Incremental ADC Datasheet ADCINC V 3.00

Copyright © 2008-2013 Cypress Semiconductor Corporation. All Rights Reserved.

Resources	PSoC® Blocks				API Memory (Bytes)		Pins (per External I/O)
	CapSense®	I2C/SPI	Timer	Comparator	Flash	RAM	
CY8C20xx6/6A/6AS/6H/6L, CY8C20xx7/7S, CY7C643xx, CY7C604xx, CYONS2xxx, CYONSxNxxxx, CYRF89x35, CY8C20065, CY8C24x93, CY7C69xxx							
	-	-	-	-	1000	100	1

Features and Overview

- 8 to 10-bit resolution
- Sample rate up to 11.71 ksps (10-bit resolution)
- Sample rate up to 46.875 ksps (8-bit resolution)
- Input range up to reference voltage
- Internal clock divider

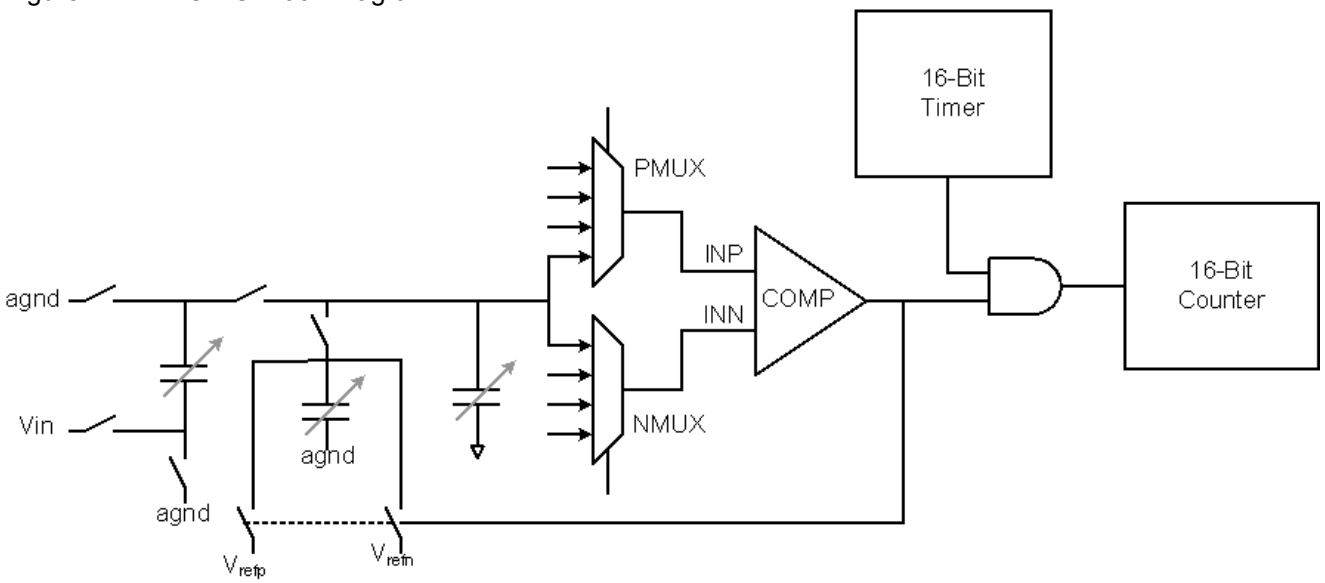
The ADCINC User Module is part of the System Performance Controller (SPC). SPC is a modular system for product performance compensation over process, voltage, and temperature variations. The system is based on the M8C microcontroller. The heart of the SPC is the microcontroller core, temperature sensor, and the SPC system bus.

The SPC is intended to be a slave peripheral under command of the application. The SPC core is usually powered down. When the application processor writes an SPC command, the SPC wakes up and executes the command. After the command is complete, the SPC automatically reenters a power down state. The SPC contains one read/write data port and one read-only status port.

The ADCINC User Module implements an incremental analog-to-digital converter with a range of 8 to 10 bits and signed or unsigned data formats. The input voltage range is fixed at 0 to reference voltage. The ADCINC programming interface enables you to select between polling the SPC or checking a status bit that is set with the SPC interrupt.

