AN2096

PSoC® 1 - Using the ADCINC ADC

Author: David Van Ess, Arvind Krishnan
Associated Project: Yes
Associated Part Family: CY8C23x33, CY8C24x23A, CY8C24x94xx, CY8C27xxx
Software Version: PSoC Designer™ 5.4
Related Application Notes: AN2239

AN2096 steps the user through the necessary configuration and code required to use the PSoC 1 ADCINC Analog-to-Digital Converter (ADC). The ADCINC can be configured from 8 to 14 bits, and returns either an unsigned or signed (2's complement) output. Configuration of the ADCINC parameters and the Global Resources are covered to give the user a complete picture on creating an ADC project using PSoC Designer™. Example projects are provided in both C and assembly language. ADCINC is one of the many user modules available for analog to digital conversion. See AN2239 - PSoC® 1 - Selecting The Right ADC for other converters.

Introduction

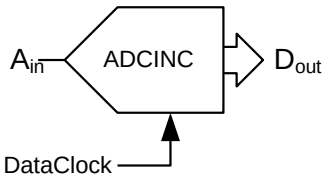
The ADCINC (or Incremental ADC) is one of the ADC user modules offered with PSoC Designer™ for the PSoC 1 family of parts. It uses one or two analog switched capacitor blocks and one digital block. To operate the ADCINC, you need to properly place the ADCINC and select the correct user module parameters. This application note assists you in the selection of these parameters.

The example project includes a Programmable Gain Amplifier (PGA) and 2-Line by 16-character LCD, along with the ADCINC, create a project that is both flexible and that makes it easy to evaluate the ADC. The PGA is added to provide a high input impedance to the signal and to give more flexibility to input signal routing. The 2-line by 16-character LCD is a convenient way to display the ADC results in an easily viewable format and only requires 3 lines of C code.

ADCINC: A Brief Description

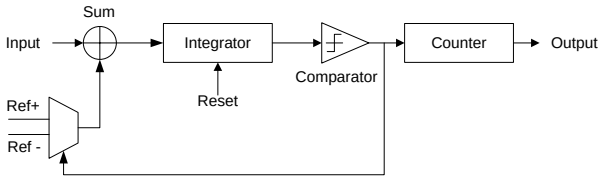
The following figure shows the ADCINC at its highest level of abstraction.

Figure 1. ADCINC Equivalent Circuit



The ADCINC is an averaging ADC that samples the analog input many times (driven by *DataClock*) to produce one *n*-bit digital output sample. The architecture is similar to that of a delta-sigma ADC, but operationally similar to an integrating ADC. The ADC works as an integrator which can be reset at the beginning of each conversion. Given below is a simplified diagram of the incremental ADC.

Figure 2. ADCINC Equivalent Circuit



Assuming the input is constant and positive, it is integrated until it is greater than the Reference value of the comparator. At this point, the Reference selection (*Ref*+ or *Ref*-) is made so that the signal is integrated back to zero. The value of the Counter is incremented when the comparator output is high. The process is repeated for 2^n cycles, to fetch a resolution of *n* bits. Before the next sample is converted, the integrator gets reset. The number of cycles in which comparator output is positive is counted to obtain the digital result.

The *SampleWindow* is the amount of time during which the input is integrated. The *SampleRate* is the number of consecutive samples that can be taken during one second. Equation 1 shows the relationship between *SampleRate* and *DataClock*. In this equation, *n* is the resolution setting of the user module.

The ADC resolution may be set between 6 and 14 bits. The ADC uses a Pulse Width Modulator (PWM) module to count the number of integration cycles. The PWM is set to provide an interrupt every 256 counts of the Data Clock. The counter integrates the signal for 2^{n-6} of these cycles. An additional cycle is required to reset the integrator and process the data.

$$SampleRate = \frac{DataClock}{256 \cdot (2^{n-6} + 1)} \qquad \text{Equation 1}$$

From Equation 1 it is clear that *SampleRate* is directly proportional to the input *DataClock* and is approximately inversely proportional to the resolution. The required resolution is inversely proportional to the maximum possible *SampleRate*.

The choice of resolution, *DataClock*, and *SampleRate* are determined by the particular application requirements. Refer to the following guidelines:

- The *DataClock* must always have a frequency less than or equal to the CPU clock. This is a strict requirement, if user code is added to the ADCINC's interrupt service routine.
- *DataClock* must be between 125 kHz and 8 MHz.

- The User Module specifications limit the *SampleRate* to a range of 125 sps to 31.25 ksps. (Exceeding these limits may produce undesirable results, and the resultant performance is not specified).

