are required for a given application. For a 100 ms IntegrateTime, the data clock must be:

Equation 16

$$DataClock = \frac{2^{Bits+2}}{IntegrateTime} = \frac{2^{13+2}}{100ms} = 327.7kHz$$

The CalcTime in terms of the data clock must be calculated from the DataClock and the CPU clock. If the CPU clock is 12 MHz, the minimum calculation time is:

Equation 17

$$CalcTime = \frac{DataClock * 180}{CPUClock} = \frac{327.7kHz * 180}{12{,}000\,kHz} = 4.9\,DataClocks$$

This CalcTime should be rounded up to the nearest whole number, which is '5' in this example. Now determine the sample rate using Equation 18:

Equation 18

$$SampleRate = \frac{DataClock}{2^{13+2} + CalcTime} = \frac{327.7kHz}{32768 + 5} = 9.99Samples/Second$$

If a longer sample rate is desired, the CalcTime may be increased until the CalcTime + 2^{13+2} is less than or equal to 2^{16}-1 (65535).

DC and AC Electrical Characteristics

The following values are indicative of expected performance and based on initial characterization data. Unless otherwise specified in the table below, TA = 25°C, Vdd= 5.0V, Power HIGH, Op-Amp bias LOW, output referenced to 2.5 V external Analog Ground on P2[4] with 1.25 external Vref on P2[6] and resolution set at 13 bits.

Table 1. 5.0V ADCINCVR DC and AC Electrical Characteristics, CY8C29/27/24/22xxxFamily of PSoC Devices

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range[a]	–	Vss to Vdd		Ref Mux = Vdd/2 ± Vdd/2
Input Capacitance[b]	3	–	pF	
Input Impedance[c, d]	1/(C*clk)	–	W	
Resolution	–	7 to 13	Bits	
Sample Rate	–	4 to 5018	SPS	
SNR	77	–	dB	
DC Accuracy				
DNL	0.4	–	LSB	Column clock 2 MHz
INL	1.0	–	LSB	
Offset Error	9	–	mV	
Gain Error				
Including Reference Gain Error	2.0	–	% FSR	
Excluding Reference Gain Error[e]	0.1	–	% FSR	
Operating Current				

Parameter	Typical	Limit	Units	Conditions and Notes
Low Power	250	–	µA	
Med Power	640	–	µA	
High Power	2000	–	µA	
Data Clock	–	0.125 to 8	MHz	Input to digital blocks and analog column clock

a. Input voltages above the maximum generate a maximum positive reading. Input voltages below the minimum generate a maximum negative reading.

b. Includes I/O pin.

c. The input capacitance or impedance is only applicable when input to the analog block is directly to a pin.

d. C = input Capacitance, clk = Data Clock (Analog Column Clock).

e. Reference Gain Error measured by comparing the external reference to VRefHigh and VRefLow routed through the test mux and back out to a pin.

The following values are indicative of expected performance and based on initial characterization data. Unless otherwise specified in the table below, TA =25°C, Vdd= 3.3V, Power HIGH, Op-Amp bias LOW, output referenced to 1.64V external Analog Ground on P2[4] with 1.25 external Vref on P2[6], and resolution set at 13 bits.

Table 2. 3.3V ADCINCVR DC and AC Electrical Characteristics, CY8C29/27/24/22xxxFamily of PSoC Devices

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range[a]	–	Vss to Vdd		Ref Mux = Vdd/2 ± Vdd/2
Input Capacitance[b]	3	–	pF	
Input Impedance[c, d]	1/(C*clk)	–	W	
Resolution	–	7 to 13	Bits	
Sample Rate	–	4 to 5018	SPS	
SNR	77	–	dB	
DC Accuracy				
DNL	0.4	–	LSB	Column clock 2 MHz
INL	1.0	–	LSB	
Offset Error	4	–	mV	
Gain Error				
Including Reference Gain Error	2.0	–	% FSR	
Excluding Reference Gain Error[e]	0.4	–	% FSR	
Operating Current				

Parameter	Typical	Limit	Units	Conditions and Notes
Low Power	140	–	µA	
Med Power	490	–	µA	
High Power	1830	–	µA	
Data Clock	–	0.125 to 8	MHz	Input to digital blocks and analog column clock

a. Input voltages above the maximum generate a maximum positive reading. Input voltages below the minimum generate a maximum negative reading.

b. .Includes I/O pin.

c. The input capacitance or impedance is only applicable when input to the analog block is directly to a pin.

d. C = input Capacitance, clk = Data Clock (Analog Column Clock).

e. Reference Gain Error measured by comparing the external reference to VRefHigh and VRefLow routed through the test mux and back out to a pin.

