Triple Input 8-Bit Incremental ADC Datasheet TriADC8 V 1.10

Copyright © 2004-2015 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O and ADC Input)
	Digital	Analog CT	Analog SC	flash	RAM	
CY8C29/27xxx, CY8CLED08/16, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52	5	0	3	431	10	1

See AN2239, ADC Selection Guide for other converters.

Features and Overview

- Samples three inputs simultaneously
- 8-bit resolution, two's complement or unsigned results
- Sample rates from 4 to greater than 10,000 sps
- Maximum input range Vss to Vdd
- Integrating converter provides good normal mode rejection
- Internal or external clock

The TriADC8 is a triple input integrating ADC with 8-bits of resolution. 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 configurable two's complement or unsigned integers based on an input voltage between –Vref and +Vref centered at AGND.

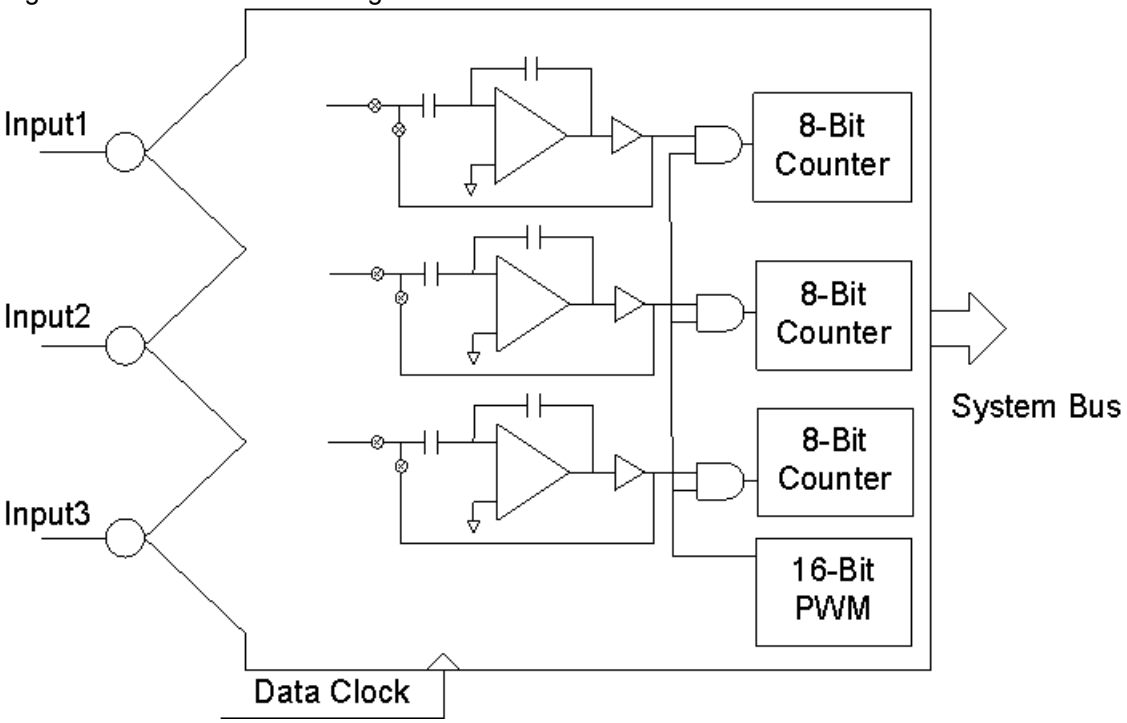
Sample rates up 5000 sps are achievable, depending on the selection of the Data Clock, and CalcTime parameters. Please refer to the triADC8_SetCalcTime API for setting the CalcTime parameter.

The programming interface allows the user to specify the number of sequential samples to be converted or to select continuous sampling. The three input channels are sampled at the same time and duration since they are controlled by a common signal. Power settings, resolution, and speed are common to all three input channels.

The TriADC8 is ideal for applications that require simultaneous sampling of three signals, such as a three phase voltage measurement. As with other PSoC ADCs, signals to both inputs may be multiplexed. Note that you need to review the Parameters section before module placement.

Note When initially selecting the TriADC8, a warning may appear that states “Resource allocation prevents placement." This warning is displayed if the original placement has two ADC blocks in the same column. Simply move each ADC block to its own column.