As stated earlier, the ADCINC is an averaging ADC. It averages many individual 1-bit samples to produce one *n*-bit result. Equation 2 shows this relationship. The resolution (*n*) of the ADC determines the number of samples taken to output one result. For example, an 8-bit resolution requires 256 individual samples.

$$SampleWindow = \frac{2^n}{DataClock} \qquad \text{Equation 2}$$

If the resolution of the ADCINC is 12 bits, a *DataClock* of 40,956 Hz can be used to create a 100 ms *SampleWindow*. This frequency is realized by dividing the 24 MHz System Clock (SysClk) by 586 (24 MHz / 40,956Hz). Equation 3 shows that this *DataClock* produces a *SampleWindow* of 100 ms.

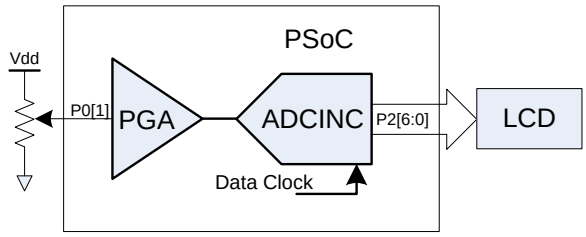
$$SampleWindow = \frac{2^{12}}{40{,}956} = 100\text{ms} \qquad \text{Equation 3}$$

A *SampleWindow* of 100 ms averages out any 50 Hz or 60 Hz line noise that may be on your signal by integrating over an integer number of AC line cycles. If noise or harmonic rejection is important for your particular application, select the ideal *SampleWindow* value with care.

Building a Project

Figure 3 shows the block diagram for the project to be developed.

Figure 3. Project Block Diagram



At a high level, this project is designed to measure the analog voltage on P0[1] pin and display the digital counts on the character LCD on Port2. This is done in the following stages:

- A potentiometer is used to supply a variable analog signal to GPIO P0[1].
- The input signal is buffered with the PGA and connected to the ADC.
- The ADCINC digitizes this signal and outputs it to the LCD connected to port 2. The LCD is a simple way to view the output of the ADC.

The fol-lowing walks you through the 10 steps to creating a project in either C or assembly code using the ADCINC User Module. You may follow the steps to create your own project or open either the C or assembly projects supplied with this application note.

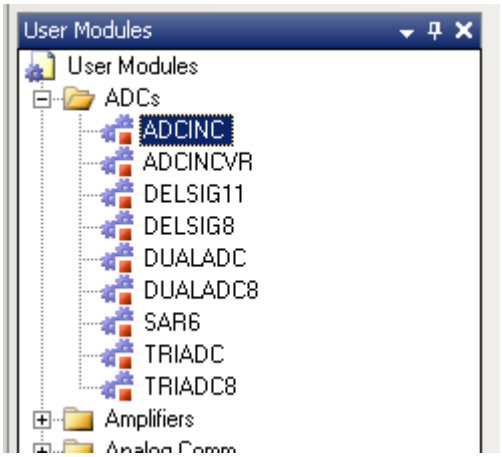
Step 1: Start the Design Tool

Start PsoC Designer and take the necessary steps to open a new project for a CY8C27443-24PXI device. Name this project "ADCINC_Demo." You can select either assembly or C as your programming language.

Step 2: Select and Place ADCINC

PsoC Designer starts a new project in Chip Editor view. In the User Modules window, open the ADCs directory. Next, open the ADCINC directory.

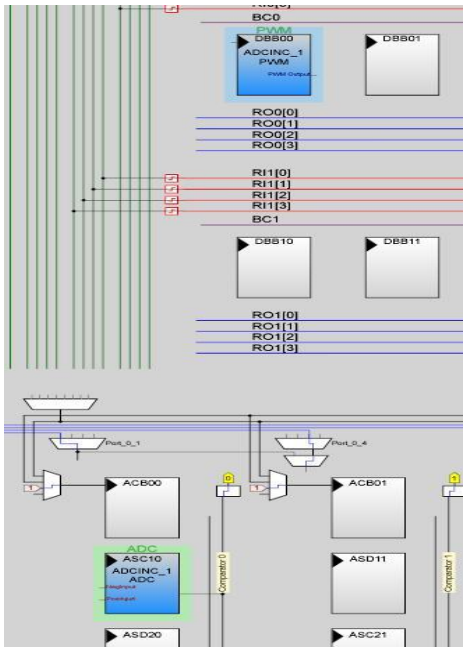
Figure 4. User Modules Window



Double-click the ADCINC icon shown in Figure 4. A new window appears. Choose the “Select Single Stage Modulator” option in this window and click OK. The user module is placed into the Chip Editor view in the first available blocks it can occupy.