Note Electrical Characteristics:

1. Typical values represent parametric norm at +25°C.
2. Input voltages above the maximum generate a maximum positive reading. Input voltages below the minimum generate a maximum negative reading.
3. User module only, not including I/O pin.
4. The input Capacitance or impedance is only applicable when input to analog block is directly to a pin.
5. C = input Capacitance, clk = Data Clock (Analog Column Clock).
6. Specifications are for sample rates of 100 sps and a data clock of 8 MHz, unless otherwise noted. Sample rate is dependent on both Data Clock and resolution.
7. SNR = Ratio of power of full scale single tone divided by total noise integrated to $F_{sample}/2$.

Placement

The ADC block can be placed in any of the switched capacitor PSoC blocks. It must be able to exclusively drive the comparator bus for the particular column in which it is placed.

The counter block may be placed in any available digital block, but the PWM16 must only be placed in specific locations. See the following table for valid placements:

Table 3. Valid Placements

Part Family	Valid PWM16 Placements LSB/MSB
CY8C27xxx	DBB00/DBB01, DBB01/DCB02, DBB10/DBB11, DBB11/DCB12
CY8C24xxx/CY8C22xxx	DBB00/DBB01, DBB01/DCB02

Both digital blocks have an interrupt service routine. It is desirable, but not required, that the counter block have a higher interrupt priority than the PWM16 block. Therefore, it is recommended that the counter block be placed in a lower digital block position than the PWM16 block.

Note Placing the ADCINCVR uses DEC_CR0[7:4] and DEC_CR1[5:3]. While the decimator is not used by this ADC, the decimator registers are used for gating purposes and therefore limits the ability to place multiple instances of this ADC.

Parameters and Resources

Input

The selection of the input is done after the analog PSoC block is placed. The eight switched capacitor blocks have differing input selections. Each can be connected to most of its neighbors, while some can be directly connected to external input pins. Placement of the analog block must be done with some consideration of how to get an input signal to it. Some placements allow inputs to be routed directly from package pins to the input. These direct connections allow inputs that are within 40 mV of the supply rails to be measured accurately. Signals may also be routed through one of the column muxes, through one of the CT Block test muxes, and onto an analog column where the ADCINCVR can also measure signals near the power supply rails.

ClockPhase

The selection of the Clock Phase is used to synchronize the output of one switched capacitor analog PSoC block to the input of another. The switched cap analog PSoC blocks use a two-phase clock (φ_1, φ_2) to acquire and transfer signal. Typically, the input to the ADCINCVR is sampled on φ_1, the Normal setting. A problem arises in that many of the user modules auto-zero their output during φ_1 and only provide a valid output during φ_2. If such a module's output is fed to the ADCINCVR's input, the ADCINCVR acquires an auto-zeroed output instead of a valid signal. The Clock Phase selection allows the phases to be swapped so that the input signal is now acquired during φ_2, the Swapped setting.

ADCResolution

This selection allows the resolution of the ADCINCVR to be set in the Device Editor. Although there is an API routine to set or change the resolution, it is not required if set in the Device Editor. The resolution can also be changed at anytime with the API call, but the ADC is stopped and must be restarted. Valid resolution settings are 7 to 13 inclusive.

CalcTime

The CalcTime is the amount of time it takes the CPU to calculate intermediate integration result before the next integrate cycle can start. The time it takes to calculate the result "CalcTime" varies inversely proportionally with the CPU clock. This value must be in terms of the DataClock. Minimum CPU calculation time is 180 CPU clocks. CalcTime may also be increased to optimize the sample rate.

Note Ensure that the CalcTime + of 2^{Bits+2} does not exceed 2^{16}-1 or 65,535.