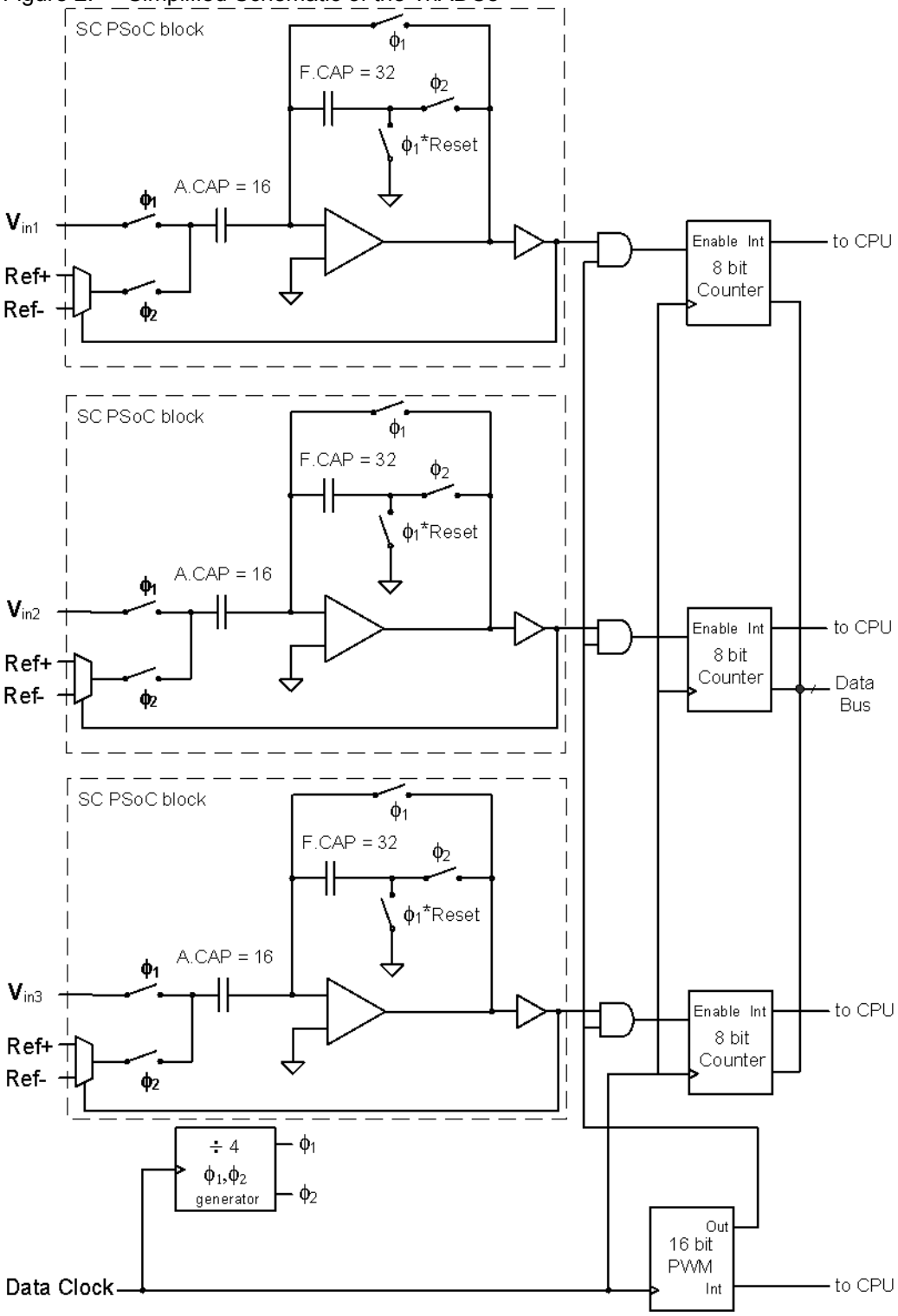
Figure 1. TriADC8 Block Diagram



Functional Description

The TriADC8 is three incrementing ADCs in a single user module. The 16-bit sample rate timer (PWM) is shared to reduce the required digital blocks. Since all three ADCs use the same timer, the sampling is fully synchronized. Five digital PSoC blocks and three analog switch cap PSoC blocks are required, as shown in the following figure.

