14-Bit Incremental ADC Datasheet ADCINC14 V 1.4

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	
CY8C29/27/24/22xxx, CY8C23x33, CY8CLED04/08/16, CY8C28x45, CY8C28x43, CY8C28x52						
	4	0	1	266	6	1

See AN2239, ADC Selection Guide for other converters.

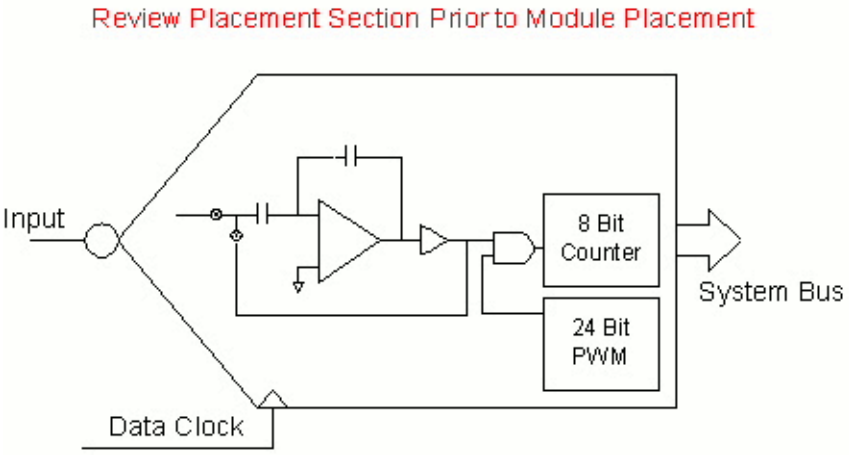
Features and Overview

- 14-bit resolution, 2's complement
- Sample rate from 2 to 120 sps
- Input range from Vss to Vdd
- Integrating converter provides good normal mode rejection
- Internal or external clock

The ADCINC14 is an integrating ADC with 14 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 result format is selectable between signed or unsigned, based on an input voltage between -Vref and +Vref centered at AGND.

Sample rates from 2 to over 120 sps are achievable, depending on the selection of the DataClock and CalcTime parameters. The programming interface allows you to specify the number of sequential samples to be taken or to select continuous sampling. The CPU load varies with the input level. For example, when Vin = +Vref, there are 9832 CPU cycles (maximum 13 bit). When Vin = AGND, there are 5076 CPU cycles. When Vin = -Vref, there are 360 CPU cycles.