You can use Equation 19 to determine what the CalcTime should be set to:

Equation 19

$$CalcTime \geq \frac{180 \cdot DataClock}{CPU\_Clock}$$

For example, if the DataClock is set to 1.5 MHz and the CPU is running at 6 MHz, the CalcTime should be set to greater than or equal to 45.

Clock and Integrator Column Clock

The Data Clock determines the sample rate and the signal sample window. This clock must be routed to the clock input of the counter block, the 16 bit PWM block, and the column clock for the column containing the integrator.

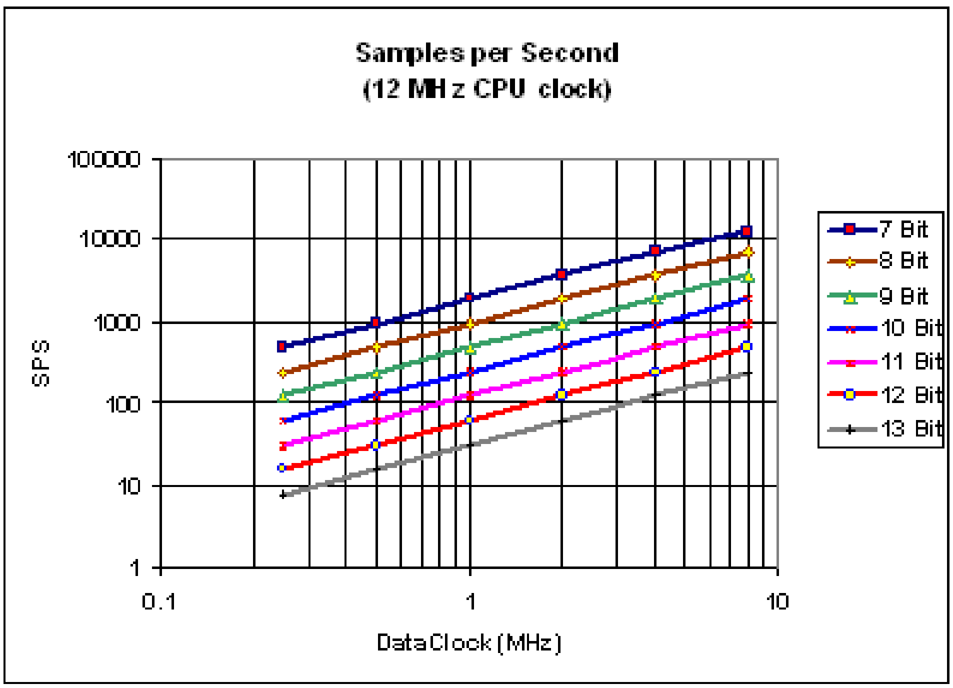
Note The column clock of the integrator switch cap block must be manually set to the SAME clock. It is imperative that the same clock is used for all three blocks or this user module does not function correctly.

This parameter setting only sets the clock to the counter block and the PWM block. This clock may be any source with a clock rate between 125 kHz and 8 MHz.

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

Figure 5 shows possible sample rates for each of the resolution options for the ADCINCVR.

Figure 5. Sample Rates for ADCINCVR



DataFormat

This selection determines in what format the result is returned. If “Signed" is selected and “N" is the selected resolution, the result ranges from 2^{N-1} to 2^{N-1} –1. If “Unsigned" is selected, the result is between 0 and 2^N-1. See the following table for result ranges for each Data Format and resolution.

Table 4. ADCINCVR Data Format and Resolution Result Ranges

Resolution Setting	Signed Data Format	Unsigned Data Format
7	-64 to 63	0 to 127
8	-128 to 127	0 to 255
9	-256 to 255	0 to 511
10	-512 to 511	0 to 1023
11	-1024 to 1023	0 to 2047
12	-2048 to 2047	0 to 4095
13	-4096 to 4095	0 to 8191

Interrupt Generation Control

The following parameter is only available if the **Enable interrupt generation control** check box in PSoC Designer is checked. This is available under **Project > Settings > Chip Editor**. Interrupt Generation Control is important when multiple overlays are used with interrupts shared by multiple user modules across overlays.