Figure 2. Simplified Schematic of the TriADC8



The three analog blocks are configured identically as resettable integrators. 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^{56} times and the output voltage comparator is positive “n" of those times, the residual voltage (V_{resid}) at the output is:

Equation 1

$$V_{resid} = 256 \cdot V_{in} - (n \cdot V_{ref}) + (256 - n) \cdot V_{ref}$$

Equation 2

$$V_{in} = \frac{n - 128}{128} V_{ref} + \frac{V_{resid}}{256}$$

This equation states that the range of this ADC is ± V_{ref}, the resolution (LSb) is $V_{ref}/128$, 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}, $V_{resid}/256$ is less than half an LSb and can be ignored. The resulting equation is:

Equation 3

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

Example 1

For a V_{ref} of 1.3V we can easily calculate the input voltage based on the value read from the incremental ADC at the time the data is ready. The equation which can be used is:

Equation 4

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

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

Equation 5

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

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

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

Equation 6

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

Example 2

For a V_{ref} of 1.3V the expected ADC code can be easily calculated based on the input Voltage. The equation which can be used is:

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 based on this calculation:

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 (one per channel).
- A 16-bit PWM to time the integrate time and gate the clock into the 8-bit counter (shared between all three channels).

A single DataClock is connected to the 8-bit counters, the 16-bit PWM, and the analog column clocks which connect to the analog SC PSoC blocks. 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 10 bits worth of data.

Note 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 counters are 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 TriADC8 module to be implemented with only five digital blocks instead of eight.

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 TriADC8.

Equation 9

$$SampleRate = \frac{DataClock}{1024 + CalcTime}$$

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 371 CPU cycles and must be expressed in terms of the DataClock. The CalcTime may also be increased beyond the minimum to optimize the sample rate.

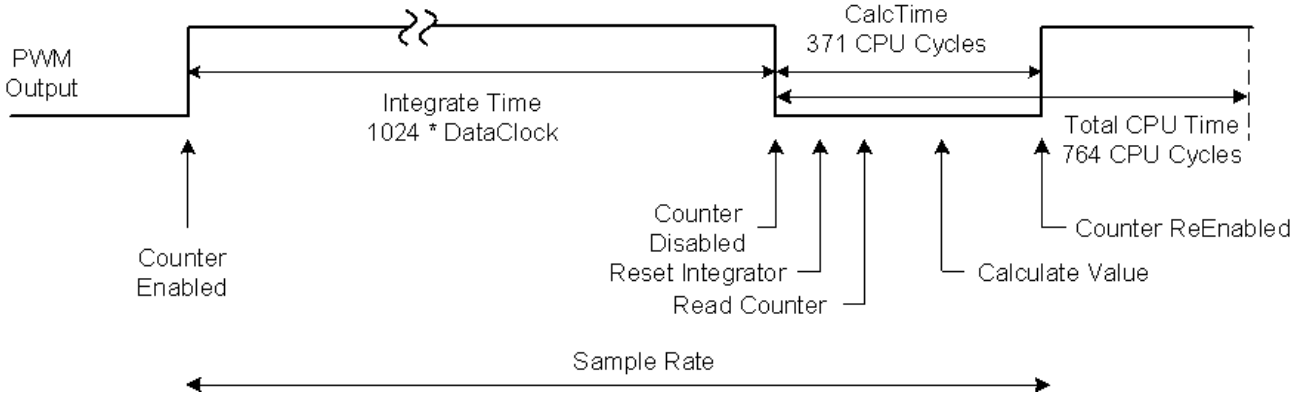
Note The total of 2^{10} plus the CalcTime must not exceed 2^{16}-1 or 65,535.

Equation 10

$$CalcTime \geq \frac{371 \cdot DataClock}{CPU_Clock}$$

The 16 bit PWM is programmed to output a high signal that is 1024 times the DataClock. The PWM output is low for the time it takes to do the minimum result calculations and to reset the integrator. This period is controlled by the CalcTime parameter. The CalcTime 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. TriADC8 Timing with Respect to PWM Output



When the first reading is initiated, the PWM configuration is calculated, the integrator is reset, and the counters are all 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 re-initialized unless the 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 counters to count down. The output of the PWM stays high until the counter reaches zero. At this point, the clock to the 8-bit counters is disabled and the PWM interrupt is generated.

The initial value of the 8-bit software counters is set to 2^{56}/64 times the most negative value. Each time the 8-bit counters overflow, the interrupt for the 8-bit counter is executed and 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 counters increment 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 counters never decrement and therefore, never generate 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 counters vary from 0 to 4.

With the TriADC8 control interrupt based and the sample time so long for a higher resolution result, it is unreasonable to expect the processor to wait while a sample is being processed. The primary communication between the ADC routine and the main program is a flag that may be polled. When the most significant bit of TriADC8_bfStatus has a non zero value, the new data is available in TriADC8_cResultn (n=1,2,3). APIs are available to check the data flag and retrieve data.