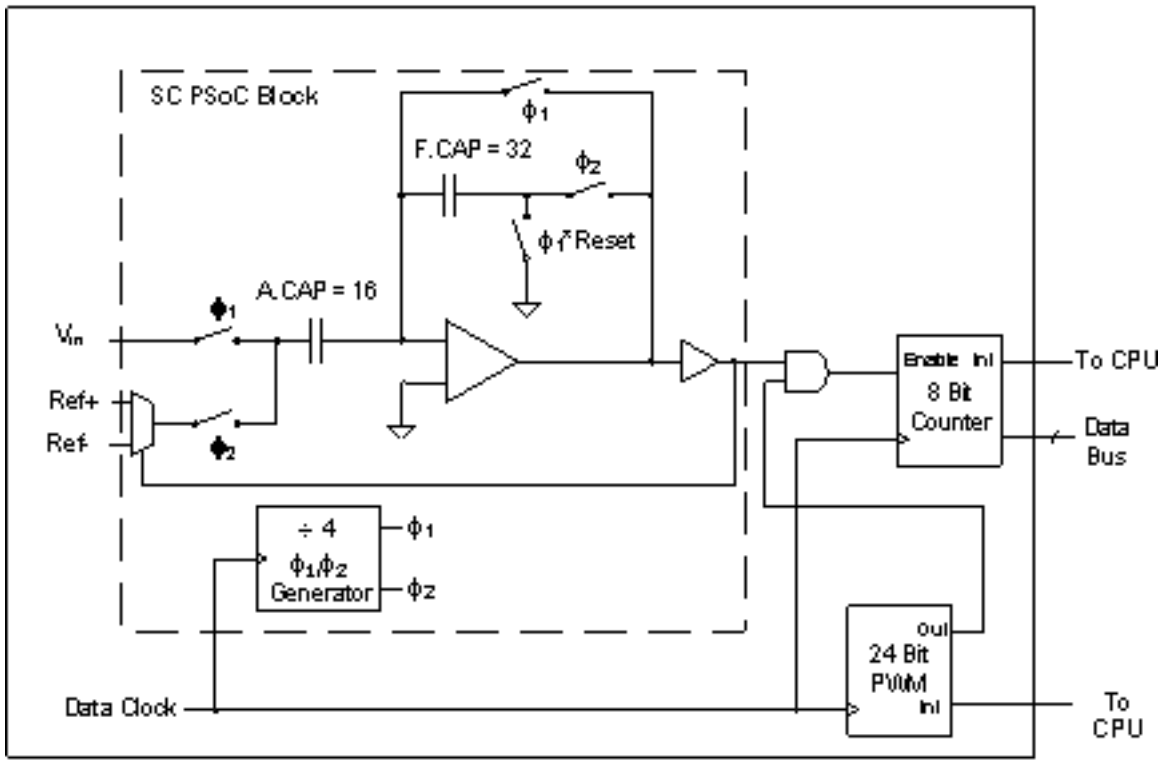
Figure 1. ADCINC14 Block Diagram



Functional Description

The ADCINC14 is formed from a single analog switched cap PSoC block and four digital PSoC blocks, as shown in Figure 2:

Figure 2. Simplified Schematic of the ADCINC14



The analog block is configured as a resettable integrator. 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^{14} times and the output voltage comparator is positive “n" of those times, the residual voltage (V_{resid}) at the output is:

Equation 1

$$V_{resid} = 2^{14} \times V_{in} - n \times V_{Ref} + (2^{14} - n) \times V_{Ref}$$

Equation 2

$$V_{in} = \frac{n - 2^{14-1}}{2^{14-1}} \times V_{Ref} + \frac{V_{resid}}{2^{14}}$$

This equation states that the range of this ADC is ± V_{ref}, the resolution (LSB) is $V_{ref}/2^{14-1}$, and the voltage on the output at the end of the computation is defined as the residue. Since V_{resid} is always less than V_{ref}, $V^{resid}/2^{14}$ is less than half a LSB and can be ignored.

The resulting equation is Equation 3:

Equation 3

$$V_{in} = \frac{n - 8192}{8192} V_{ref}$$

Example 1

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

Equation 4

$$V_{in} = \frac{n - 8192}{8192} 1.3$$

The result of the calculation is referenced to AGND. For a ADC data value of 10000, the Voltage measured is calculated to be 0.29V. This is shown in Equation 5:

Equation 5

$$V_{in} = \frac{10000 - 8192}{8192} 1.3 = 0.29\,V$$

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

To determine the expected code given a specific input voltage, you can rearrange the equation to give:

Equation 6

$$n = \frac{8192 \cdot V_{in}}{V_{ref}} + 8192$$

Example 2

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

Equation 7

$$n = \frac{8192 \cdot V_{in}}{V_{ref}} + 8192$$

For an input voltage of 1V above AGND, the code from the ADC can be expected to be 14493. This is based on Equation 8:

Equation 8

$$n = \frac{8192 \cdot 1}{1.3} + 8192 = 14493$$

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

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

- An 8-bit counter to accumulate the number of cycles that the output is positive.
- A 17-bit PWM to time the integrate time and gate the clock into the 8-bit counter.

A single DataClock is connected to the 8-bit counter, the 17-bit PWM (implemented from a 24-bit PWM), and the analog column clock which connects to the analog SC PSoC block. The analog column clock is actually two clocks, φ_1 and φ_2, which are generated from the data clock. These two additional clocks are exactly one-fourth the frequency of the data clock. This means that the PWM and counter operate 4 times faster than required and therefore, need to accumulate 16 bits worth of data.

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

The counter is implemented with an 8-bit digital block for the LSB and a software counter for the MSB. Each time the hardware counter overflows, an interrupt is generated and the upper MSB of the counter is incremented. This allows the ADCINC14 module to be implemented with only four digital blocks instead of five.

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 ADCINC14.