This user module requires one digital block and one Switched Capacitor (SC) block. In the default placement for this user module, the digital block occupies DBB00 and the analog block occupies ASC10. Leave the blocks in their default placement. Figure 5 shows the Chip Editor view.

Figure 5. Default ADCINC Placement



Note that each block receives a name when the ADCINC is placed. They are:

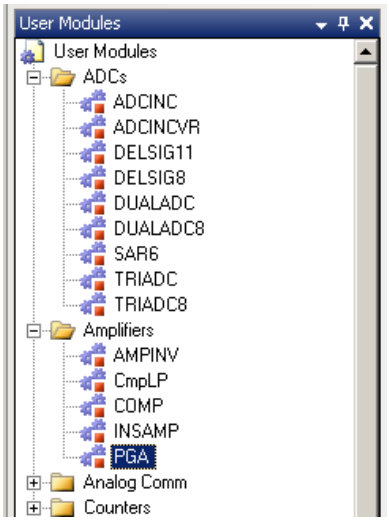
- PWM (in a digital block)
- ADC (in an analog block)

Step 3: Select and Place PGA

The programmable gain amplifier receives the input signal, buffers it, and the output is routed to the input of ADC.

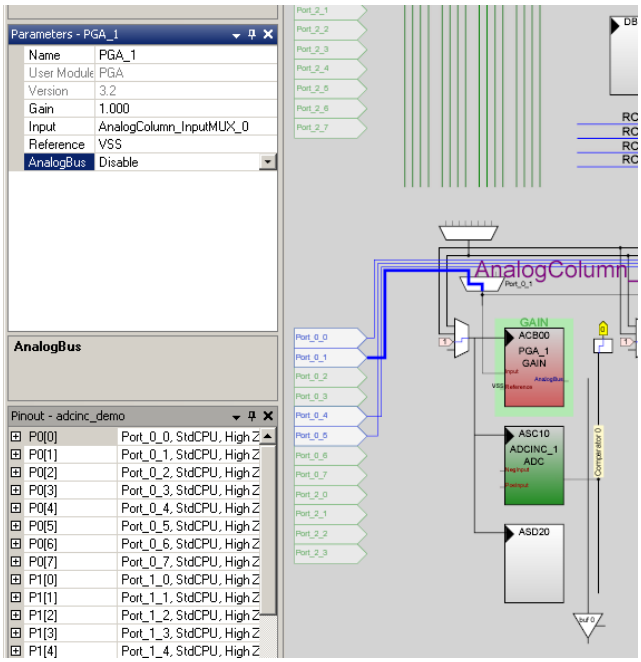
- To place the PGA, select it from the User Module window as you did with the ADCINC, by double clicking the PGA icon.

Figure 6. Selecting the PGA in User Module window



- By default it will be placed just above the ADC in the analog section.

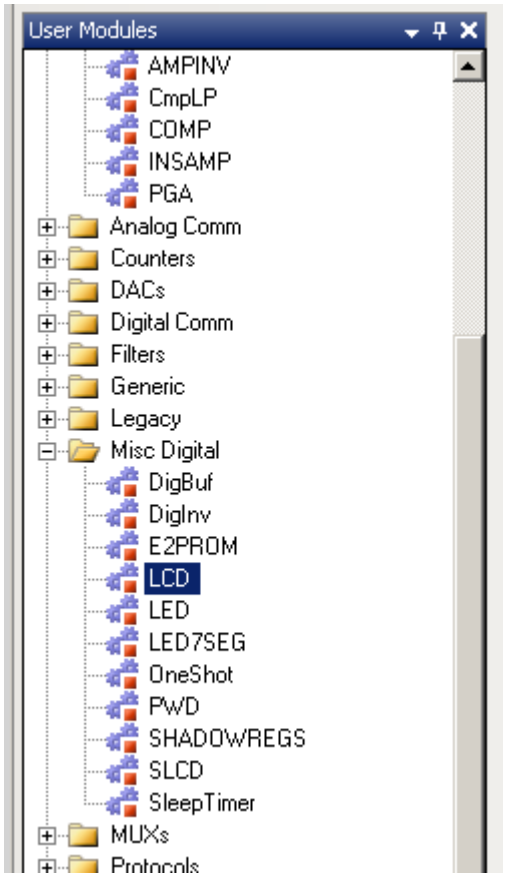
Figure 7. Placement of PGA



Step 4: Select and Place LCD

The LCD is very easy to add to a design. Simply select it as you did the ADCINC and PGA. It can be found in the User Module area under “Misc Digital”. As you did before double click the LCD icon and it will be placed in your design.

