Dual Input 8-Bit Incremental ADC Datasheet DualADC8 V 1.20

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	flash	RAM	
CY8C29/27/24xxx, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8C28x43, CY8C28x52, CY8CPLC20, CY8CLED16P01						
	4	0	2	308	7	

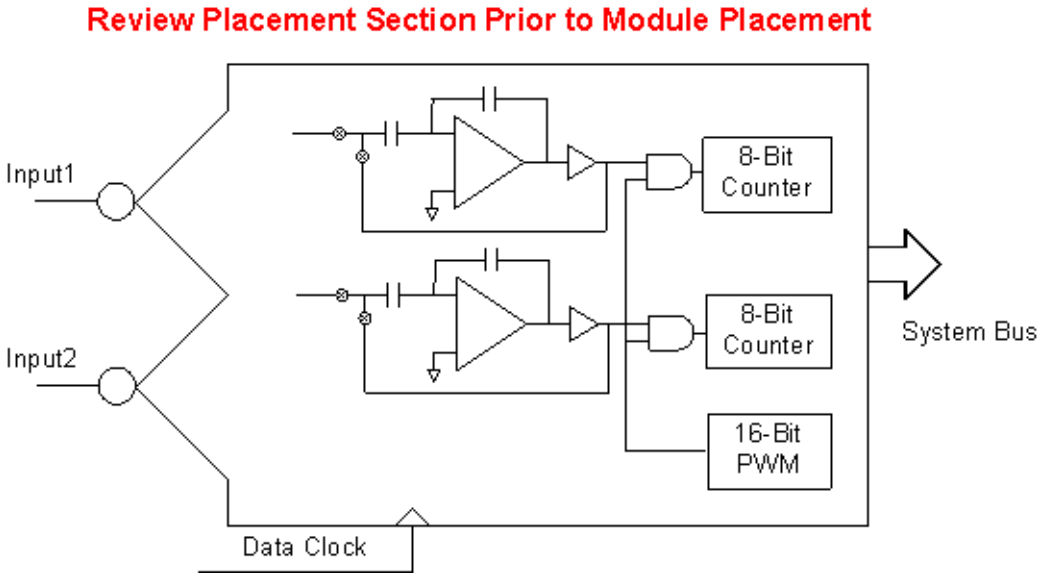
See application note “Analog - ADC Selection” AN2239 for other converters.

Features and Overview

- Samples two inputs simultaneously
- 8-bit resolution
- 2’s complement or unsigned result
- Sample rates from 122 to greater than 7600 sps
- Multiple input ranges including Vss to Vdd
- Integrating Converter provides good normal mode rejection
- Sample rate may be changed dynamically to sync to input signal
- Internal or external clock

The DualADC8 is a dual input integrating 8-bit Analog to digital converter. It can be configured to remove unwanted high frequencies by optimizing the integrate time. Input voltages within 40 mV or less, of the supply rails may be measured by configuring the proper reference voltage and analog ground. The output is configurable 2’s complement or unsigned chars based on an input voltage between –Vref and +Vref centered at AGND.

