Single Slope 8-Bit ADC Datasheet ADC8 V 1.1

Copyright © 2005-2013 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks			API Memory (Bytes)		Pins (per External I/O)
	Digital	Analog CT	Analog SC	Flash	RAM	
CY8C21x23, CY8C21x34, CY8C21x45, CY8C22x45, CY8CLED02	1	1	1	277	2	1
CYWUSB6953	1	1	1	277	2	1

See AN2239, ADC Selection Guide for other converters.

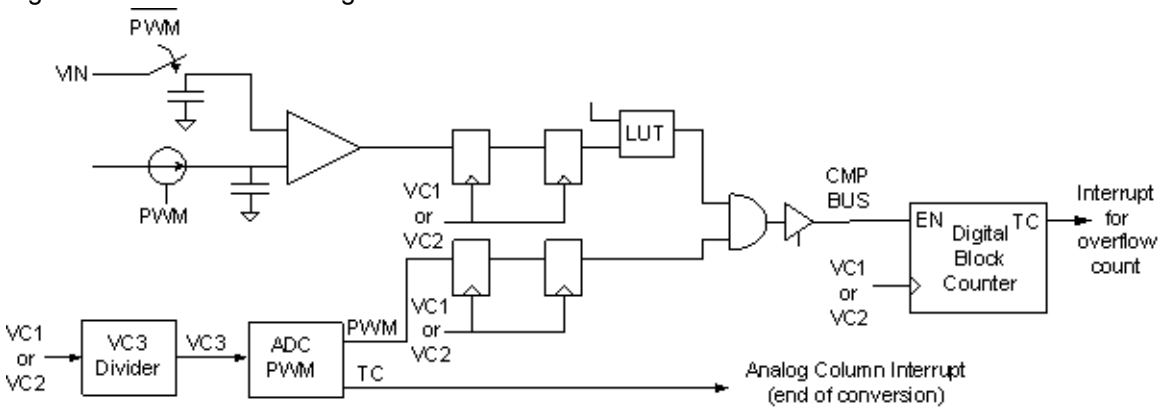
For one or more fully configured, functional example projects that use this user module go to www.cypress.com/psocexampleprojects.

Features and Overview

- 8-bit resolution
- Sample rates up to 8.8 ksps
- Input range 0 to VDD-1 V

The ADC8 User Module implements a Single Slope A/D converter that generates an 8-bit, full scale output (0 to 255 count range).

Figure 1. ADC8 Block Diagram



Functional Description

The ADC8 User Module implements a Single Slope A/D Converter that generates an 8-bit, full scale output (0 to 255 count range). The sampling rate is determined by the user's configuration of the clock sources and is limited to a maximum of 8.8 ksps. A calibration function is available to properly trim the ramp rate of the integrator's slope to accommodate the chosen clock.

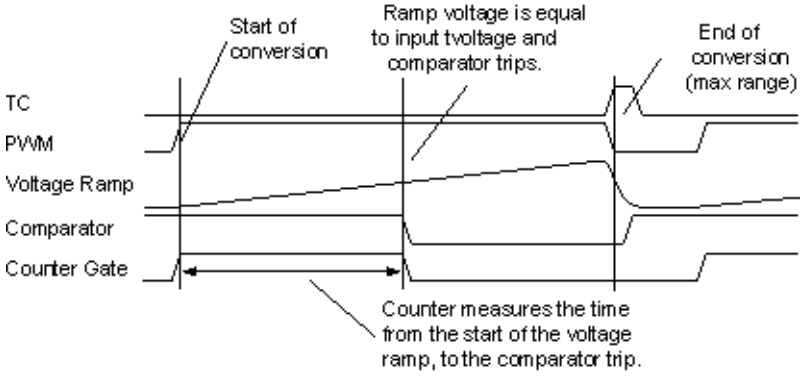
The ADC8 is constructed with the following PSoC resources:

- An analog CT block, configured as a comparator
- An analog SC block, configured as a ramp generator
- A digital block, configured as an eight-bit counter
- A dedicated PWM used for ADC timing

A block diagram of the single slope ADC implementation is shown in the previous section. The core of the conversion algorithm involves a current source, an integrating capacitor and a comparator. When the current source is activated, a linear voltage ramp is generated on the capacitor. The capacitor voltage is an input to an analog comparator circuit, the other input of which is the analog input voltage being converted. The comparator will be high until the ramp voltage exceeds the input voltage at which time it transitions low. The counter keeps track of the time between the start of the ramp and the time when the comparator is tripped.