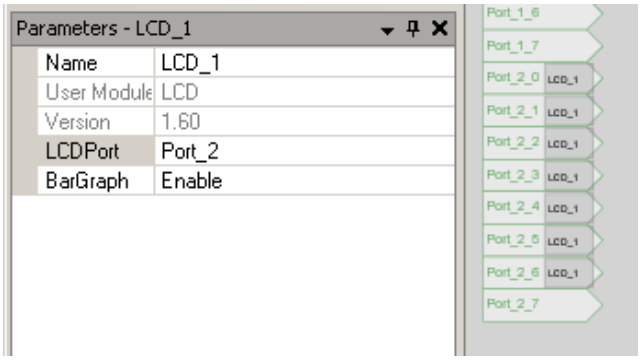
Figure 8. LCD Selection in User Module Window



If you are using the PSoCEVAL1, it has a dedicated connector (J9) for the LCD at the bottom of the board, no wiring required. When the LCD is plugged into this J9, it is connected directly to Port 2.

By clicking on the LCD_1 icon in the Workspace Explorer window, the LCD parameters will appear on the right.

Figure 9. LCD Parameters



Select “Port_2” for the “LCDPort” parameter. The BarGraph parameter does not matter for this project.

Step 5: Parameter and Resource Selection

The first group of properties to be selected is in the Global Resources window (at the top left side of the Chip Editor view by default). These parameters may affect more than just the ADCINC, so when you have multiple analog and digital blocks, pay attention to these global parameters to make sure the settings are compatible with all placed User Modules. This window is shown in Figure 10.

Figure 10. Global Resources Window

Global Resources - adcinc_demo

CPU_Clock	12_MHz (SysClk/2)
32K_Select	Internal
PLL_Mode	Disable
Sleep_Timer	512_Hz
VC1= SysClk/N	6
VC2= VC1/N	4
VC3 Source	VC2
VC3 Divider	256
SysClk Source	Internal 24_MHz
SysClk*2 Disable	No
Analog Power	SC On/Ref High
Ref Mux	(Vdd/2)+/-(Vdd/2)
AGndBypass	Disable
Op-Amp Bias	Low
A_Buff_Power	Low
SwitchModePum	OFF
Trip Voltage [LVI	4.81V (5.00V)
LVDThrottleBacl	Disable
Supply Voltage	5.0V
Watchdog Enab	Disable

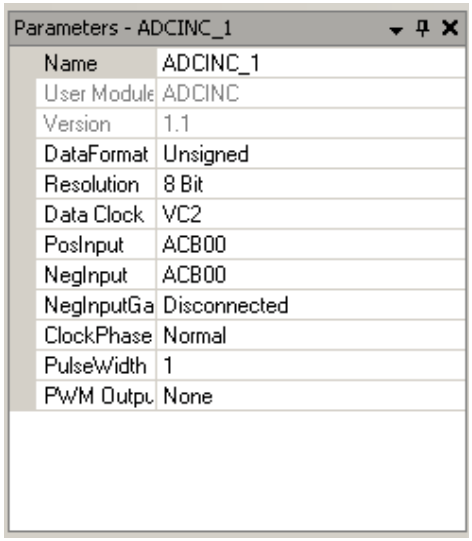
Set the Global Resources settings to match those in Figure 10. These settings are discussed here to understand them better.