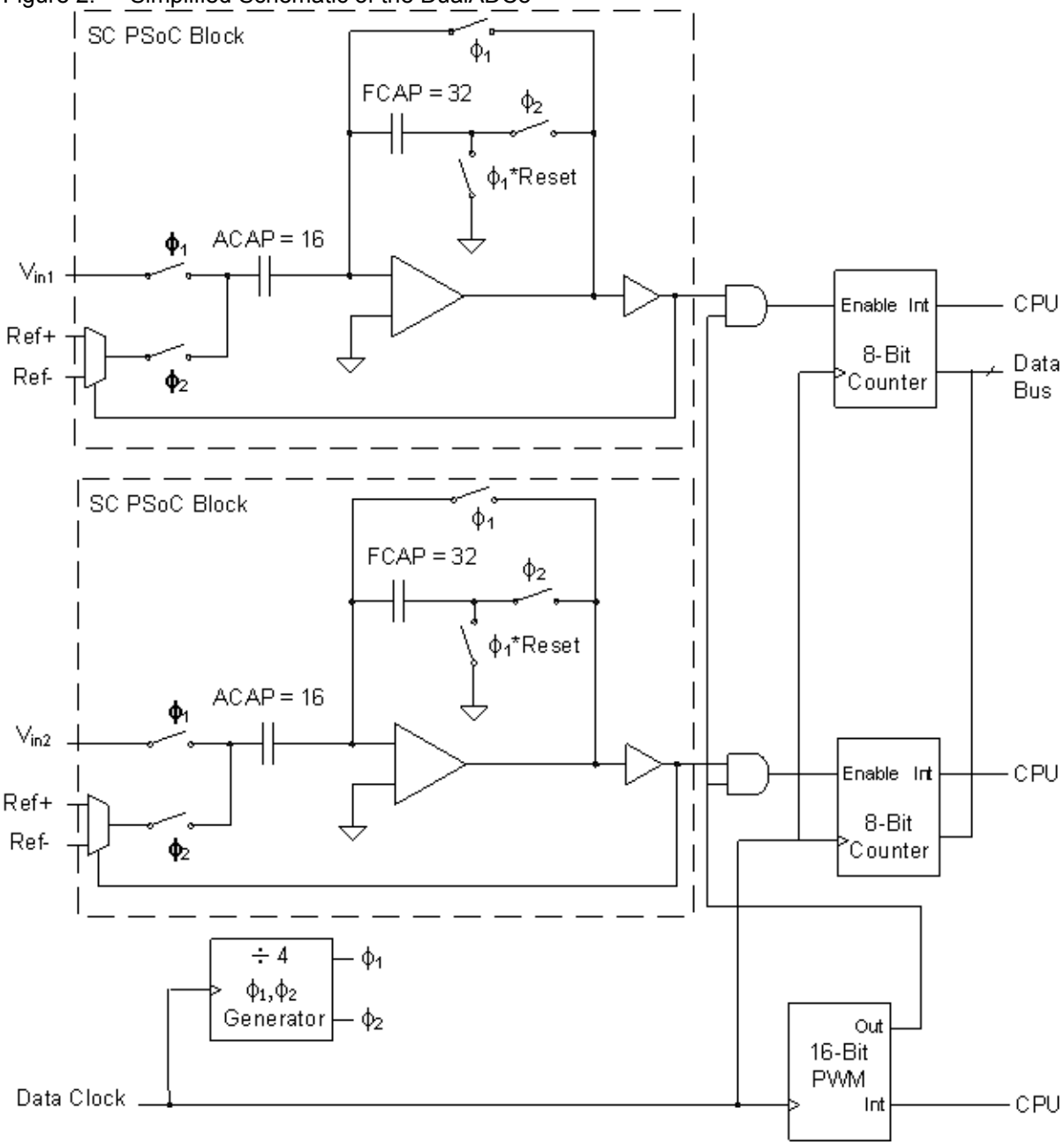
Figure 1. DualADC8 Block Diagram



Functional Description

The DualADC8 User Module is composed of two incremental ADCs in a single user module. The 16-bit sample rate timer (PWM) is shared to reduce the required digital blocks. Since both ADCs use the same timer, the sampling is fully synchronized. At the end of a conversion cycle, results for both inputs are available simultaneously. Four digital PSoC blocks and two analog switch cap PSoC blocks are required, see the figure below for a simplified schematic.

Figure 2. Simplified Schematic of the DualADC8



The two 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^{Bits} times and the output voltage comparator is positive “n" of those times, the residual voltage (V_{resid}) at the output is as follows.

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 listed below.