Equation 9

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

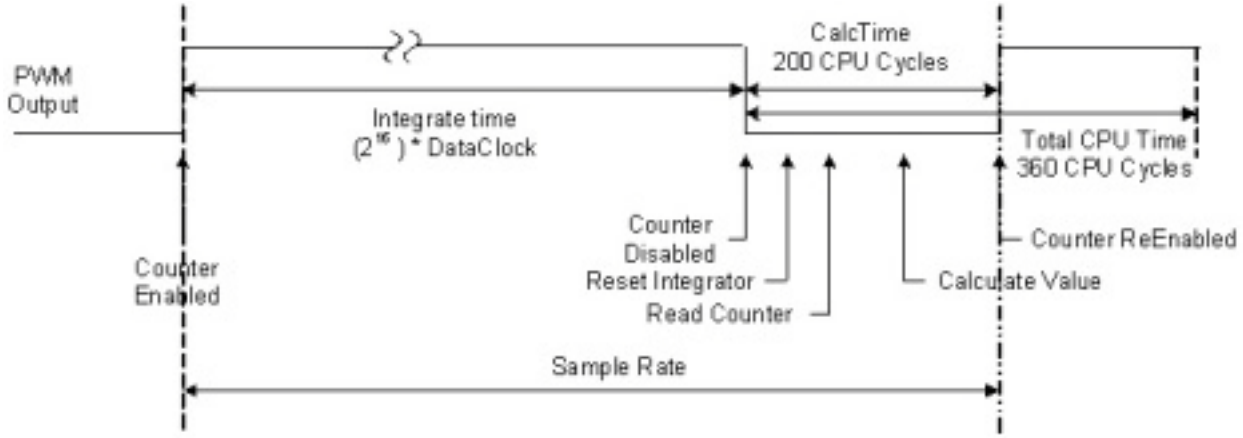
The time it takes to calculate the result, CalcTime, varies inversely proportional with the CPU clock. The CalcTime must be set to a value greater than what is required to calculate the result. The minimum CalcTime is equivalent to 200 CPU cycles and must be expressed in terms of the DataClock. The CalcTime may also be increased beyond the minimum to optimize the sample rate.

Note The total of 2^{16} plus the CalcTime must not exceed 2^{24}-1 or 16,777,215.

Equation 10

$$CalcTime \geq \frac{DataClock * 200}{CPUClock}$$

Figure 3. ADCINC14 Timing with Respect to PWM Output



The 24-bit PWM is programmed to output a high signal that is 2^{16} 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 low time can also be adjusted to help provide a more exact sample rate in combination with the DataClock. The total period of the PWM is the sum of the integrate time and the CalcTime.

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

The initial value of this 8-bit software counter is set to $2^{14}/64$ times the most negative value. Each time the 8-bit counter overflows, the interrupt for the 8-bit counter is executed and the software counter (MSB) is incremented by one.

When the input to the ADC is greater than or equal to the most positive value, the 8-bit counter increments on every positive transition of the DataClock. If the input to the ADC is less than or equal to the most negative input value, the 8-bit counter never decrements and therefore, never generates an interrupt. An input near analog ground under ideal conditions allows the counter to increment half the time. It is easy to see that, depending on the input voltage level, the amount of interrupts from the 8-bit counter varies from 0 to $(2^{16})/256$. This means that it is possible that the processor could be interrupted a maximum of 65536/256 or 256 times during the integrate period.

With the sample time taking 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 least significant bit of ADCINC14_bfStatus has a non zero value, the new data is available in ADCINC14_iResult. 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, you can insert your own data handler code into the interrupt routine ADCINC14_CNT_ISR, located in the assembly file *ADCINC14INT.asm*. The point to best insert code is clearly marked.

CPU Usage

The ADCINC14 requires CPU time to calculate the result and to increment the software counter each time the hardware counter overflows. The CPU overhead is dependent on three variables: CPU clock, DataClock, and input voltage. Note that 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 the most CPU overhead. Use the following equations to calculate the CPU cycles required for a given input:

Equation 11

$$CPUcycles = PWM16_IRQ_CPUcycles + \left(\frac{2^{14}}{64} * \left(\frac{Vref + Vin}{2 * Vref}\right) * Counter_IRQ_CPUcycles\right)$$

Equation 12

$$CPUcycles = 360 + \left(\frac{2^{14}}{64} * \left(\frac{Vref + Vin}{2 * Vref}\right) * 37\right)$$

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

Equation 13

$$CPUcycles = 360 + \left(\frac{2^{14}}{64} * \left(\frac{Vref + Vref}{2 * Vref}\right) * 37\right) = 360 + (256 * 1 * 37) = 9832$$

To calculate the percent CPU usage of the ADCINC14, you can use Equation 14:

Equation 14

$$Percent_CPU_Utilization = \frac{Sample_Rate * CPUcycles}{CPU_freqency} * 100$$

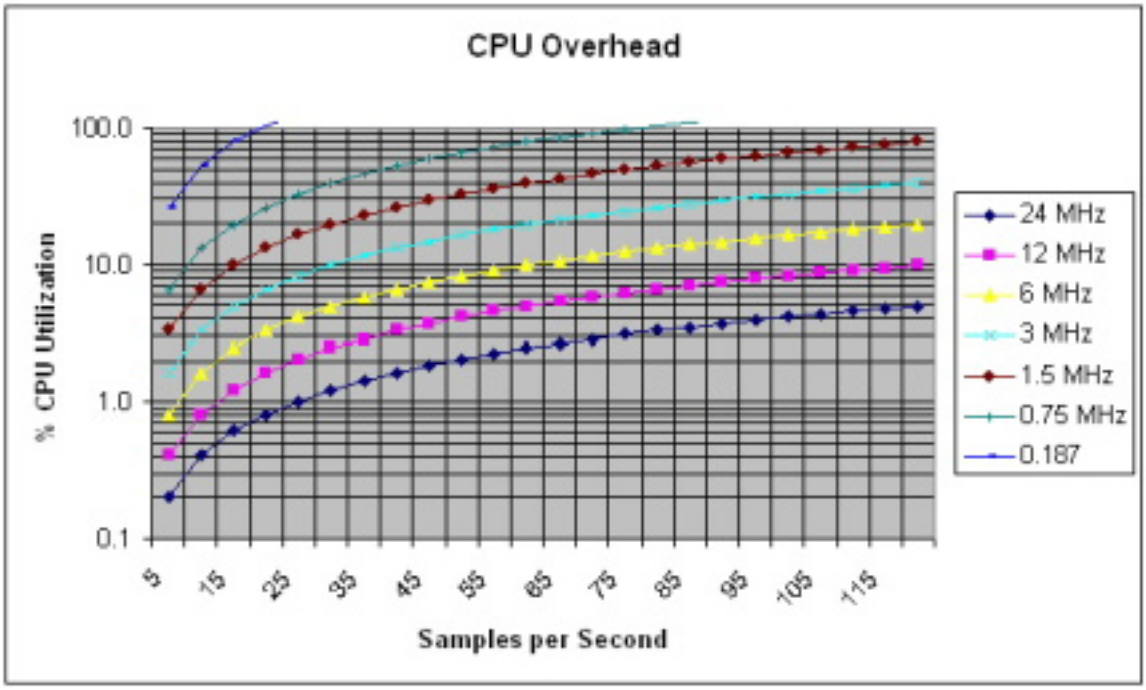
Setting the sample rate to 50 samples/second and the CPU clock to 12 MHz, then (in Equation 15) about 4% of the CPU is used:

Equation 15

$$Percent_CPU_Utilization = \frac{50 * 9832}{12MHz} * 100 = 4.1\%$$

Figure 4 shows CPU usage for the supported sample rates and CPU clock:

Figure 4. CPU Usage vs. Sample Rates and CPU Clock



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.

Example 1

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

Equation 16

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

An Integrate Time 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

$$DataClock = \frac{2^{16}}{IntegrageTime}$$

Notice that the CalcTime is not used in this calculation, although it affects the sample rate. The IntegrateTime is the period when the ADCINC14 is actually sampling the input voltage. The sample rate is based on the IntegrateTime and the time it takes to calculate the result.

Example 2

An IntegrateTime of 100 ms is required for a given application. For a 100 ms IntegrateTime, the data clock must be:

Equation 18

$$DataClock = \frac{2^{16}}{IntegrateTime} = \frac{65536}{100m\,\mathrm{sec}\,?} = 655.4kHz$$

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

Equation 19

$$CalcTime = \frac{DataClock \cdot 200}{CPU_Clock} = \frac{655.4kHz \cdot 200}{12000kHz} \cong 10.9\,DataClocks$$

This CalcTime should be rounded up to the nearest whole number, in this example it is ‘11‘. To determine the sample rate, use Equation 20:

Equation 20

$$SampleRate = \frac{DataClock}{2^{16} + CalcTime} = \frac{655.4kHz}{65536 + 11} = 10\,Samples/Second$$

If a longer sample rate is required, the CalcTime may be increased until the CalcTime + 2^{16} is less than or equal to $2^{24} - 1$ (16,777,215). This should not be much of a restriction.