- **CPU_Clock** is set to 12_MHz. This dictates the speed of the CPU. Make sure it is always as fast as or faster than the ADCINC clock (this is set using **Data Clock** parameter under ADCINC user module, which we will see in the next section).
- **32K_Select** is set to Internal because an external crystal is not used for this project.
- **PLL_Mode** is set to Disable. With no crystal, there is no input to the phase-locked loop (PLL) circuit.
- **Sleep_Timer** is left at 512_Hz. It is not used for this application.
- **VC1=SysClk/N** is set to 6. This sets the internal VC1 clock to 4 MHz (24 MHz / 6).
- **VC2=VC1/N** is set to 4. This sets the internal VC2 clock to 1 MHz (4 MHz / 4). Together with VC1, the system clock is divided by 24 (6 * 4) so that the ADC clock runs at 1 MHz.
- **VC3 Source** is left at VC2. The source for the VC3 internal clock is VC2. This clock divider is not used in this project.
- **VC3 Divider** is left at 256. This sets the internal VC3 clock to 24 MHz. VC3 is not used in this project.
- **SysClk Source** is Internal_24_MHz. The internal 24 MHz oscillator is used generate SysClk.
- **SysClk*2 Disable** is set to No. The 48 MHz clock is not disabled, even though it is not used in this project.
- **Analog Power** is set to SC On/Ref High. This turns on the clock to the SC blocks and sets the internal references for their best performance. The power level can be reduced after a project is successfully developed and evaluated.
- **Ref Mux** is set to (Vdd/2)+/-(Vdd/2). This sets the ADC to use 2.5 V for its zero reference (AGND) and measures signals in a range that is 2.5 V above and below AGND. This allows the ADC to measure the full range of voltage between 0 V and 5 V. This is one of the most important analog global parameters, since it affects almost all analog blocks. Care should be taken when setting this parameter when multiple analog blocks are placed in the design. It is highly recommended to experiment with other settings for this parameter once your project is operating as expected to understand the impact of this setting.
- **AgndBypass** is set to Disable. This is used to help reduce noise on the analog ground.
- **Opamp Bias** is set Low. Sets the bias current to all opamp.
- **A_Buff_Power** is set Low. This controls the power of the analog output buffers, which are not used in this design.
- **SwitchModePump** is not used and is set to OFF. This is enabled if the PsoC is powered by and external low voltage source such as a single cell battery.

- **Trip Voltage** is set to 4.81 V for 5 V operation. This sets the maximum low voltage threshold before to PSoC.
- **LVD ThrottleBack** is set to Disable. This option is used to slow down the processor at lower supply voltages.
- **Supply Voltage** is set to 5.0 V. This is set to reflect the operating voltage of the device.
- **Watchdog Enable** is set to Disable. Enable or disable watchdog timer, not used in this project.

Next, set the ADCINC parameters. Figure 11 shows the user module Parameter window for the ADCINC. Apply the settings shown in this figure to the ADCINC User Module.

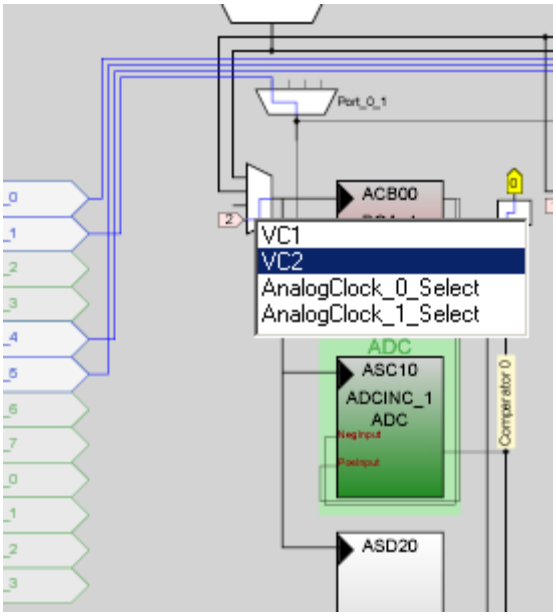
Figure 11. ADCINC User Module Settings



- **DataFormat** is set to Unsigned. The ADCINC User Module can output signed or unsigned samples.
- **Resolution** is set to 8 Bit. The ADCINC may be set between 8 and 14 bits of resolution.
- **Data Clock** is set to VC2. In the Global Resources window, VC2 is given a frequency of 1 MHz. This means the data clock for this ADCINC user module is 1 MHz.
- **PosInput** is selected to be ACB00. This configures the ADC input to connect directly to the output of the PGA in the analog block directly above the ADC.
- **NegInput** is set to ACB00. The negative input is not used in this project, so this setting is not important. The negative input is used to measure a differential voltage instead of assuming the reference voltage is Vss.
- **NegInputGain** is set to Disconnected. This disconnects the negative input, but is not used in the example project.
- **ClockPhase** is set to Normal. It must be set to Normal if the input comes from an input pin, CT Block, or filter. It generally must be set to Swapped only when the inputcomes from a SC block.
- **PulseWidth** is set to 1. This ADCINC User Module has a built in PWM output. It is not used in this project.
- **PWM Output** is set to None.