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 would be as follows:

Equation 4

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

The result of the calculation will be referenced to AGND. For a ADC data value of 200 the Voltage measured can be calculated to be 0.73V as follows:

Equation 5

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

The value calculated is an ideal value and will most likely 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 us:

Equation 6

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

Example 2

For a V_{ref} of 1.3V we can easily calculate the expected ADC code based on the input Voltage. The equation which can be used would be as follows:

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 the calculation below:

Equation 8

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

The value calculated is an ideal value and will most likely differ based on system noise and chips offsets.

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

- 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 both channels).

A single DataClock is connected to the 8-bit counters, the 16-bit PWM, and the analog column clock, which connects 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 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. *It is imperative, when placing this module, that you configure it with the same clock for all blocks. Failure to do so will cause 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 DualADC8 module to be implemented with only four digital blocks instead of six.

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 DualADC8.

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 248 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.

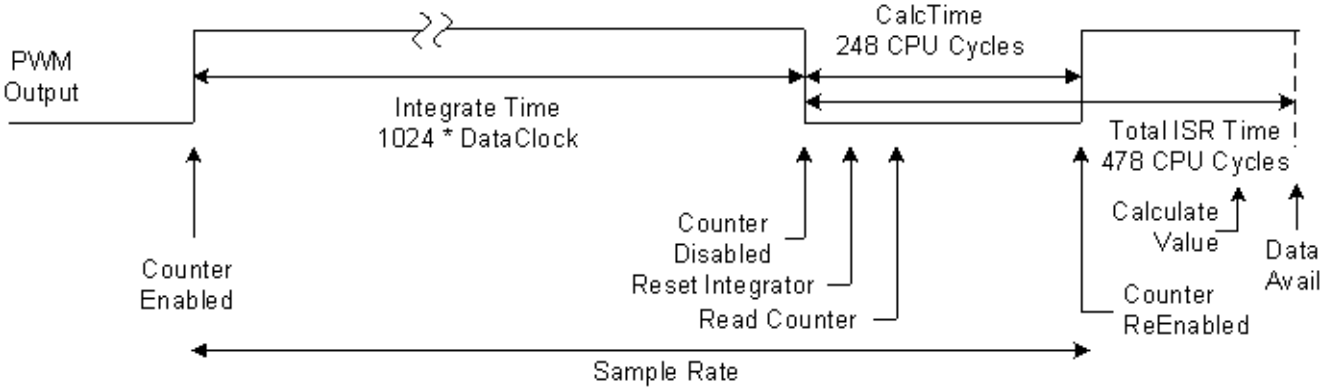
Note The total of 1024 plus the CalcTime must not exceed 2^{16}-1 or 65,535.

Equation 10

$$CalcTime \geq \frac{248 \times DataClock}{CPU\_Clock}$$

The 16-bit PWM is programmed to output a high signal that is 1024 times the DataClock. The PWM output will be 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. DualADC8 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 will always be at least that of the calculation time. The PWM is initialized only prior to the first reading. After the compare and period registers are set once, they do not have to be re-initialized unless the 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 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 4 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 will 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 will never decrement and therefore, never generate an interrupt. An input near analog ground under ideal conditions will allow the counter to increment half the time. Depending on the input voltage level, the amount of interrupts from the 8-bit counters will vary from 0 to 4.

Because the DualADC8 control is interrupt based and the sample time takes a relatively long time for a 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 DualADC8_bfStatus has a non-zero value, the new data is available in DualADC8_iResultn (n=1,2). 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 data handler code into the interrupt routine DualADC8_PWM16_ISR_INT, located in the assembly file *DualADC8INT.asm*. The point to best insert code is clearly marked.

Channel to Channel Differences

When using the DualADC8, there will be 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 Utilization

The DualADC8 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 equations below assume the input signal is the same for both inputs. To calculate the CPU cycles required for a given input, reference the following equations.