All access to control and recover the result is through two registers used to interface the main M8C processor and the SPC. The ADC consists of a timer that determines the conversion time and the resolution, a counter to accumulate the result, and an analog modulator.

Figure 1. ADCINC Block Diagram



The ADCINC User Module contains an integrator block and one comparator. The comparator's positive input is set by the PMUX. Its negative input is set by the NMUX (See Figure 1 above). The input to the integrator stage comes from the Analog Global Input Mux with full scale input being 0V to reference voltage. The ADC is run for a number of cycles set by the Timer, depending upon the resolution of the ADC that is required. A counter counts the number of trips by the comparator, which is proportional to the input voltage. The SPC clock speed is 36 MHz and is divided down to 12 – 2.4 MHz for ADCINC operation.

Functional Description

The range of the ADCINC is set at 0 to reference voltage. The analog block is configured as a integrator that can be reset. Depending on the output polarity, the reference control is configured so that the reference voltage is either added or subtracted from the input and placed in the integrator. This reference control attempts to pull the integrator output back towards AGND. If the integrator is operated 2^{Bits} times and the output voltage comparator is positive "n" of those times, the residual voltage (V_{resid}) at the output is:

Equation 1

$$Result = \frac{V_{in}}{V_{ref}}(2^n)$$

This equation states that the range of this ADC is 0 to V_{ref}, the resolution (LSB) is $V_{ref}/2^n$, and the voltage on the output at the end of the computation is defined as the residue. Since V_{resid} is always less than V_{ref}, $V_{resid}/2^{Bits}$ less than half an LSB and can be ignored.

The accumulated value is sampled at the start and finish of the integrate time. A single cycle is added to reset the integrator and process the answer.

Timing

The Pulse Width Modulator (PWM) is set up to generate an interrupt every 2^n counts. This causes the input to be sampled 64 times. This defines one integrate cycle. The decimator counter is set up to accumulate $2^n/64$ bits of these integrate cycles. The accumulated value is sampled at the start and finish of the integrate time. A single cycle is added to reset the integrator and process the answer.

Because the ACDINC control is interrupt-based and the sample time is relatively long, 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 data-available flag that may be polled. APIs are available to check the data flag and retrieve data.

DC and AC Electrical Characteristics

The following table lists the guaranteed maximum and minimum specifications. Unless stated otherwise, the specifications are for the entire device voltage and temperature operating range: –40 °C ≤ T_A ≤ 85 °C, 1.71 V ≤ V_{DD} ≤ 5.5 V.

Symbol	Description	Conditions	Min	Typ	Max	Units
Input						
V_{IN}	Input voltage range	–	0	–	V_{REFADC}	V
C_{IN}	Input capacitance	–	–	–	5	pF
R_{IN}	Input resistance	Equivalent switched cap input resistance for 8-, 9-, or 10-bit resolution		1/(400fF × data clock)		Ω
Reference						
V_{REFADC}[1]	ADC reference voltage	–	1.14	–	1.26	V
Conversion Rate						
F_{CLK}	Data clock	Source is chip's internal main oscillator. See AC Chip-Level Specifications for accuracy	2.25	–	6	MHz
S8	8-bit sample rate	Data clock set to 6 MHz. sample rate = 0.001/ (2^Resolution/Data Clock)	–	23.43	–	ksps
S10	10-bit sample rate	Data clock set to 6 MHz. sample rate = 0.001/ (2^resolution/data clock)	–	5.85	–	ksps
DC Accuracy						
RES	Resolution	Can be set to 8-, 9-, or 10-bit	8	–	10	bits
DNL	Differential nonlinearity	–	–1	–	+2	LSB
INL	Integral nonlinearity	–	–2	–	+2	LSB
E_{OFFSET}	Offset error	8-bit resolution	0	3.20	19.20	LSB
		10-bit resolution	0	12.80	76.80	LSB

Symbol	Description	Conditions	Min	Typ	Max	Units
E_{GAIN}	Gain error	For any resolution	–5	–	+5	%FSR
Power						
I_{ADC}	Operating current	–	–	2.10	2.60	mA
PSRR	Power supply rejection ratio	PSRR (V_{DD} > 3.0 V)	–	24	–	dB
		PSRR (V_{DD} < 3.0 V)	–	30	–	dB

Note 1: The precise value can be obtained from the device datasheet.