This data handler was designed to be poll based. If an interrupt based data handler is desired, the user may insert his or her own data handler code into the interrupt routine TriADC8CNTn_INT, located in the assembly file TriADC8*INT.asm*. The point to best insert code is clearly marked.

Channel to Channel Differences

When using the TriADC8, there are differences between channels when measuring the same input voltage. This difference is due to the input offset variations in the switch cap block amplifiers and the column AGND buffers. This channel to channel offset is easily compensated for by routing the same signal into each of the ADC channels. One of the channels can be used as the reference and the difference between subsequent channels would be subtracted after each reading.

CPU Use

The TriADC8 requires CPU time to calculate the result and to increment the software counters each time the hardware counters overflow. 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 -Vref require very little CPU overhead. Input voltages that are near or greater than +Vref require more CPU overhead. The following equations assume the input signal is the same for all three inputs. To calculate the CPU cycles required for a given input:

Equation 11

$$\text{Jcycles} = \text{PWM16_IRQ_Cycles} + \left(4\frac{Vref + Vin}{2 \cdot Vref} \cdot (\text{Counter_IRQ_CPUcycles} \cdot \right.$$

Equation 12

$$\text{CPUcycles} = 764 + \left(4\frac{Vref + Vin}{2 \cdot Vref} \cdot (37 \cdot 3)\right)$$

To calculate the maximum CPU cycles, set Vin to Vref.

Equation 13

$$\text{CPUcycles} = 764 + \left(4\frac{Vref + Vref}{2 \cdot Vref} \cdot 111\right) = 764 + (4 \cdot 1 \cdot 111) = 1208$$

To calculate the percent CPU use of the TriADC8, the following equation can be used:

Equation 14

$$\text{Percent_CPU_Utilization} = \frac{\text{SampleRate} \cdot CPUcycles}{CPUfrequency} \times 100$$

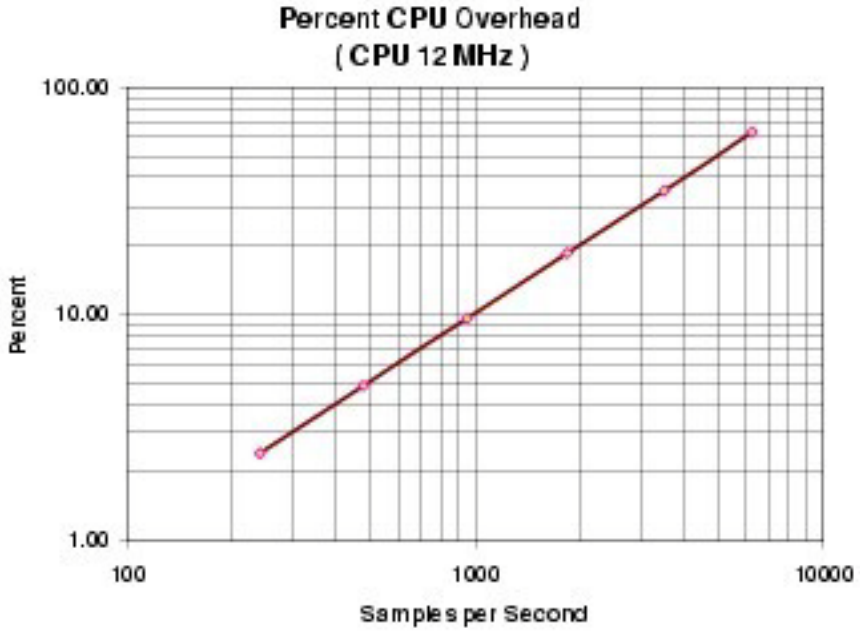
Setting the sample rate to 1000 samples/sec, and the CPU clock to 12 MHz, then (in the equation below) about 21 percent of the CPU is used.

Equation 15

$$\text{Percent_CPU_Utilization} = \frac{1000 \cdot 1208}{12\ \text{MHz}} \times 100 = 10\%$$

The graph below shows CPU use for the supported sample rates. The default CPU speed is set to 12 MHz.

Figure 4. CPU Use for Supported Sample Rates



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, if noise caused by 50 Hz and 60 Hz signals is to be rejected, select a period that contains an integral number of both the 50 Hz and 60 Hz signals.

Equation 16

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

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.