Equation 11

$$CPUcycles = 478 + \left(4 \cdot \left(\frac{Vref + Vin}{2 \cdot Vref}\right) \cdot (37 \cdot 2)\right)$$

To calculate the maximum CPU cycles set Vin to Vref, and reference the following equation.

Equation 12

$$CPUcycles = 478 + \left(4 \cdot \left(\frac{Vref + Vin}{2 \cdot Vref}\right) \cdot 74\right) = 478 + (4 \cdot 1 \cdot 74) = 774$$

To calculate the percent CPU utilization of the DualADC8, reference the following equation.

Equation 13

$$Percent\_CPU\_Utilization = \frac{Sample\_Rate \cdot CPUcycles}{CPU\_frequency} \cdot 100$$

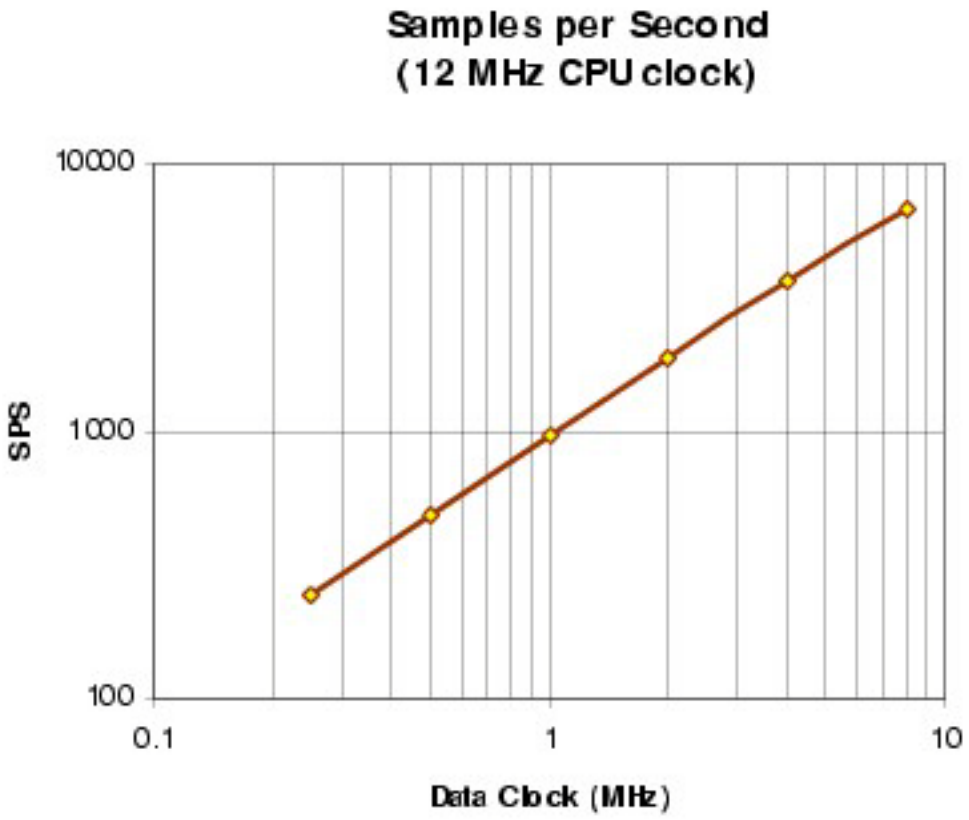
Setting the sample rate to 1000 samples/sec and the CPU clock to 12 MHz, then in the equation below, about 6.5% of the CPU is utilized.

Equation 14

$$Percent\_CPU\_Utilization = \frac{1000 \cdot 774}{12MHz} \cdot 100 = 6.5\%$$

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

Figure 4. CPU Usage versus Sample Rate



Frequency Rejection

By selecting the proper integrate time, specific 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 15

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

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

Equation 16

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

Notice that the CalcTime is not used in this calculation, although it affects the sample rate. The IntegrateTime is the period when the DualADC8 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 5ms is required for a given application, therefore the data clock must be as follows.