IntDispatchMode

The IntDispatchMode parameter is used to specify how an interrupt request is handled for interrupts shared by multiple user modules existing in the same block but in different overlays. Selecting “ActiveStatus" causes firmware to test which overlay is active before servicing the shared interrupt request. This test occurs every time the shared interrupt is requested. This adds latency and also produces a nondeterministic procedure of servicing shared interrupt requests, but does not require any RAM. Selecting “OffsetPreCalc" causes firmware to calculate the source of a shared interrupt request only when an overlay is initially loaded. This calculation decreases interrupt latency and produces a deterministic procedure for servicing shared interrupt requests, but at the expense of a byte of RAM.

Global Resources

The usable input voltage is determined by the selection of the “Ref Mux" option in the “Global Resource" section of the Device Editor. The Ref Mux selection determines the analog ground and the usable range of the input voltage about analog ground. For example, if “Vdd/2 +/-BandGap" is selected, and Vdd = 5 volts, the usable input range is 2.5 ± 1.3 volts (1.2 to 3.8 volts). If “Vdd/2 ± Vdd/2" is selected, then the usable input voltage is the full rail-to-rail supply range. The following table lists the valid ranges for a Vdd of 5V and 3.3V.

Table 5. CY8C29/27/24/22xxxInput Voltage Ranges for Each Ref Mux Setting

RefMux Setting	Vdd = 5V	Vdd = 3.3V
(Vdd/2) ± BandGap	$1.2 < V_{in} < 3.8$	$0.35 < V_{in} < 2.95$
(Vdd/2) ± (Vdd/2)	$0 < V_{in} < 5$	$0 < V_{in} < 3.3$
BandGap ± BandGap	$0 < V_{in} < 2.6$	$0 < V_{in} < 2.6$
(1.6*BandGap) ± (1.6*BandGap)	$0 < V_{in} < 4.16$	NA
(2*BandGap) ± BandGap	$1.3 < V_{in} < 3.9$	NA
(2*BandGap) ± P2[6]	$(2.6 - V_{P2[6]}) < V_{in} < (2.6 + V_{P2[6]})$	NA
P2[4] ± BandGap	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$	$(V_{P2[4]} - 1.3) < V_{in} < (V_{P2[4]} + 1.3)$
P2[4] ± P2[6]	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$

Application Programming Interface

The Application Programming Interface (API) routines are provided as part of the user module to allow the designer to deal with the module at a higher level. This section specifies the interface to each function together with related constants provided by the “include" files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X prior to the call if those values are required after the call. This “registers are volatile" policy was selected for efficiency reasons and has been in force since version 1.0 of PSoC Designer. The C compiler automatically takes care of this requirement. Assembly language programmers must ensure their code observes the policy, too. Though some user module API function may leave A and X unchanged, there is no guarantee they may do so in the future.

For Large Memory Model devices, it is also the caller's responsibility to preserve any value in the CUR_PP, IDX_PP, MVR_PP, and MVW_PP registers. Even though some of these registers may not be modified now, there is no guarantee that will remain the case in future releases.

Entry Points are supplied to initialize the ADC, start it sampling, and stop the ADC. In all cases the “instance name" of the module replaces the “ADCINCVR" prefix shown in the following entry points. Failure to use the correct instance name is a common cause of syntax errors.

ADCINCVR_Start

Description:

Performs all required initialization for this user module and sets the power level for the switched capacitor PSoC block.

C Prototype:

```
void  ADCINCVR_Start(BYTE bPower)
```

Assembly:

```
mov   A, ADCINCVR_HIGHPOWER
lcall  ADCINCVR_Start
```

Parameters:

Power: One byte that specifies the power level. Following reset and configuration, the analog PSoC block assigned to ADCINCVR is powered down. Symbolic names provided in C and assembly, and their associated values are listed in the following table:

Symbolic Name	Value
ADCINCVR_OFF	0
ADCINCVR_LOWPOWER	1
ADCINCVR_MEDPOWER	2
ADCINCVR_HIGHPOWER	3

Power level has an effect on analog performance. The correct power setting is sensitive to the sample rate of the data clock and has to be determined for each application. It is recommended that you start your development with full power selected. Testing can later be done to determine how low you can set the power setting.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

ADCINCVR_SetPower

Description:

Sets the power level for the switched capacitor PSoC block.