Equation 17

$$DataClock = \frac{1024}{IntegrateTime}$$

Notice that the CalcTime is not used in this calculation, although it affects the sample rate. The IntegrateTime is the period when the TriADC8 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 is required for a given application. For a 100 ms IntegrateTime, the data clock must be:

Equation 18

$$DataClock = \frac{1024}{IntegrateTime} = \frac{1024}{100ms} = 10.24kHz$$

DC and AC Electrical Characteristics

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

Table 1. 5.0V TriADC8 DC and AC Electrical Characteristics, CY8C29/27xxx, CY8CLED08/16, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52 Family of PSoC Devices

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range	---	Vss to Vdd		Ref Mux = Vdd/2 ± Vdd/2
Input Capacitance[1]	3	---	pF	
Input Impedance	1/(C*clk)	---	Ω	
Resolution		8	Bits	
Sample Rate		50 to 5,000	sps	
SNR	48	---	dB	
DC Accuracy				
DNL	0.5	---	LSB	Column clock 2 MHz
INL	0.5	---	LSB	
Offset Error	9	---	mV	
Gain Error				
Including Reference Gain Error	3.0	--	% FSR	
Excluding Reference Gain Error[2]	0.1	--	% FSR	

Parameter	Typical	Limit	Units	Conditions and Notes
Operating Current				
Low Power	500	---	µA	
Med Power	1600	---	µA	
High Power	6000	---	µA	
Data Clock	---	0.125 to 8.0	MHz	Input to digital blocks and analog column clock

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

Table 2. 3.3V TriADC8 DC and AC 3Electrical Characteristics, CY8C29/27xxx, CY8CLED08/16, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52 Family of PSoC Devices

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input Voltage Range	---	Vss to Vdd		Ref Mux = Vdd/2 ± Vdd/2
Input Capacitance[1]	3	---	pF	
Input Impedance	1/(C*clk)	---	Ω	
Resolution		8	Bits	
Sample Rate		50 to 5000	sps	
SNR	77	---	dB	
DC Accuracy				
DNL	0.5	---	LSB	Column clock 2 MHz
INL	0.5	---	LSB	
Offset Error	4	---	mV	
Gain Error				
Including Reference Gain Error	3.0	--	% FSR	
Excluding Reference Gain Error[2]	0.4	--	% FSR	
Operating Current				
Low Power	440	---	µA	
Med Power	1500	---	µA	
High Power	5700	---	µA	
Data Clock	---	0.125 to 8.0	MHz	Input to digital blocks and analog column clock

Electrical Characteristics Notes

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

Placement

The ADC (switch cap) blocks can be placed in any of the switched capacitor PSoC blocks. They must be able to each exclusively drive the comparator bus for the particular column in which it is placed. In other words, each of the three blocks must be in a different column and can not share a column with another switch cap block that connects to the comparator bus.

The counter blocks may be placed in any available digital block, but the PWM16 may only be placed in specific locations. In the CY8C27xxx and CY8CLED08 device families possible placements for the PWB16 (LSB/MSB) are DBB00/DBB01, DBB01/DCB02, DBB10/DBB11, and DBB11/DCB12. In the CY8C29/27xxx, CY8CLED08/16, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52 device families the PWM16 can be placed in any two consecutive digital blocks.

The three counter blocks and PWM block each have an interrupt service routine. It is desirable 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 When initially selecting the TriADC8, a warning may appear that states “Resource allocation prevents placement." This warning is displayed if the original placement has two ADC blocks in the same column. Simply move each ADC block to its own column.

Parameters and Resources

ADC Input1, ADC Input2, ADC Input3

The selection of the input is done after the analog PSoC block has been placed. The eight switched cap blocks have different 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 blocks 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 TriADC8 can also measure signals near the power supply rails. There is one selection for each of the three ADC inputs.

ClockPhase1, ClockPhase2, ClockPhase3

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 TriADC8 is sampled on φ_1, the Normal setting. A problem arises in that many of the user modules autozero their output during φ_1 and only provide a valid output during φ_2. If such a module's output is fed to the TriADC8's input, the TriADC8 acquires an autozeroed 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. There is one selection for each of the three switch cap blocks.

DataClock and Integrator Column Clock

The DataClock 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.

This parameter setting only sets the clock to the counter block and the PWM block.