Equation 17

$$DataClock = \frac{1024}{IntegrateTime} = \frac{1024}{5ms} = 204.8kHz$$

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 would be as follows.

Equation 18

$$CalcTime = \frac{DataClock \cdot 248}{CPUClock} = \frac{204.8kHz \cdot 248}{12000kHz} = 4.2\_DataClocks$$

This CalcTime should be rounded up to the nearest whole number, in this example it is 5. To determine the sample rate, proceed as follows.

Equation 19

$$SampleRate = \frac{DataClock}{1024 + CalcTime} = \frac{204.8kHz}{1024 + 5} = 199\ Samples/Second$$

If a longer sample rate is desired, the CalcTime may be increased until the CalcTime + 1024 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 2.5V external Analog Ground on P2[4] with 1.25 external Vref on P2[6].

Table 1. 5.0V DualADC8 DC and AC Electrical Characteristics CY8C29/27/24xxx, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8C28x43, CY8C28x52Family 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)	---	W	
Resolution	---	8	Bits	
Sample Rate	---	122 to 7800	SPS	
SNR	47	---	dB	

Parameter	Typical	Limit	Units	Conditions and Notes
DC Accuracy				
DNL	0.25	---	LSB	Column clock 2 MHz
INL	1.0	---	LSB	
Offset Error	9	---	mV	
Gain Error				
Including Reference Gain Error	3.0	--	% FSR	
Excluding Reference Gain Error[2]	0.1	--	% FSR	
Operating Current				
Low Power	370	---	uA	
Med Power	1200	---	uA	
High Power	4000	---	uA	
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, all limits guaranteed for 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].

Table 2. 3.3V DualADC8 DC and AC Electrical Characteristics, CY8C29/27/24xxx, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8C28x43, CY8C28x52Family 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)	---	W	
Resolution	---	8	Bits	
Sample Rate	---	122 to 7800	SPS	
SNR	47	---	dB	
DC Accuracy				
DNL	0.25	---	LSB	Column clock 2 MHz
INL	1.0	---	LSB	
Offset Error	4	---	mV	
Gain Error				

Parameter	Typical	Limit	Units	Conditions and Notes
Including Reference Gain Error	3.0	--	% FSR	
Excluding Reference Gain Error[2]	0.4	--	% FSR	
Operating Current				
Low Power	300	---	uA	
Med Power	1000	---	uA	
High Power	3800	---	uA	
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 +25°C.
4. Input voltages above the maximum will generate a maximum positive reading. Input voltages below the minimum will 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 maximum data clock of 8 MHz, unless otherwise noted. Sample rate is dependent on both data clock and resolution.
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 two 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 device family possible placements for the PWB16 (LSB/MSB) are DBB00/DBB01, DBB01/DCB02, DBB10/DBB11, and DBB11/DCB12. In the CY8C29/24/22xxx device families the PWM16 can be placed in any two consecutive digital blocks.

The two 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 blocks are placed in a lower digital block position than the PWM16 block.

Note When initially selecting the DualADC8, 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

The selection of the ADC Input is done after the analog PSoC block is placed. The eight switched cap blocks have different input selections. Each can be connected to most of its neighbors, while some can be connected directly 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 DualADC8 can also measure signals near the power supply rails. There is one selection for each of the two ADC inputs.

ClockPhase1, ClockPhase2

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 DualADC8 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 DualADC8's input, the DualADC8 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. There is one selection for each of the two switch cap blocks.

Clock 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.

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 six blocks or the DualADC8 User Module will not function correctly.