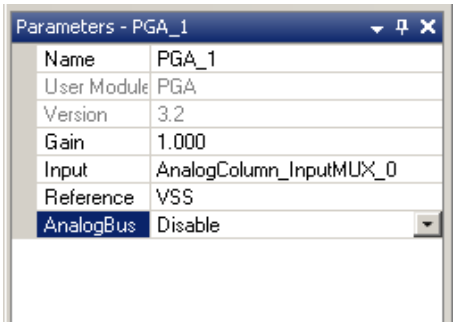
The same clock must be connected to both the analog “ADC” and digital “PWM” sections of the ADCINC. The clock used in this project is VC2. It was connected to the digital section in the ADCINC’s parameter window, but the clock connection to the analog block is a manual operation. As shown in Figure 12, click on the mux next to the left most analog column and select “VC2”. Failure to connect both blocks to the same clock will result in unexpected results.

Figure 12. Selecting Analog Column Clock



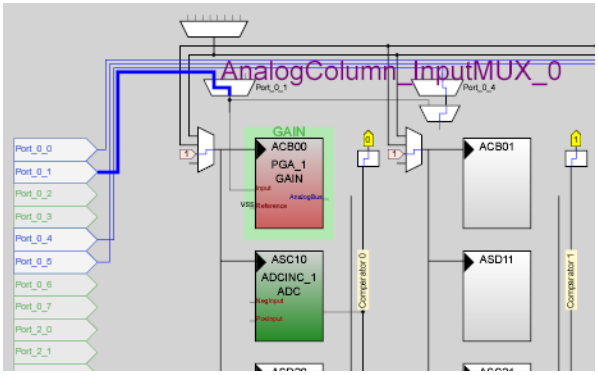
The PGA parameters are next.

Figure 13. PGA User Module Parameters



- **Gain** is set to 1.000. This parameter has several settings from 0.062 to 48. Feel free to experiment with different settings after the project is operating as expected.
- **Input** is set to AnalogColumn_InputMux_0. This mux can be connected to any of the odd pins on port 0 (P0[1], P0[3], P0[5], or P0[7]), simply by clicking on the mux in the diagram, see Figure 14. By default this will connect the input of the PGA to pin P0[1]. This can be changed by clicking on the mux symbol above the PGA symbol.

Figure 14. Input Mux

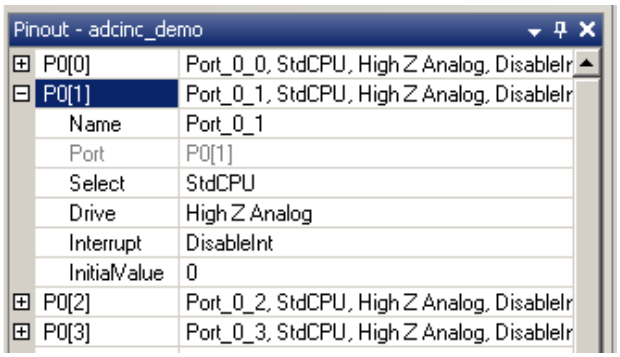


- **Reference** is set to “VSS”. This parameter selects the connection point of the PGA reference input.
- **AnalogBus** is set to Disable. This allows the user to route the output to the local analog bus and then down to the output buffer0.

Figure 14 shows the Pinout window. This window contains the settings for the GPIO pins. P0[1] by default is set as an analog input pin “High Z Analog”, so no changes are required.

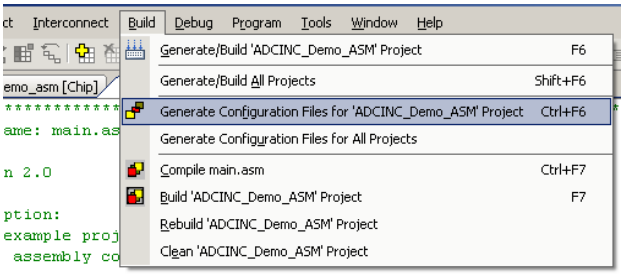
Step 6: Generate Application

Figure 15. Pinout Window with GPIO Settings



To generate the configuration files for this project, select the “Generate Configuration Files” option under the “Build” menu as shown in Figure 16. This will generate the system and user module source files.

Figure 16. Generate Configuration Files



Step 7: Edit Your Application Source

Once all the source files have been generated, you will initially want to add code to your *main.c* or *main.asm* file depending on which language you selected in Step 1. The code listing for each language is shown below. Both the C and Assembly version should operate the same. The sources for both these applications have been included in the example projects included with this application note.