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

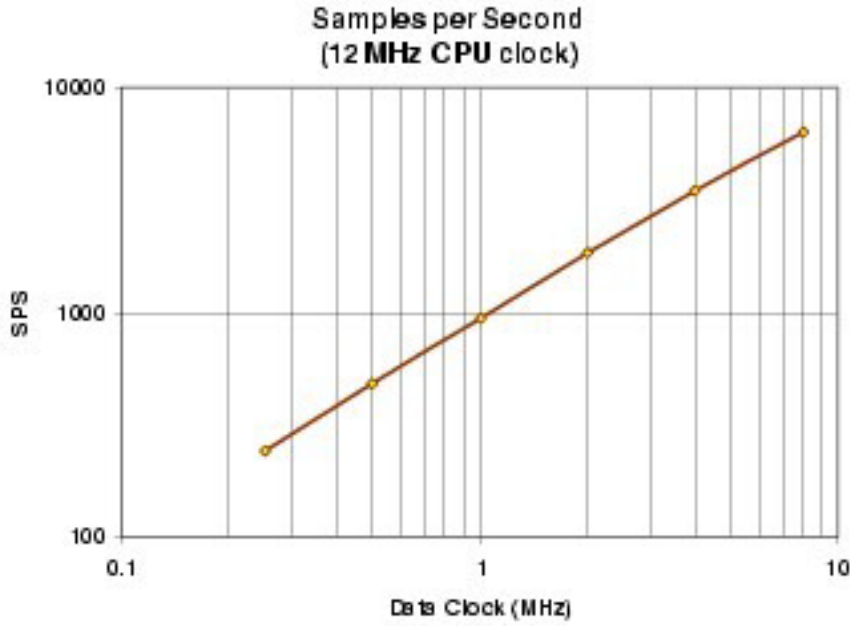
This clock may be any source with a clock rate between 125 kHz and 8 MHz.

Equation 19

$$SampleRate = \frac{DataClock}{2^{10} + CalcTime}$$

The following graph shows possible sample rates for the TriADC8.

Figure 5. Sample Rate vs. Data Clock



Ref Mux Global Resource

The usable input voltage range 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 = 5V, the usable input range is 2.5 +/- 1.3V (1.2 to 3.8V). The following table shows the valid ranges for a Vdd of 5V and 3.3V.

Table 3. CY8C29/27xxx, CY8CLED08/16, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28x43, CY8C28x52 Input Voltage Ranges for Each Ref Mux Setting

RefMux Setting	Vdd = 5 Volts	Vdd = 3.3 Volts
(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]	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$

DataFormat

This selection determines in what format the result is returned. If “Signed" is selected the result ranges from -128 to 127. If “Unsigned" is selected, the result is between 0 and 255.

Interrupt Generation Control

The following parameter is only accessible when 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

Use the IntDispatchMode parameter 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.

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 before 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.

API routines can initialize, configure, start sampling, stop, and read the resultant data from the ADC. In all cases the “instance name" of the module replaces the “TriADC8" prefix shown in the following entry points.

TriADC8_Start

Description:

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

C Prototype:

```
void  TriADC8_Start (BYTE bPowerSetting)
```

Assembly:

```
mov   A, TriADC8_HIGHPOWER
lcall  TriADC8_Start
```

Parameters:

PowerSetting: One byte that specifies the power level. Following reset and configuration, the analog PSoC block assigned to TriADC8 is powered down. Symbolic names, provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value
TriADC8_OFF	0
TriADC8_LOWPOWER	1
TriADC8_MEDPOWER	2
TriADC8_HIGHPOWER	3

Power levels have 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.

TriADC8_SetPower

Description:

Sets the power level for the switched capacitor PSoC block.

C Prototype:

```
void  TriADC8_SetPower (BYTE bPowerSetting)
```

Assembly:

```
mov   A, [bPowerSetting]
lcall  TriADC8_SetPower
```

Parameters:

PowerSetting: Same as the PowerSetting parameter used for the "Start" API routine. Allows the user 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.

TriADC8_Stop

Description:

Sets the power level on the switched capacitor integrator block to Off. This is done when the TriADC8 in not being used and the user wishes 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  TriADC8_Stop()
```

Assembly:

```
lcall  TriADC8_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.

TriADC8_GetSamples

Description:

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

C Prototype:

```
void  TriADC8_GetSamples (void)
```

Assembly:

```
lcall  TriADC8_GetSamples
```

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.

TriADC8_StopAD

Description:

Immediately halts the ADC.