C Prototype:

```
void  ADCINCVR_SetPower(BYTE bPower)
```

Assembly:

```
mov   A, [bPower]
lcall  ACDINCVR_SetPower
```

Parameters:

Power: Same as the bPower parameter used for the "Start" API routine, above. Allows you to change the power level while operating the ADC.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

ADCINCVR_SetResolution

Description:

Sets the resolution of the A/D converter.

C Prototype:

```
void  ADCINCVR_SetResolution(BYTE bResolution)
```

Assembly:

```
mov   A, [bResolution]
lcall  ADCINCVR_SetResolution
```

Parameters:

Resolution: The resolution of the A/D converter may be set either in the Device Editor, or in the user firmware. If not set in the firmware, the ADC uses the resolution set in the Device Editor by default. Values for resolution may be set between 7 and 13 bits.

Return Value:

If the ADCINCVR is sampling the input, it is stopped if this function is called.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

ADCINCVR_Stop

Description:

Sets the power level on the switched capacitor integrator block to 0ff. This is done when the ADCINCVR in not being used and the user wants to save power. This routine powers down the analog switch capacitor block and disables the digital blocks. To achieve the lowest power level, the clock should be removed from the digital blocks as well.

C Prototype:

```
void  ADCINCVR_Stop(void)
```

Assembly:

```
lcall  ACDINCVR_Stop
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

ADCINCVR_GetSamples

Description:

Initializes and starts the ADC algorithm to collect the specified number of samples. Remember to enable global interrupts by calling the M8C_EnableGInt macro call in *M8C.inc* or *M8C.h*.

C Prototype:

```
void  ADCINCVR_GetSamples(BYTE bNumSamples)
```

Assembly:

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

Parameters:

NumSamples: An 8-bit value that sets the number of samples to be retrieved. A value of ‘0‘ causes the ADC to run continuously.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

ADCINCVR_StopAD

Description:

Stops the ADC immediately.

C Prototype:

```
void  ADCINCVR_StopAD(void)
```

Assembly:

```
lcall  ADCINCVR_StopAD
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

ADCINCVR_fIsData, ADCINCVR_fIsDataAvailable

Description:

Returns non-zero when a data conversion has been completed and data is available for reading.

C Prototype:

```
CHAR  ADCINCVR_fIsDataAvailable(void)
CHAR  ADCINCVR_fIsData(void)
```

Assembly:

```
lcall  ADCINCVR_fIsDataAvailable
```

Parameters:

None

Return Value:

Returns non-zero when data is available.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

ADCINCVR_iGetData

Description:

Returns last converted data. ADCINCVR_fIsDataAvailable() should be called before getting the data, to ensure that the data is valid. Data must be retrieved before the next conversion cycle is completed or else the data is overwritten. There is a possibility that the returned data is corrupted if the call to this function is done exactly at the end of an integration period. It is therefore highly recommended that the data retrieval be done at a higher frequency than the sampling rate, or if that cannot be guaranteed that interrupts be turned off before calling this function.

C Prototype:

```
INT   ADCINCVR_iGetData(void)
```

Assembly:

```
lcall  ADCINCVR_iGetData
```

Parameters:

None

Return Value:

Conversion result is returned. In assembler, the MSB is returned in X and the LSB in the Accumulator.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

ADCINCVR_ClearFlag

Description:

Clears Data Available flag. This function should be called after data is read.

C Prototype:

```
void  ADCINCVR_ClearFlag(void)
```

Assembly:

```
lcall  ADCINCVR_ClearFlag
```

Parameters:

None

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxx). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

ADCINCVR_iGetDataClearFlag

Description:

Returns last conversion data and clears the Data Available flag. ADCINCVR_fIsDataAvailable() should be called before getting the data, to ensure that the data is valid. Data must be retrieved before the next conversion cycle is completed or else the data is overwritten. There is a possibility that the returned data is corrupted if the call to this function is done exactly at the end of an integration period. It is therefore highly recommended that the data retrieval be done at a higher frequency than the sampling rate, or if that cannot be guaranteed that interrupts be turned off before calling this function.