The basic conversion waveform in show in the diagram below. The high time of the PWM is set so that the Counter will count to 256. The low time of the PWM is designed to allow the capacitor to discharge. The terminal count of the PWM is used as an interrupt to read the results of the conversion. Bit 7 of ADC_CR is set high if the conversion is greater than 255 counts. This PWM generator is clocked from VC3 only and has a limited selection of high and low times.

Figure 2. ADC8 Conversion Timing Diagram



The ADC input is a sample hold of the comparator input. The sample and hold circuitry is controlled by the PWM. This guarantees that the signal remains constant while the conversion takes place. Even though there are two analog columns available, there is only one ADCPWM circuit available and it is shared by both columns, which means that only one ADC8 can be placed in a project.

Calibration

The ADC must be calibrated before use. Each column has an ADC capacitor trim register for this purpose. This register controls the capacitor value that determines the slope of the ADC voltage ramp. The CAPVAL [7:0] bits of the ADC_TR register are used for calibration. Before the converter can be used, the capacitor array must be set to the correct value. The goal of this calibration process is to tune the ramp time (slope) such that a full-scale ADC input value results in a full scale ADC code. This is accomplished by matching the ramp time to the desired full-scale conversion period.

The user module API includes a routine called ADC_bCal that executes a calibration algorithm that trims the ADC_TR register based on a reference voltage and the value that the user identifies as the expected full-scale result for that voltage. The ADC8_bCal routine accepts two arguments – the expected full-scale conversion result for the reference voltage and the source of the reference voltage.

Equation 1 shows how the expected conversion result can be calculated based on the reference voltage and the full-scale voltage:

Equation 1

$$ExpectedCode = \left(\frac{V_{CalRef}}{V_{FullScale}}\right)255$$

The internal Bandgap voltage (AGND), which provides a 1.3-V signal, is a convenient calibration reference. If VDD is at 5 V, and the ADC is powered to FULL_RANGE, then the expected conversion result for the 1.3-V reference is 66 as shown in Equation 2:

Equation 2

$$ExpectedCode = \left(\frac{1.3V}{5V}\right)255 = 66.3$$

Alternatively an external voltage can be placed on any of the pins and then specified as the source for the calibration reference. The bCal routine automatically switches its input to the selected reference source and then returns to the original value upon exit.

The return value of the ADC8_bCal routine specifies the nearest possible value to which it is possible to calibrate the ADC relative to the expected conversion value. If there is a mismatch, then the returned value can be used to perform offset error compensation to achieve more accuracy.

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 = 25 °C, VDD = 5.0 V, Low Range, calibrated to VBG (1.3 V), sample rates of 5.2 ksps, and a data clock of 2 MHz, unless otherwise noted.

Table 1. 5.0 V ADC8 DC and AC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input voltage range	---	Vss to Vdd-1 V		
Input capacitance[1]		--	pF	
Input impedance	1/(C*Clk)	--	Ω	
Resolution	--	8	Bits	
Sample rate	--	.5 to 8.8	ksps	
SNR		--	dB	
DC accuracy				
INL		--	LSB	
DNL		--	LSB	
Offset error		--	mV	
Gain error			mV	
Operating current			uA	
Data clock	--	0.24 to 2.4	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 following table, TA = 25 °C, VDD = 3.3 V, Low Range, calibrated to VBG(1.3 V), sample rates of 5.2 ksps, and a data clock of 2 MHz, unless otherwise noted.

Table 2. 3.3 V ADC8 DC and AC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input voltage range	---	Vss to Vdd-1 V		
Input capacitance[1]		--	pF	
Input impedance	1/(C*Clk)	--	Ω	
Resolution	--	8	Bits	
Sample rate	--	.5 to 8.8	ksps	
SNR		--	dB	
DC accuracy				

Parameter	Typical	Limit	Units	Conditions and Notes
INL		--	LSB	
DNL		--	LSB	
Offset error		--	mV	
Gain error			mV	
Operating current			uA	
Data clock	--	0.24 to 2.4	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 following table, TA = 25°C, Vdd = 2.7V, Low Range, calibrated to VBG(1.3V), sample rates of 5.2 ksps, and a data clock of 2 MHz, unless otherwise noted.