C Prototype:

```
void  TriADC8_StopAD()
```

Assembly:

```
lcall  TriADC8_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.

TriADC8_fIsDataAvailable, fIsData

Description:

Returns nonzero when a data conversion has been completed and data is available for reading.

C Prototype:

```
CHAR  TriADC8_fIsDataAvailable()
CHAR  TriADC8_fIsData()
```

Assembly:

```
lcall  TriADC8_fIsDataAvailable
```

Parameters:

None

Return Value:

Returns nonzero 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.

TriADC8_cGetData1

Description:

Returns last converted data for ADC Input1. 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:

```
char   TriADC8_cGetData1()
```

Assembly:

```
lcall  TriADC8_cGetData1
```

Parameters:

None

Return Value:

Converted value is returned. In assembler, the value is returned 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.

TriADC8_cGetData2

Description:

Returns last converted data for ADC Input2. 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:

```
char   TriADC8_cGetData2()
```

Assembly:

```
lcall  TriADC8_cGetData2
```

Parameters:

None

Return Value:

Converted value is returned. In assembler, the result is returned 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.

TriADC8_cGetData3

Description:

Returns last converted data for ADC Input3. 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:

```
char   TriADC8_cGetData3()
```

Assembly:

```
lcall  TriADC8_cGetData3
```

Parameters:

None

Return Value:

Converted integer value is returned. In assembler, the result is returned 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.

TriADC8_ClearFlag

Description:

Clears Data Available flag.

C Prototype:

```
void  TriADC8_ClearFlag()
```

Assembly:

```
lcall  TriADC8_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.

TriADC8_cGetData1ClearFlag

Description:

Returns last converted data for ADC Input1 and clears the Data Available flag. 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:

```
char   TriADC8_cGetData1ClearFlag()
```

Assembly:

```
lcall  TriADC8_cGetData1ClearFlag
```

Parameters:

None

Return Value:

Converted integer value is returned. In assembler, the result is returned 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.

TriADC8_cGetData2ClearFlag

Description:

Returns last converted data for ADC Input2 and clears the Data Available flag. 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:

```
char   TriADC8_cGetData2ClearFlag()
```

Assembly:

```
lcall  TriADC8_cGetData2ClearFlag
```

Parameters:

None

Return Value:

Converted integer value is returned. In assembler, the result is returned 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.

TriADC8_cGetData3ClearFlag

Description:

Returns last converted data for ADC Input3 and clears the Data Available flag. 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:

```
char   TriADC8_cGetData3ClearFlag()
```

Assembly:

```
lcall  TriADC8_cGetData3ClearFlag
```

Parameters:

None

Return Value:

Converted integer value is returned. In assembler, the result is returned in the Accumulator.

Note The functions ClearFlag, cGetData1ClearFlag, cGetData2ClearFlag, and cGetData3ClearFlag all clear the same flag. They are included to provide the greatest amount of flexibility when clearing the conversion complete flag. When the A/D conversion is complete, the user may choose to ignore the result of one or both channels and simply clear the flag without retrieving the data.

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.

TriADC8_SetCalcTime

Description:

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 Data Clock. Minimum CPU calculation time is 248 CPU clocks. CalcTime may also be increased to optimize the sample rate. **Care must be taken to ensure the CalcTime + 1024 does not exceed 216-1 or 65,535.** Here is an equation to determine what the CalcTime should be set to:

Equation 20

$$CalcTime \geq \frac{DataClock * 248}{CPUClock}$$

For example, if the DataClock is set to 1.5 MHz and the CPU is running at 12 MHz, the CalcTime should be set to greater than or equal to 31. See the following equation:

Equation 21

$$CalcTime \geq \frac{DataClock * 248}{CPUClock} = \frac{1.50MHz * 248}{12.0MHz} = 31_DataClocks$$

C Prototype:

```
void TriADC8_SetCalcTime(int iCalcTime)
```

Assembly:

```
mov   A, [iCalcTimeLSB]
mov   X, [iCalcTimeMSB]
lcall  DualADC8_SetCalcTime
```

Parameters:

Calctime, see previous equations.

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.