C Prototype:

```
INT   ADCINCVR_iGetDataClearFlag(void)
```

Assembly:

```
lcall  ADCINCVR_iGetDataClearFlag
```

Parameters:

None

Return Value:

Conversion result is returned. In assembler, the MSB is returned in X and the LSB in the Accumulator.

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (CY8C29xxxc). When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently, only the CUR_PP page pointer register is modified.

Sample Firmware Source Code

This sample code starts a continuous conversion, polls the data available flag, and sends the converted byte to a user function.

```
;;;  Sample Code for the ADCINCVR
;;;  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, 10                       ;Set resolution to 10 Bits
    call  ADCINCVR_SetResolution

    mov   a, ADCINCVR_HIGHPOWER       ;Set Power and Enable A/D
    call  ADCINCVR_Start

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

;A/D conversion loop
loop1:

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

    call  ADCINCVR_ClearFlag          ;Reset flag
    call  ADCINCVR_iGetData           ;Get Data - X=MSB A=LSB
    ; Place user code here

    jmp   loop1
```

The same project written in C.

```
//--------------------------------------------------------------------
//  Sample C Code for the ADCINCVR
//  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
```

```
    ADCINCVR_Start(ADCINCVR_HIGHPOWER); // Turn on Analog section
    ADCINCVR_SetResolution(10);         // Set resolution to 10 Bits
    ADCINCVR_GetSamples(0);             // Start ADC to read continuously
    for(;;)
    {
        while(ADCINCVR_fIsDataAvailable() == 0); // Wait for data to
                                                  // be ready.
        iData = ADCINCVR_iGetData();    // Get Data
        ADCINCVR_ClearFlag();           // Clear data ready flag
       // Place user code here
    }
}
```

Configuration Registers

These registers are configured by the initialization and API library. You do not have to change or read these registers directly. This section is given as a reference.

The ADC is a switched capacitor PSoC block. It is configured to make an analog modulator. To build the modulator, the block is configured to be an integrator with reference feedback that converts the input value into a digital pulse stream. The input multiplexer determines what signal is digitized.

Table 6. Block ADC: Register CR0

Bit	7	6	5	4	3	2	1	0
Value	1	0	0	1	0	0	0	0

Table 7. Block ADC: Register CR1

Bit	7	6	5	4	3	2	1	0
Value	ACMux, AMux			0	0	0	0	0

ACMux is used when the block is placed in a type “A" block. Field value depends on how you connect the input. AMux is used when the block is placed in a type “B" block. Field value depends on how you connect the input.

Table 8. Block ADC: Register CR2

Bit	7	6	5	4	3	2	1	0
Value	0	1	1	0	0	0	0	0

Table 9. Block ADC: Register CR3

Bit	7	6	5	4	3	2	1	0
Value	1	1	1	FSW0	0	0	0	0

FSW0 is used by the PWM16 interrupt handler and various APIs. A ‘0‘ value causes ADC to be a disabled integrator. A ‘1‘ value causes ADC to be an enabled integrator.

The PWM16 is a digital PsoC block that is used to control the integration time of the ADC. The compare value is set to 2^{Bits+2} and the period is set to the CalcTime plus the compare value.

Table 10. Block PWM16_MSB: Register Function

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	Compare Type	Interrupt Type	0	0	1

Compare Type is a flag that indicates whether the capture comparison is “equal to or less than" or “less than." Interrupt Type is a flag that indicates whether to trigger the interrupt on the capture event or the terminal condition. Both parameters are set in the Device Editor.

Table 11. Block PWM16_LSB: Register Function

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	Compare Type	0	0	0	1

Compare Type is a flag that indicates whether the compare function is set to “equal to or less than" or “less than." This parameter is set in the Device Editor.

Table 12. Block PWM16_MSB: Register Input

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	1	Clock			

Clock selects the clock input from one of 16 sources. This parameter is set in the Device Editor.

Table 13. Block PWM16_LSB: Register Input

Bit	7	6	5	4	3	2	1	0
Value	Enable				Clock			

Enable selects the data input from one of 16 sources and Clock selects the clock input from one of 16 sources. Both parameters are set in the Device Editor.

Table 14. Block PWM16_MSB: Register Output

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	Output Enable	Output Sel	

Output Enable is the flag that indicates the output is enabled. Output Sel is the flag that indicates where the output of the PWM16 will be routed. Both parameters are set in the Device Editor.