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 = 5.0V, Power HIGH, opamp bias LOW, input referenced to 2.5V external Analog Ground on P2[4] with 1.25 external Vref on P2[6], and resolution set at 14 bits.

Table 1. 5.0V ADCINC14 DC and AC Electrical Characteristics

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	---	14	Bits	
Sample Rate		2 to 60	sps	
SNR	80	---	dB	
DC Accuracy				
DNL	.25	---	LSB	Column Clock 2.0 MHz
INL	1.5	---	LSB	
Offset Error	4	---	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	750	---	uA	
High Power	2130	---	uA	
Data Clock	---	0.125 to 8	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 = 3.3V, Power HIGH, opamp bias LOW, input referenced to 1.64V external Analog Ground on P2[4] with 1.25 external Vref on P2[6], and resolution set at 14 bits.

Table 2. 3.3V ADCINC14 DC and AC Electrical Characteristics

Parameter	Typical	Limit	Units	Conditions and Notes
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	---	14	Bits	
Sample Rate		2 to 60	sps	
SNR	80	---	dB	
DC Accuracy				
DNL	.25	---	LSB	Column Clock 2.0 MHz
INL	1.5	---	LSB	
Offset Error	4	---	mV	
Gain Error	2.8	---	% FSR	
Including Reference Gain Error	3.0	--	% FSR	
Excluding Reference Gain Error[2]	0.4	--	% FSR	
Operating Current				
Low Power	170	---	uA	
Med Power	540	---	uA	
High Power	1950	---	uA	
Data Clock	---	0.125 to 8	MHz	Input to digital blocks and analog column clock

Note 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.

Placement

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

The counter block may be placed in any available digital block, but the PWM24 may only be placed in specific locations. See the following table valid placements.

Table 3. Valid Placements

Part Family	Valid PWM24 Placements LSB/ISB/MSB
CY8C24/22xxx	DBB00/DBB01/DCB02
CY8C27xxx	DBB00/DBB01/DCB02, DCB03/DBB10/DBB11, and DBB10/DBB11/DCB12
CY8C27X66	DBB00/DBB01/DCB02, DBB01/DCB02/DCB03, DCB02/DCB03/DBB10, DCB03/DBB10/DBB11, DBB10/DBB11/DCB12, DBB11/DCB12/DCB13
CY8C29xxx	DBB00/DBB01/DCB02, DBB01/DCB02/DCB03, DCB02/DCB03/DBB10, DCB03/DBB10/DBB11, DBB10/DBB11/DCB12, DBB11/DCB12/DCB13, DCB12/DCB13/DBB20, DCB13/DBB20/DBB21, DBB20/DBB21/DCB22, DBB21/DCB22/DCB23, DCB22/DCB23/DBB30, DCB23/DBB30/DBB31, DBB30/DBB31/DCB32, DBB31/DCB32/DCB33

Both digital blocks have an interrupt service routine. It is desirable, but not required, that the counter block have a higher interrupt priority than the PWM24 block. Therefore, it is recommended that the counter block be placed in a lower digital block position than the PWM24 block.

Parameters and Resources

Input

The selection of the input is done after the analog PSoC block has been placed. The eight switched cap blocks have differing input selections. Each can be connected to most of its neighbors while some can be directly connected to external input pins. Placement of the analog block must be done with some consideration of how to get an input signal to it. Some placements allow inputs to be routed directly from package pins to the input. These direct connections allow inputs that are within 40 mV of the supply rails to be measured accurately. Signals may also be routed through one of the column muxes, through one of the CT Block test muxes, and onto an analog column where the ADCINC14 can also measure signals near the power supply rails.

ClockPhase

The selection of the Clock Phase is used to synchronize the output of one switched capacitor analog PSoC block to the input of another. The switched cap analog PSoC blocks use a two-phase clock (φ_1, φ_2) to acquire and transfer signal. Typically, the input to the ADCINC14 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 ADCINC14's input, the ADCINC14 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.