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 TRIADC8
;;;  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, TriADC8_HIGHPOWER    ;Set Power and Enable A/D
    call  TRIADC8_Start
    call  TRIADC8_GetSamples
;A/D conversion loop
loop1:
wait:                             ;Poll until data is complete
    call  TRIADC8_fIsDataAvailable
    jz    wait
    call  TRIADC8_ClearFlag       ;Reset flag
    call  TRIADC8_cGetData1       ;Get Data
    call  User_Function           ;Call user routine to use data from
                                  ;ADC Input1
    call  TRIADC8_cGetData2       ;Get Data
    call  User_Function           ;Call user routine to use data
                                  ;ADC Input2
    call  TRIADC8_cGetData3       ;Get Data
    call  User_Function           ;Call user routine to use data
                                  ;ADC Input3
    jmp   loop1
```

A sample project written in C:

```
//-------------------------------------------------------------------------
//  Sample C Code for the TriADC8
//  Continuously Sample and call a user function with the data.
//  This example differs from the ASM example, in that the DataAvailable
//  flag is automatically cleared when the third value is read instead
//  of clearing the flag prior to reading the data.
//
//-------------------------------------------------------------------------
```

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

void main(void)
{
  char cResult1, cResult2, cResult3;
M8C_EnableGInt;                    // Enable global interrupts
TRIADC8_Start(TRIADC8_HIGHPOWER); // Turn on Analog section
TRIADC8_GetSamples();              // Start ADC to read continuously
for(;;)
{
 while(TRIADC8_fIsDataAvailable() == 0);     // Wait for data to be ready
    cResult1 = TRIADC8_cGetData1();             // Get Data from ADC Input1
    cResult2 = TRIADC8_cGetData2();             // Get Data from ADC Input2
    cResult3 = TRIADC8_cGetData3ClearFlag();    // Get Data from ADC Input3
                                                // and clear data ready flag

}
}
```

Configuration Registers

These registers are configured by the initialization and API library. The user does not have to change or read these registers directly. This section is supplied 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 4. Block ADC1: Register CR0

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

Table 5. Block ADC1: 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 the user connects the input. AMux is used when the block is placed in a type “B" block. Field value depends on how the user connects the input.

Table 6. Block ADC1: Register CR2

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

Table 7. Block ADC1: 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 PWM interrupt handler and various APIs. A ‘0‘ value causes the ADC to be a disabled integrator. A ‘1‘ value causes the ADC to be an enabled integrator.

Table 8. Block ADC2: Register CR0

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

Table 9. Block ADC2: 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 the user connects the input. AMux is used when the block is placed in a type “B" block. Field value depends on how the user connects the input.

Table 10. Block ADC2: Register CR2

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

Table 11. Block ADC2: 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 PWM interrupt handler and various APIs. A ‘0‘ value causes the ADC to be a disabled integrator. A ‘1‘ value causes the ADC to be an enabled integrator.

Table 12. Block ADC3: Register CR0

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

Table 13. Block ADC3: 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 the user connects the input. AMux is used when the block is placed in a type “B" block. Field value depends on how the user connects the input.

Table 14. Block ADC3: Register CR2

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

Table 15. Block ADC3: 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 PWM interrupt handler and various APIs. A ‘0‘ value causes the ADC to be a disabled integrator. A ‘1‘ value causes the 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 16. 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 17. 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 18. Block PWM16_MSB: Register Input

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

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

Table 19. Block PWM16_LSB: Register Input

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

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

Table 20. 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 is routed. Both parameters are set in the Device Editor.

Table 21. Block PWM16_LSB: Register Output

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

Table 22. 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 23. 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 24. 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 25. 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 26. 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 27. 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 28. 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 29. 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 30. Block CNT1: Register Function

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

Table 31. Block CNT1: 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 32. Block CNT1: Register Output

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

Table 33. Block CNT1: Register DR0

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

Table 34. Block CNT1: Register DR1

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

Table 35. Block CNT1: 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 36. Block CNT1: 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 TriADC8 API

Table 37. Block CNT2: Register Function

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

Table 38. Block CNT2: 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 39. Block CNT2: Register Output

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

Table 40. Block CNT2: Register DR0

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

Table 41. Block CNT2: Register DR1

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

Table 42. Block CNT2: 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 43. Block CNT2: 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 TriADC8 API.

Table 44. Block CNT3: Register Function

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

Table 45. Block CNT3: 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 46. Block CNT3: Register Output

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

Table 47. Block CNT3: Register DR0

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

Table 48. Block CNT3: Register DR1

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

Table 49. Block CNT3: 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 50. Block CNT3: 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 TriADC8 API.

Table 51. Register INT_MSK1

Bit	7	6	5	4	3	2	1	0
Value								

The mask bits corresponding to the PWM 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
1.0	DHA	Initial version
1.10	DHA	Restored VC3 as the source for the data clock.
1.10.b	MYKZ	Added design rules check for the situation when ADC clock is faster than 8MHz.

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.

Copyright © 2004-2015 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.