Code 1. ADCINC_demo (C Code)

```
#include <m8c.h>
#include "PSoCAPI.h"

void main(void)
{
    unsigned char bADCData;

    /* Enable Interrupt for ADC  */
    M8C_EnableGInt ;

    /* Initialize all User Modules  */
    LCD_1_Start();
    PGA_1_Start(PGA_1_HIGHPOWER);
    ADCINC_1_Start(ADCINC_1_HIGHPOWER);

    /* Run ADC continuously  */
    ADCINC_1_GetSamples(0);
```

```
    while(1)  /* Loop forever  */
    {
        if(ADCINC_1_fIsDataAvailable() != 0)
        /* If data is valid display on LCD  */
        {
            /* Read ADC Result  */
            bADCData = ADCINC_1_bGetData();

            /* Display on LCD  */
            LCD_1_Position(0,0);
            LCD_1_PrHexByte(bADCData);

            /* Clear ADC flag for next reading */
            ADCINC_1_fClearFlag();
        }
```

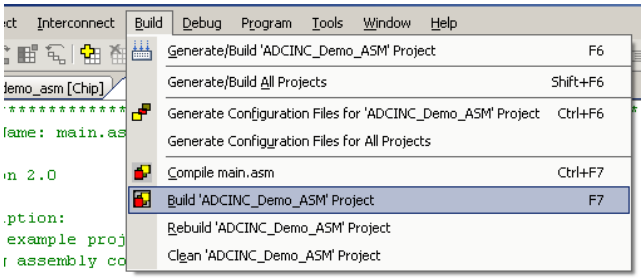
Code 2. ADCINC_Demo_ASM (Assembly Code)

```
include "m8c.inc"
include "memory.inc"
include "PSoCAPI.inc"

export _main

_main:

    M8C_EnableGInt ; Enable interrupts for ADC

    ; Initialize all User Modules
    mov   A,ADCINC_1_HIGHPOWER
    call  ADCINC_1_Start
    mov   A,PGA_1_HIGHPOWER
    call  PGA_1_Start
    call  LCD_1_Start

    ; Run ADC continuously
    mov   A,0
    call  ADCINC_1_GetSamples

Loop:  ; Loop forever
    ; If data is valid display on LCD
    call  ADCINC_1_fIsDataAvailable
    jz    ProgramStuff

    ; Position cursor at row 0, col 0 on LCD
    mov   A,0
    mov   X,0
    call  LCD_1_Position

    ; Get data and display on LCD
    call  ADCINC_1_bGetData
    call  LCD_1_PrHexByte

    ; Clear ADC flag for next reading
    call  ADCINC_1_fClearFlag

ProgramStuff:
    ; Put rest of program here

    jmp  Loop
```

Step 8: Compile your source code

Once you have added your code, you can compile the project by selecting the Build Project option in the “Build” menu as shown in Figure 17.

Figure 17. Build/Compile Project

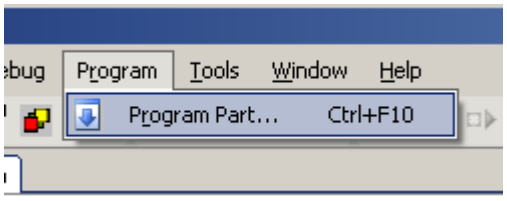


If you were careful your project should compile without errors. If you find an error, review the error messages, modify your source and rebuild your project until your project compiles without errors.

Step 9: Program the PSoC

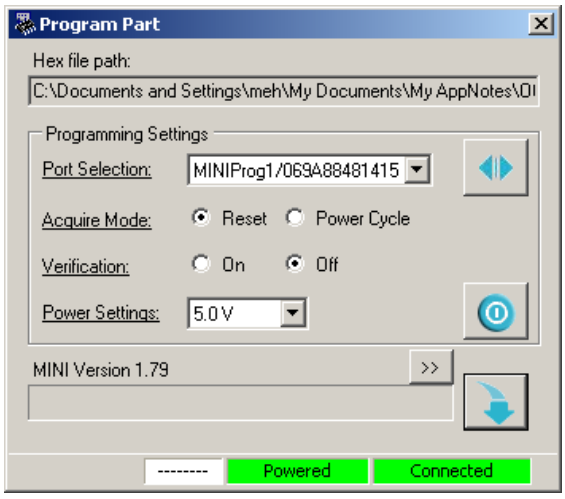
Now that your program compiles without errors, it is time to program your device. Make sure your project board is powered up and the programmer is attached to the programming connector. You may use either the Cypress PSoC MiniProg or MiniProg3 for programming. To start programming, select the “Program Part” item under the “Program” menu.