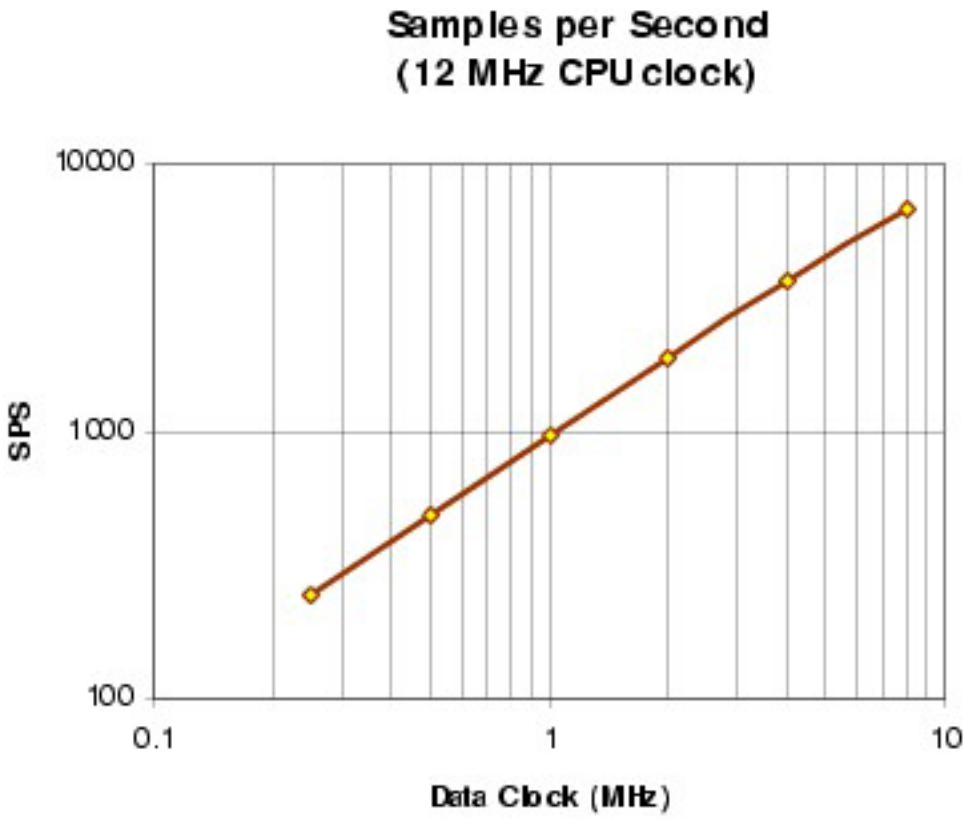
This parameter setting will only set the clock to the counter block and the PWM block. This clock may be any source with a clock rate between 125 kHz and 8 MHz.

Equation 20

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

The graph below shows possible sample rates for each of the resolution options for the DualADC8.

Figure 5. Samples per Second versus Data Clock



Ref Mux Global Resource

The most important global resource when dealing with analog to digital converters (ADC) is the RefMux. The setting of the RefMux determines the usable input voltage range of the ADC. The following table shows the ranges for a Vdd of 5 and 3.3 volts.

Table 3. CY8C29/27/24xxx, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, 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 will be returned. If “Signed" is selected and “N" is the selected resolution, the result will range from –128 to +127. If “Unsigned" is selected, the result will be from 0 to 255.

Interrupt Generation Control

There is an additional parameter that becomes available 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

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.

Application Programming Interface

API routines are provided to initialize, configure, start sampling, stop, and read the resultant data from the ADC.

Note 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.

DualADC8_Start

Description:

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

C Prototype:

```
void  DualADC8_Start (BYTE bPowerSetting)
```

Assembly:

```
mov   A, DualADC8_HIGHPOWER
call  DualADC8_Start
```

Parameters:

PowerSetting: One byte that specifies the power level. Following reset and configuration, the analog PSoC block assigned to DualADC8 is powered down. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value
DualADC8_OFF	0
DualADC8_LOWPOWER	1
DualADC8_MEDPOWER	2
DualADC8_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.

DualADC8_SetPower

Description:

Sets the power level for the switched capacitor PSoC block.