Table 3. 2.7V ADC8 DC and AC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
Input				
Input voltage range	---	Vss to Vdd-1 V		
Input capacitance[1]		--	pF	
Input impedance	1/(C*Clk)	--	Ω	
Resolution	--	8	Bits	
Sample rate	--	.5 to 8.8	ksps	
SNR		--	dB	
DC accuracy				
INL		--	LSB	
DNL		--	LSB	
Offset error		--	mV	
Gain error			mV	
Operating current			uA	
Data clock	--	0.24 to 2.4	MHz	Input to digital blocks and analog column clock

Placement

The analog blocks are placed in the same column and use the comparator bus. There are two columns available on the CY8C21xxxfamily of devices. The Counter block can be placed in any of the digital blocks available. Only one ADC8 User Module can be used within a single project.

Note The single slope ADCs, ADC8 and ADC10, use a hardware resource, ADCPWM, for timing. There is only one ADCPWM resource on the PSoC devices that support this user module; therefore, if you place more than one of these ADCs in a design, one or both of the ADCs exhibits timing problems.

Parameters and Resources

You can select four input parameters:

- Input (Signal Multiplexer)
- Data clock
- Current setting
- PWM High
- PWM Low

Input

This parameter determines the signal source for the A/D conversion. After the analog blocks are placed, this parameter can be configured to one of the allowed values. A signal on a pin can be connected through the use of the Analog Column Input Mux or the Analog Mux Bus. The other options are to connect to the neighboring column or to the Analog Reference VBG.

Data Clock

This parameter selects the clock to be fed into the data block of the user module. This, in turn, determines the clock that should be used to feed into VC3 and the analog column clock. Each ADC sample consists of a measurement period (the high time of the PWM) and a calculation period (the low time of the PWM). The measurement period should be exactly 256 x data clocks. Maintaining this strict relationship between the PWM high time and the Data Clock ensures that the ADC does not roll over and thus give false readings.

The possible values of the Data Clock parameter are limited to VC1 and VC2. The divider value for those clocks should be set so that they lie within the limits set by the maximum and minimum slope generated by the SC block. The clock source provided by VC1 or VC2 should lie between 240 KHz and 2.4 MHz.

For the user module to function properly, the Analog Column Clock for the ADC should be the same as the Data Clock.

Current Setting

Selects either Normal Current (150 nA) or Low Current (37.5 nA).

PWM High

This parameter selects the number of VC3 clocks during which the ADC's dedicated PWM stays high. This number multiplied by the VC3 divider should be equal to 256 Data Clocks and is the measurement period of the A/D conversion.

PWM Low

This parameter selects the number of VC3 clocks during which the ADC's dedicated PWM stays low. This period represents the calculation period of the conversion. In deciding this value it is important to be aware of the fact that the calculation period requires 130 CPU cycles to complete. If any user code is added to the ISR, then it becomes important to accommodate those extra cycles if necessary.

Sample Rate

The Sample Rate is determined by a combination of the Data Clock and the PWM High and Low Times. This section outlines the required equations to determine the Sample Rate.

Equation 3

$$VC3Divider \cdot PWMHigh = 256$$

Equation 4

$$VC3Source = DataClock$$

Equation 5

$$SampleRate = \frac{DataClock(Hz)}{VC3_{(PWMHigh + PWMLow)}}$$

The following table illustrates combinations of Parameter and Resource selections and what kind of sample rates that results in. The Data Clock chosen is assumed to provide a 2 MHz clock.

Data Clock	VC3 Source	VC3 Divider	PWM High	PWM Low	Sample Rate (Hz)
VC1	VC1	256	1	1	3.9 kHz
VC1	VC1	128	2	1	5.2 kHz
VC1	VC1	64	4	1	6.2 kHz
VC2	VC2	32	8	2	6.2 kHz
VC2	VC2	16	16	2	6.9 kHz
VC2	VC2	32	8	3	5.3 kHz

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... > Device Editor. This parameter is only needed when more than one overlay is being 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

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 along with related constants provided by the “include” files.