Table 15. Block PWM16_LSB: Register Output

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 16. Block PWM16_MSB: Count Register DR0

Bit	7	6	5	4	3	2	1	0
Value	Count(MSB)							

Count: PWM16 MSB down PWM. It can be read using the PWM16 API.

Table 17. Block PWM16_LSB: Count Register DR0

Bit	7	6	5	4	3	2	1	0
Value	Count(LSB)							

Count: PWM16 LSB down PWM. It can be read using the PWM16 API.

Table 18. Block PWM16_MSB: Period Register DR1

Bit	7	6	5	4	3	2	1	0
Value	Period(MSB)							

Period holds the MSB of the period value that is loaded into the Counter register, upon enable or terminal count condition. It can be set by the Device Editor and the PWM16 API.

Table 19. Block PWM16_LSB: Period Register DR1

Bit	7	6	5	4	3	2	1	0
Value	Period(LSB)							

Period holds the LSB of the period value that is loaded into the Counter register upon enable or terminal count condition. It can be set by the Device Editor and the PWM16 API.

Table 20. Block PWM16_MSB: Pulse Width Register DR2

Bit	7	6	5	4	3	2	1	0
Value	Pulse Width(MSB)							

PulseWidth holds the MSB of the pulse width value used to generate the compare event. It can be set by the Device Editor and the PWM16 API.

Table 21. Block PWM16_LSB: Pulse Width Register DR2

Bit	7	6	5	4	3	2	1	0
Value	Pulse Width(LSB)							

PulseWidth holds the LSB of the pulse width value used to generate the compare event. It can be set by the Device Editor and the PWM16 API.

Table 22. Block PWM16_MSB: Control Register CR0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Start/ Stop(0)

Start/Stop is controlled by the LSB Control register value, set to zero.

Table 23. Block PWM16_LSB: Control Register CR0

Bit	7	6	5	4	3	2	1	0
Value								Start/ Stop

Start/Stop, when set, indicates that the PWM16 is enabled. It is modified by using the PWM16 API

The CNT is a digital PSoC block configured as a counter. When the value in DR0 counts down to terminal count, an interrupt is called to decrement a higher value software counter and CNT reloads from DR1. The data is outputted through DR2.

Table 24. Block CNT: Register Function

Bit	7	6	5	4	3	2	1	0
Value	0	0	1	0	0	0	0	1

Table 25. Block CNT: Register Input

Bit	7	6	5	4	3	2	1	0
Value	Data				Clock			

Data selects the column comparator where the ADC block has been placed. Clock selects clock input from one of 16 sources and is set in the Device Editor.

Table 26. Block CNT: Register Output

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0

Table 27. Block CNT: Register DR0

Bit	7	6	5	4	3	2	1	0
Value	Count Value							

Table 28. Block CNT: Register DR1

Bit	7	6	5	4	3	2	1	0
Value	1	1	1	1	1	1	1	1

Table 29. Block CNT: Register DR2

Bit	7	6	5	4	3	2	1	0
Value	Data Out							

Data Out is used by the API to get the counter value.

Table 30. Block CNT: Register CR0

Bit	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	Enable

When Enable is set, CNT is enabled. It is modified and controlled by the ADCINCVR API

Table 31. Register INT_MSK1

Bit	7	6	5	4	3	2	1	0
Value								

The mask bits corresponding to the TMR block and CNT block are set here to enable their respective interrupts. The actual mask values are determined by the placement position of each block.

Version History

Version	Originator	Description
3.2	DHA	Added DRC to check if: 1. The source clock is different between digital and analog resources. 2. The ADC Clock is higher than CPU Clock. Added DRC warning when two ADCINC14 or ADCINCVR blocks are added into a project.
4.00	DHA	1. Updated to address issues seen when CalcTime is much higher than calculated. 2. Restored VC3 as a source for the data clock.
4.00.b	HPHA	Added design rules check for the situation when the ADC clock is faster than 8 MHz.

Note PSoC Designer 5.1 introduces a Version History in all user module datasheets. This section documents high level descriptions of the differences between the current and previous user module versions.