C Prototype:

```
void  DualADC8_SetPower (BYTE bPowerSetting)
```

Assembly:

```
mov   A, [bPowerSetting]
lcall  DualADC8_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.

DualADC8_Stop

Description:

Sets the power level on the switched capacitor integrator block to 0ff. This is done when the DualADC8 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  DualADC8_Stop(void)
```

Assembly:

```
lcall  DualADC8_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.

DualADC8_GetSamples

Description:

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

C Prototype:

```
void  DualADC8_GetSamples (void)
```

Assembly:

```
lcall  DualADC8_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.

DualADC8_StopAD

Description:

Immediately halts the ADC.

C Prototype:

```
void  DualADC8_StopAD(void)
```

Assembly:

```
lcall  DualADC8_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.

DualADC8_fIsDataAvailable

DualADC8_fIsData

DualADC8_fIsDataAvailable is identical to DualADC8_fIsData listed below.

Description:

Returns non-zero when a data conversion is complete and data is available for reading.

C Prototype:

```
BYTE  DualADC8_fIsData(void)
```

Assembly:

```
lcall  DualADC8_fIsData
```

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.

DualADC8_cGetData1

Description:

Returns last converted data for ADC Input1. DualADC8_fIsDataAvailable() should be called prior to 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 will be overwritten. There is a possibility that the returned data will be 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   DualADC8_cGetData1(void)
```

Assembly:

```
lcall  DualADC8_cGetData1
```

Parameters:

None

Return Value:

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

DualADC8_cGetData2

Description:

Returns last converted data for ADC Input2. DualADC8_fIsDataAvailable() should be called prior to 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 will be overwritten. There is a possibility that the returned data will be 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   DualADC8_cGetData2(void)
```

Assembly:

```
lcall  DualADC8_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.

Note The functions DualADC8_ClearFlag, DualADC8_cGetData1ClearFlag, and DualADC8_cGetData2ClearFlag 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.

DualADC8_ClearFlag

Description:

Clears Data Available flag.

C Prototype:

```
void  DualADC8_ClearFlag(void)
```

Assembly:

```
lcall  DualADC8_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.

DualADC8_cGetData1ClearFlag

Description:

Returns last converted data for ADC Input1 and clears the Data Available flag. DualADC8_fIsDataAvailable() should be called prior to 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 will be overwritten. There is a possibility that the returned data will be 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   DualADC8_cGetData1ClearFlag(void)
```

Assembly:

```
lcall  DualADC8_cGetData1ClearFlag
```

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.

DualADC8_cGetData2ClearFlag

Description:

Returns last converted data for ADC Input2 and clears the Data Available flag. DualADC8_fIsDataAvailable() should be called prior to 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 will be overwritten. There is a possibility that the returned data will be 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   DualADC8_cGetData2ClearFlag(void)
```

Assembly:

```
lcall  DualADC8_cGetData2ClearFlag
```

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.

DualADC8_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 should be taken to make sure the CalcTime + of 1024 does not exceed 2^{16}-1 or 65,535. Below is an equation to determine what the CalcTime should be set to;

Equation 21

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

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 equation below.

Equation 22

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

C Prototype:

```
INT DualADC8_SetCalcTime(int iCalcTime)
```

Assembly:

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

Parameters:

Calctime, see equations above.

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

Sample project written in assembly code.

```
;;; Sample ASM Code for the DualADC8
;;;
;;; Continuously sample using the DUALADC8 and store the values in RAM.
;;;
;;; Note: For this example the instance name of this ADC is: "DualADC8"
;;;
;;;    If a different instance name is selected, the variables must
;;;    change to reflect this.
;;;

include "M8C.inc"
include "DualADC8.inc"

;; Create storage for readings
area bss(RAM)
cResult1:       BLK 1    ; ADC1 result storage
cResult2:       BLK 1    ; ADC2 result storage
export cResult1          ; Export results in case they are
export cResult2          ; used elsewhere.