Placement

The ADCINC User Module consumes the SPC block, but this block is not displayed in the Chip Editor. Only one instance of the ADCINC User Module can be placed because one SPC block is available in the chips, which support this user module.

Parameters and Resources

After the ADCINC is placed, these parameters must be configured for proper operation: ADC Resolution and Clock Divider.

ADC Resolution

This selection determines the data format of the return result. Valid resolution options are from 8 to 10 bits. Set by cap select (Cs Select) and Timer period.

Clock Divider

The Data Clock determines the sample rate. The maximum DataClock that can be used is 12 MHz. This is due to limitations of the Switched Cap blocks. The maximum sample rate for each of the various bit rates are listed in the following table. Operation at absolute maximum sample rate results in reduced accuracy and does not meet the INL/DNL specs listed in the AC and DC electrical specifications.

Resolution	Max Counts	Recommended Maximum Sample Rate	Absolute Maximum Sample Rate
8-bit	255	23.43 ksps	46.875 ksps
9-bit	511	11.71 ksps	23.43 ksps
10-bit	1023	5.85 ksps	11.71 ksps

The sample window determines the normal mode frequencies the ADC rejects. It is defined as:

Equation 2

$$SampleWindow = \frac{2^{Bits}}{DataClock}$$

Application Programming Interface

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

Each time a user module is placed, it is assigned an instance name. By default, PSoC Designer assigns ADCINC_1 to the first instance of this user module in a given project. It can be changed to any unique value that follows the syntactic rules for identifiers. The assigned instance name becomes the prefix of every global function name, variable, and constant symbol. In the following descriptions the instance name has been shortened to ADCINC for simplicity.

Note

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

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

ADCINC_Start

Description:

The ADCINC for this device is internally constructed to connect to only two sources:

1. Internal temp sensor
2. Analog mux bus

If the analog mux bus is chosen, an external pin may also be connected to the same mux bus to give an external sense point.

C Prototype:

```
void  ADCINC_Start (BYTE bMux)
```

Assembly:

```
mov   A, bMux
lcall  ADCINC_Start
```

Parameters:

bMux: One byte that specifies the connection chosen as described earlier. The symbolic names and their associated values given in C and assembly are listed in the following table:

Symbolic Name	Value
ADCINC_INPUT_ANALOG_BUS	0
ADCINC_INPUT_TEMP_SENSOR	1

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.

ADCINC_Stop

Description:

Disables the ADC in the SPC.

C Prototype:

```
void  ADCINC_Stop (void)
```

Assembly:

```
lcall  ADCINC_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.

ADCINC_GetSample

Description:

Runs the ADC until one ADC sample is complete.

C Prototype:

```
WORD  ADCINC_GetSample (void)
```

Assembly:

```
lcall  ADCINC_GetSample
```

Parameters:

None

Return Value:

WORD (ADC_reading)

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.

ADCINC_StartSample

Description:

Starts the single-ADC conversion. To check if conversion is complete, call the ADCINC_fIsDataAvailable() function.

C Prototype:

```
void  ADCINC_StartSample (void)
```

Assembly:

```
lcall  ADCINC_StartSample
```

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.

ADCINC_fIsDataAvailable

Description:

Checks the availability of sampled data.

C Prototype:

```
BYTE  ADCINC_fIsDataAvailable(void)
```

Assembly:

```
lcall  ADCINC_fIsDataAvailable
```

Parameters:

None

Return Value:

Returns a nonzero 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 (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.

ADCINC_iGetData

Note: This API function is legacy and is not recommended for new designs.

Description:

Returns converted data as a signed integer. ADCINC_fIsDataAvailable() should be called to verify that the data sample is ready. The data valid status is cleared after this function is called.

C Prototype:

```
INT   ADCINC_iGetData(void)
```

Assembly:

```
lcall  ADCINC_iGetData           ; Data will be in A and X upon return
```

Parameters:

None

Return Value:

Returns the converted data sample in 16-bit 2’s complement format.

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.

ADCINC_wGetData

Description:

Returns converted data as an unsigned integer. ADCINC_fIsDataAvailable() should be called to verify that the data sample is ready. The data valid status is cleared after this function is called.

C Prototype:

```
WORD  ADCINC_wGetData(void)
```

Assembly:

```
lcall  ADCINC_wGetData           ; Data will be in A and X upon return
```

Parameters:

None

Return Value:

Returns the converted 16-bit unsigned 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 (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.

ADCINC_cGetData

Note: This API function is legacy and is not recommended for new designs.

Description:

Returns converted data as a signed char. ADCINC_fIsDataAvailable() should be called to verify that the data sample is ready. The data valid status is cleared after this function is called.

C Prototype:

```
CHAR  ADCINC_cGetData(void)
```

Assembly:

```
lcall  ADCINC_cGetData           ; Data will be in A upon return
```

Parameters:

None

Return Value:

Returns the converted data sample in 8-bit 2’s complement format.

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.

ADCINC_bGetData

Description:

Returns converted data as an unsigned char. ADCINC_fIsDataAvailable() should be called to verify that the data sample is ready. The data valid status is cleared after this function is called.

C Prototype:

```
BYTE  ADCINC_bGetData(void)
```

Assembly:

```
lcall  ADCINC_bGetData           ; Data will be in A upon return
```

Parameters:

None

Return Value:

Returns the converted 8-bit unsigned 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 (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

The following sample code polls the Flag register and sends the data to a routine that shifts the data out one of the I/O pins:

```
include "m8c.inc"                ; part specific constants and macros
include "PSoCAPI.inc"            ; PSoC API definitions for all User Modules
export _main
_main:
   M8C_SetBank1
   mov reg[MUX_CR1],  0x10;  //connect p1_4 to the mux bus
   M8C_SetBank0
   M8C_EnableGInt                  ; enable global interrupts
mov  a,ADCINC_INPUT_ANALOG_BUS  ; set ADC Mode
call ADCINC_Start
call ADCINC_GetSample           ;places WORD data in A and X
loop1:
   call ADCINC_StartSample
wait:
   call ADCINC_fIsDataAvailable    ; poll flag
jz wait
call ADCINC_iGetData            ; reset flag and retrieve data
;; call shift_it_out            ; (user provided) send data to output pin
jmp loop1
```

A similar project written in C is:

```
//-------------------------------------------------------------------------
//  Sample C Code for the ADCINC
//  Continuously Sample input voltage
//
//-------------------------------------------------------------------------
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
INT val;
void main(void)
{
    // Insert your main routine code here.
    val = 0xaa;
    LCD_Start();
    LCD_Position(1,0);
    LCD_PrCString("value = ");
    //connect the pins up
    MUX_CR1 = 0x10;  //connect p1_4 to the mux bus

    ADCINC_Start(ADCINC_INPUT_ANALOG_BUS);

while(1)
{
    val = ADCINC_GetSample();
LCD_Position(1,8);
    LCD_PrHexInt(val);
}
}
```

Configuration Registers

This ADCINC has no user registers. Any settings must be performed through the API.

Version History

Version	Originator	Description
1.1	DHA	Corrected Sample code for C language.
1.20	DHA	Added additional poll_int instructions into GetSample() and StartSample() API functions.
1.20.b	DHA	The following updates were done for this user module datasheet: 1. Removed "ADC Mode" parameter. 2. Corrected valid resolution options. 3. Updated Input resistance value.
2.00	DHA	1. Data Clock minimum value set to 2.4 MHz. 2. Added ADCINC_StartSample description to this user module datasheet.
3.00	HPHA	1. Removed ISR handler as the SPC interrupt is not posted to the M8C core. Users updating from a previous version of the user module should manually remove the ISR file. 2. Changed "Clock Divider" parameter default value to ‘3 (accy reduced-12 MHz)’. 3. Implemented limitation of returned value in wGetData and GetSample API functions. 4. Moved iGetData and cGetData functions to Legacy state.

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 © 2008-2013 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.