CalcTime

The CalcTime is the amount of time it takes the CPU to calculate intermediate integration result before the next integrate cycle can start. The time it takes to calculate the result “CalcTime" varies inversely proportionally with the CPU clock. This value must be in terms of the data clock. Minimum CPU calculation time is 200 CPU clocks. CalcTime may also be increased to optimize the sample rate. Equation 21 determines what the CalcTime should be set to:

Equation 21

$$CalcTime \geq \frac{DataClock * 200}{CPUClock}$$

Example

If the DataClock is set to 1.5 MHz and the CPU is running at 6 MHz, the CalcTime should be set to greater than or equal to 25. This is shown in Equation 22:

Equation 22

$$CalcTime \geq \frac{DataClock * 200}{CPUClock} = \frac{1.50MHz * 200}{12.0MHz} = 25DataClocks$$

Clock and Integrator Column Clock

The data clock determines the sample rate and the signal sample window. This clock must be routed to the clock input of the counter block, the 24-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 clock of the integrator switch cap block must be manually set to the SAME clock. It is imperative that the same clock be used for all three 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:

$$SampleRate = \frac{DataClock}{2^{16} + CalcTime}$$

DataFormat

This selection determines in what format the result is returned. If “Signed" is selected, the data range is between -8182 and 8191. When the unsigned format is selected, the data range is between 0 and 16,383.

Ref Mux Global Resource

The usable input voltage is determined by the selection of the “Ref Mux" option in the “Global Resource" section of the Device Editor. The Ref Mux selection determines the analog ground and the usable range of the input voltage about analog ground. The following table shows the valid ranges for a Vdd of 5V and 3.3V:

Table 4. 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]	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$	$(V_{P2[4]}-V_{P2[6]}) < V_{in} < (V_{P2[4]}+V_{P2[6]})$

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. In all cases the “instance name" of the module replaces the “ADCINC14" prefix shown in the following entry points. Failure to use the correct instance name is a common cause of syntax errors.

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.

ADCINC14_Start

Description:

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

C Prototype:

```
void ADCINC14_Start (BYTE bPowerSetting)
```

Assembly:

```
mov   A, ADCINC14_HIGHPOWER
lcall  ADCINC14_Start
```

Parameters:

PowerSetting: One byte that specifies the power level. Following reset and configuration, the analog PSoC block assigned to ADCINC14 is powered down. Symbolic names, provided in C and assembly, and their associated values, are listed in the following table:

Symbolic Name	Value
ADCINC14_OFF	0
ADCINC14_LOWPOWER	1
ADCINC14_MEDPOWER	2
ADCINC14_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.

ADCINC14_SetPower

Description:

Sets the power level for the switched capacitor PSoC block.

C Prototype:

```
void ADCINC14_SetPower (BYTE bPowerSetting)
```

Assembly:

```
mov   A, [bPowerSetting]
lcall  ADCINC14_SetPower
```

Parameters:

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

Return Value:

None

Side Effects:

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

ADCINC14_Stop

Description:

Sets the power level on the switched capacitor Integrator block to ‘0‘. This is done when the ADCINC14 in not being used and you wish 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  ADCINC14_Stop()
```

Assembly:

```
lcall  ADCINC14_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.

ADCINC14_GetSamples

Description:

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

C Prototype:

```
void ADCINC14_GetSamples (BYTE bNumSamples)
```

Assembly:

```
mov   A, [bNumSamples]
lcall  ADCINC14_GetSamples
```

Parameters:

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

Return Value:

None

Side Effects:

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

ADCINC14_StopAD

Description:

Immediately halts the ADC.

C Prototype:

```
void ADCINC14_StopAD()
```

Assembly:

```
lcall  ADCINC14_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.

ADCINC14_fIsDataAvailable and ADCINC14_fIsData

Description:

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

C Prototype:

```
CHAR ADCINC14_fIsDataAvailable()
CHAR ADCINC14_fIsData()
```

Assembly:

```
lcall  ADCINC14_fIsDataAvailable
```

Parameters:

None

Return Value:

Returns non-zero when data is available.

Side Effects:

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

ADCINC14_iGetData

Description:

Returns last converted data. 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. If that cannot be guaranteed, then interrupts must be turned off before calling this function.

C Prototype:

```
INT ADCINC14_iGetData()
```

Assembly:

```
lcall  ADCINC14_iGetData
```

Parameters:

None

Return Value:

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

Side Effects:

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

ADCINC14_ClearFlag

Description:

Clears Data Available flag.

C Prototype:

```
void ADCINC14_ClearFlag()
```

Assembly:

```
lcall  ADCINC14_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.

ADCINC14_iGetDataClearFlag"

Description:

Returns last converted data 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 will be overwritten. There is a possibility that the returned data is corrupted if the call to this function is done exactly at the end of an integration period. It is therefore highly recommended that the data retrieval be done at a higher frequency than the sampling rate, or if that cannot be guaranteed that interrupts be turned off before calling this function.