area text(ROM,REL)
export _main

_main:

    M8C_EnableGInt               ; Enable interrupts
    mov   A, 100                 ; Set CalcTime to 100
    mov   X, 0                   ;
    call  DUALADC8_SetCalcTime

    mov   A, DUALADC8_HIGHPOWER  ; Set Power and Enable A/D
```

```
    call  DUALADC8_Start

    call  DUALADC8_GetSamples     ; Start A/D in continuous sampling mode


;A/D conversion loop
loop1:

wait:                             ; Poll until data is complete
    call  DUALADC8_fIsDataAvailable
    jz    wait
    call  DUALADC8_ClearFlag      ; Reset flag

    call  DUALADC8_cGetData1      ; Get ADC1 Data
    mov   [cResult1],A            ; Store value


    call  DUALADC8_cGetData2      ; Get ADC2 Data
    mov   [cResult2],A            ; Store value
    jmp   loop1
```

A sample project written in C follows.

```
//-------------------------------------------------------------------------
//  Sample C Code for the DualADC8
//  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 second value is read, instead of
//  clearing the flag prior to reading the data.
//
//-------------------------------------------------------------------------

#include "DualADC8.h"

extern void User_Function(char cResult1, char cResult2);

void main(void)

{

   char cResult1, cResult2;

   M8C_EnableGInt;                     // Enable global interrupts
   DUALADC8_Start(DUALADC8_HIGHPOWER); // Turn on Analog section
   DUALADC8_SetCalcTime(100);          // Set CalcTime to 100
   DUALADC8_GetSamples();              // Start ADC

   for(;1;)
   {
     while(DUALADC8_fIsDataAvailable == 0);    // Wait for data to be ready
     cResult1 = DUALADC8_cGetData1();          // Get Data from ADC Input1
     cResult2 = DUALADC8_cGetData2ClearFlag(); // Get Data from ADC Input2
                                               // and clear data ready flag
```

```
      User_Function(cResult1,cResult2);          // User function to use data
    }
}
```

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 only.

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 block is placed in a type ‘A’ block. AMux is used when block is placed in a type ‘B’ block. Both field values depend 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. AMux is used when the block is placed in a type ‘B’ block. Both field values depend 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 TMR 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 12. 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 13. 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 14. Block PWM16_MSB: Register Input

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

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

Table 15. 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 16. 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 17. Block PWM16_LSB: Register Output

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

Table 18. Block PWM16_MSB: Count Register DR0

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

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

Table 19. Block PWM16_LSB: Count Register DR0

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

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

Table 20. 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 21. 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 22. 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 23. 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 24. 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 25. Block PWM16_LSB: Control Register CR0

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

Start/Stop indicates that the PWM16 is enabled when set. 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 26. Block CNT1: Register Function

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

Table 27. 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 the input clock from one of 16 sources and is set in the Device Editor.

Table 28. Block CNT1: Register Output

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

Table 29. Block CNT1: Register DR0

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

Table 30. Block CNT1: Register DR1

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

Table 31. Block CNT1: Register DR2

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

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

Table 32. Block CNT1: Register CR0

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

Enable enables the CNT. It is modified and controlled by the DualADC8 API

Table 33. Block CNT2: Register Function

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

Table 34. 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 the input clock from one of 16 sources and is set in the Device Editor.

Table 35. Block CNT2: Register Output

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

Table 36. Block CNT2: Register DR0

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

Table 37. Block CNT2: Register DR1

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

Table 38. Block CNT2: Register DR2

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

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

Table 39. Block CNT2: Register CR0

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

Enable enables the CNT. It is modified and controlled by the DualADC8 API.

Table 40. 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
1.1	DHA	Added DRC to check if: 1. The source clock is different in digital and analog resources. 2. The ADC Clock is higher than CPU Clock.
1.20	DHA	Restored VC3 as the source for the data clock.
1.20.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.