Note

In this, as in all user module APIs, the values of the A and X register may be altered by calling an API function. It is the responsibility of the calling function to preserve the values of A and X prior to the call if those values are required after the call. This “registers are volatile” policy is selected for efficiency. 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 functions may leave A and X unchanged, there is no guarantee they will 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.

ADC8_Start

Description:

Performs all required initialization for this user module and turns on power for the ADC block.

C Prototype:

```
void  ADC8_Start(BYTE bRange)
```

Assembly:

```
mov   A,ADC8_FULLRANGE
lcall  ACD8_Start
```

Parameters:

bRange: Specifies the measurable input range for the ADC. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value	Description
ADC8_LOWRANGE	1	The input signal can be measured from VSS to VDD-1 V
ADC8_FULLRANGE	3	The input signal can be measured from VSS to VDD

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. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

ADC8_Stop

Description:

Turns off power to the ADC block.

C Prototype:

```
void  ADC8_Stop (void)
```

Assembly:

```
lcall  ACD8_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. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

ADC8_bCal

Description:

Calibrates the ADC by trimming the capacitor in the SC block, which in turn varies the slope of the reference ramp used in the conversion. The ADC8 input must be set to a user selected fixed reference voltage to which to calibrate. This function must be called after calling Start and before calling StartADC. Refer to the Functional Description for more information about calibration and about this function.

C Prototype:

```
BYTE   ADC8_bCal(BYTE bCalVal, BYTE bCalInput)
```

Assembly:

```
mov    A, [bCalVal]
mov    X, ADC8_CAL_P0_5
lcall   ADC_bCal
```

Parameters:

bCalVal: This number is value that is expected given the reference voltage to which the input is set. The ADC8_bCal routine will trim the capacitor in the SC block until the result is equal to or as close to as the bCalVal as possible.

bCalInput: This value specifies the source of the calibration input. This can be an internal reference, such as AGND, which provides a 1.3-V signal or an external reference available on one of the pins. Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Description
ADC8_CAL_VBG	The signal used for calibration will come from VBG the analog reference
ADC8_CAL_AMUXBUS	The signal used for calibration will come from a pin connected to AMUXBUS
ADC8_CAL_P0_0	The signal used for calibration will come from Pin 0.0 (Not available for Col 0)
ADC8_CAL_P0_1	The signal used for calibration will come from Pin 0.1
ADC8_CAL_P0_2	The signal used for calibration will come from Pin 0.2 (Not available for Col 0)
ADC8_CAL_P0_3	The signal used for calibration will come from Pin 0.3
ADC8_CAL_P0_4	The signal used for calibration will come from Pin 0.4 (Not available for Col 0)
ADC8_CAL_P0_5	The signal used for calibration will come from Pin 0.5
ADC8_CAL_P0_6	The signal used for calibration will come from Pin 0.6 (Not available for Col 0)
ADC8_CAL_P0_7	The signal used for calibration will come from Pin 0.7

Return Value:

Returns one byte indicating the nearest result to the bCalVal that was possible.

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. 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.

ADC8_StartADC

Description:

Turns on the ADC and runs it continuously. Global interrupts must be enabled for the ADC to function.

C Prototype:

```
void  ADC8_StartADC(void)
```

Assembly:

```
lcall  ACD8_StartADC
```

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. 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.

ADC8_StopADC

Description:

Immediately stops the ADC.

C Prototype:

```
void  ADC8_StopADC (void)
```

Assembly:

```
lcall  ADC8_StopADC
```

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. 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.

ADC8_fIsDataAvailable

Description:

Checks the status of the ADC.

C Prototype:

```
BYTE  ADC8_fIsDataAvailable(void)
```

Assembly:

```
lcall  ADC8_fIsDataAvailable
```

Parameters:

None

Return Value:

Returns a non zero value if data has been converted and is ready to read.

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. 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.

ADC8_ClearFlag

Description:

Resets the data-available flag.

C Prototype:

```
void  ADC8_ClearFlag(void)
```

Assembly:

```
lcall  ADC8_ClearFlag
```

Parameters:

None

Return Value:

None

Side Effect:

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. 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.

ADC8_bGetData

Description:

Returns converted data. ADC8_fIsDataAvailable() should be called to verify that the data sample is ready.

C Prototype:

```
BYTE  ADC8_bGetData(void)
```

Assembly:

```
lcall  ADC8_bGetData
```

Parameters:

None

Return Value:

Returns the converted data sample.

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. 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.

ADC8_bGetDataClearFlag

Description:

Returns converted data and resets the data available flag. ADC8_fIsDataAvailable() should be called to verify that the data sample is ready.

C Prototype:

```
BYTE  ADC8_bGetDataClearFlag(void)
```

Assembly:

```
lcall  ADC8_bGetDataClearFlag
```

Parameters:

None

Return Value:

Returns the converted data sample.

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. 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

The sample code below polls the Flag register and sends the data to a routine that will shift the data out one of the I/O pins.

```
;;; Sample Code for the ADC8
;;; Continuously Sample input voltage
;;;
include "m8c.inc"       ; part specific constants and macros
include "PSoCAPI.inc"   ; PSoC API definitions for all user modules

export _main

_main:
   M8C_EnableGInt                ; enable global interrupts
   mov  A,ADC8_FULLRANGE
   call ADC8_Start

   mov  A, 42h                   ; Set the calibration value to 42h
   mov  X, ADC8_CAL_VBG          ; Set the calibration source to AGND
   call ADC8_bCal                ; Calibrate the ADC to 1.3V = 42h

   call ADC8_StartADC
wait:
   call ADC8_fIsDataAvailable    ; poll flag
   jz wait

   call ADC8_bGetDataClearFlag   ; retrieve the data and reset flag
   ;; call shift_it_out          ; (user provided) send data to output pin
   jmp wait
```

The same project written in C is:

```
//-------------------------------------------------------------------
// Sample C Code for the ADC8
// Continuously Sample input voltage
//-------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all user modules

BYTE bData;
void main(void)
{
   M8C_EnableGInt;

   ADC8_Start(ADC8_FULLRANGE);     // Start the user module

   ADC8_bCal(0x42, ADC8_CAL_VBG);  // Calibrate the ADC so 1.3V = 0x42

   ADC8_StartADC();                // Start

// Begin sampling
while(1) {
    while(ADC8_fIsDataAvailable() == 0){};
    bData = ADC8_bGetDataClearFlag();
    // Add user code here to Display the result
   }
}
```

Configuration Registers

The input multiplexer determines what signal is digitized.

Table 4. Block ADC: Register ACE_CR1

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

The Input bits determine the signal source for the A/D conversion. This value is set by the Input parameter and can be changed in order to provide alternative reference voltage for calibration.

Table 5. Block ADC: Register ACE_CR2

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

The Enable bit is used to start the user module.

Table 6. Block RAMP: Register ASE_CR0

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

Table 7. Block RAMP Register ADC_CR

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

The Enable bit is used to start the user module. The CMPST bit is a read-only bit that indicates whether the comparator tripped during the conversion period or not. This is used to determine whether an over range condition has occurred or not.

Table 8. Block RAMP: Register ADC_TR

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

The Capacitor Trim Value is set during the bCal function. This 8-bit value sets the value of the adjustable capacitor and thus the slope of the reference ramp used by the ADC.

The CNT is a digital PSoC block configured as a counter.

Table 9. Block CNT: Register Function

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

Table 10. Block CNT: Register Input

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

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

Table 11. Block CNT: Register Output

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

Table 12. Block CNT: Register DR0

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

Table 13. Block CNT: Register DR1

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

Table 14. Block CNT: Register DR2

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

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

Table 15. Block CNT: Register CR0

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

Enable enables CNT when set. It is modified and controlled by the ADC8 API.

Table 16. Register CMP_CR0

Bit	7	6	5	4	3	2	1	0
Value							AINT1	AINT0

The AINT0/1 bits are used to specify the source of the Analog Column Interrupts. The setting of these two bits depends on the placement of the user module. AINT0 is set if the user module is placed in Column 0, and AINT1 is set if the user module is placed in column 1.

Table 17. Register INT_MSK

Bit	7	6	5	4	3	2	1	0
Value								

The mask bits corresponding to the Analog Column is 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 between digital and analog resources. 2. The ADC Clock is higher than CPU Clock.
1.1.b	HPHA	Added design rules check for the situation when the ADC clock is faster than 8 MHz.

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

Copyright © 2005-2013 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.