C Prototype:

```
INT ADCINC14_iGetDataClearFlag()
```

Assembly:

```
lcall  ADCINC14_iGetDataClearFlag
```

Parameters:

None

Return Value:

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

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model (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 ADCINC14
;;; 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, ADCINC14_HIGHPOWER     ;Set Power and Enable A/D
    call  ADCINC14_Start

    mov   a, 00h                    ;Start A/D in continuous sampling mode
    call  ADCINC14_GetSamples

;A/D conversion loop
loop1:

    wait:                           ;Poll until data is complete
    call  ADCINC14_fIsDataAvailable
    jz    wait

    call  ADCINC14_ClearFlag        ;Reset flag
    call  ADCINC14_iGetData         ;Get Data - X=MSB A=LSB
    ;; Place User code here

    jmp   loop1
```

The same project written in C is:

```
//-------------------------------------------------------------------------
//  Sample C Code for the ADCINC14
//  Continuously Sample and call a user function with the data.
//
//-------------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void main(void)
{
INT iData;
    M8C_EnableGInt;                         // Enable global interrupts
    ADCINC14_Start(ADCINC14_HIGHPOWER);     // Turn on Analog section
    ADCINC14_GetSamples(0);                 // Start ADC to read continuously
    for(;;)
    {
      while(ADCINC14_fIsDataAvailable() == 0); // Wait for data to be ready
      iData = ADCINC14_iGetData();             // Get Data
      ADCINC14_ClearFlag();                    // Clear data ready flag
      // Place user code here

    }
}
```

Configuration Registers

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

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

Table 5. Block ADC, Register: CR0

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

Table 6. Block ADC: Register CR1

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

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

Table 7. Block ADC: Register CR2

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

Table 8. Block ADC: Register CR3

Bit	7	6	5	4	3	2	1	0
Value	1	1	FSW1	FSW0	0	0	PWR	

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

The PWM24 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 9. Block PWM24_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 Compare Type and Interrupt Type are set in the Device Editor.

Table 10. Block PWM24_ISB: 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 11. Block PWM24_LSB: Register Function

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

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

Table 12. Block PWM24_MSB: Register Input

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

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

Table 13. Block PWM24_ISB: Register Input

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

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

Table 14. Block PWM24_LSB: Register Input

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

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

Table 15. Block PWN24_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 PWN24 will be routed. Both parameters are set in the Device Editor.

Table 16. Block PWM24_ISB: Register Output

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

Table 17. Block PWM24_LSB: Register Output

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

Table 18. Block PWM24_MSB: Count Register DR0

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

Count: PWM24 MSB down PWM. It can be read using the PWM24 API.

Table 19. Block PWM24_ISB: Count Register DR0

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

Count: PWM24 ISB down PWM. It can be read using the PWM24 API.

Table 20. Block PWM24_LSB: Count Register DR0

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

Count: PWM24 LSB down PWM. It can be read using the PWM24 API.

Table 21. Block PWM24_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 PWM24 API.

Table 22. Block PWM24_ISB: Period Register DR1

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

Period holds the ISB 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 PWM24 API.

Table 23. Block PWM24_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 PWM24 API.

Table 24. Block PWM24_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 PWM24 API.

Table 25. Block PWM24_ISB: Pulse Width Register DR2

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

PulseWidth holds the ISB of the pulse width value used to generate the compare event. It can be set by the Device Editor and the PWM24 API.

Table 26. Block PWM24_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 PWM24 API.

Table 27. Block PWM24_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 28. Block PWM24_ISB: Control Register CR0

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

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

Table 29. Block PWM24_LSB: Control Register CR0

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

Start/Stop, when set, indicates that the PWM24 is enabled. It is modified by using the PWM24 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 output through DR2.

Table 30. Block CNT: Register Function

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

Table 31. Block CNT: Register Input

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

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

Table 32. Block CNT: Register Output

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

Table 33. Block CNT: Register DR0

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

Table 34. Block CNT: Register DR1

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

Table 35. 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 36. Block CNT: Register CR0

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

When Enable is set, CNT is enabled. It is modified and controlled by the ADCINC14 API

Table 37. 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
1.4	DHA	Added DRC to check if: 1. The source clock is different between digital and analog resources. 2. The ADC Clock is higher than CPU Clock. Added DRC warning when two ADCINC14 or ADCINCVR blocks are added into a project.
1.4.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.