Figure 18. Programming Menu



The PSoC programmer dialog should appear as shown in Figure 19. If your board is self powered, make sure you select the “Reset” option for Acquire Mode.

Figure 19. PSoC Programmer Dialog

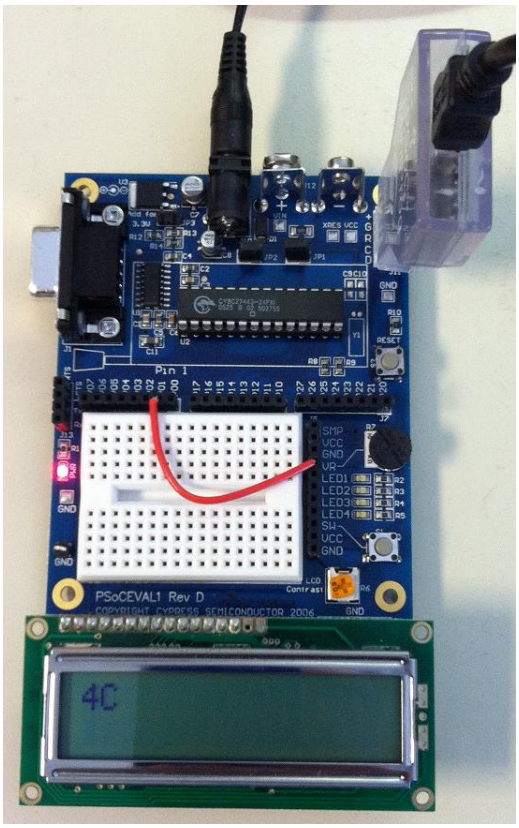


When ready, press the blue down arrow at the bottom right to start programming the device. Once the programmer has finished programming the device it will reset and start to execute your program.

Step 10: Program Validation

After programming you should see two hex characters in the upper left of the LCD. Make sure you have a wire connecting pin P0[1] to the potentiometer wiper (VR). If the display looks blank, check the contrast adjustment potentiometer above the LCD (R6). Use a screwdriver to adjust it until the characters appear on the display.

Figure 20. Example Hardware



Turn the voltage source potentiometer (R7) back and forth to see the value on the display go from 00 to FF (within an error of 2 decimal values).

Code Explanation

Each user module (ADCINC, PGA, LCD) have a set of function calls or API (Application Programming Interface). These functions have the same name whether you choose to use C or Assembly language. Every user module has a “Start” function that must be called to initialize the module before it will operate. These start functions are usually called at the beginning of the program.

```
/* Initialize all User Modules  */
LCD_1_Start();
PGA_1_Start(PGA_1_HIGHPOWER);
ADCINC_1_Start(ADCINC_1_HIGHPOWER);
```

Some of user modules require interrupts to operate properly. Most of the PSoC 1 ADCs including the ADCINC require interrupts as well. The following function enables global interrupts.

```
M8C_EnableGInt ; Enable interrupts for ADC
```

Once the ADCINC is initialized, the sampling process needs to be enabled. The function below can be used to enable between 1 and 255 samples then stop. If you want

the ADCINC to run continuously, the input parameter is set to zero, as shown below.

```
ADCINC_1_GetSamples(0);
```

While the ADCINC is operating, you need to check to see when a conversion has completed. The function below returns zero if the conversion is not yet complete and a non-zero value if the conversion is complete.

```
ADCINC_1_fIsDataAvailable();
```

When the conversion is complete, this function is used to return the result. It returns a single byte but the ADCINC includes functions that return integer (2-byte) values as well for ADC resolutions of 9 to 14 bits.

```
bADCData = ADCINC_1_bGetData();
```

After you read the ADC value, the ready flag needs to be reset. The following function is used to clear the ready flag to prepare for the next reading.

```
/* Clear ADC flag for next reading */
 ADCINC_1_fClearFlag();
```

The two lines below take the ADC result and display it on the LCD starting at row 0 column 0 in a hex format. The LCD has many other useful functions as well that are useful in displaying results or debugging your project.

```
/* Display on LCD  */
 LCD_1_Position(0,0);
LCD_1_PrHexByte(bADCData);
```

Summary

So that is it. As you can see, with a few lines of code you can continuously sample a voltage and display the result on a LCD. After you have read and understood this example, you will find that the other PSoC ADCs are just as easy to use as the ADCINC.

Don't be afraid to experiment with the Global and ADCINC parameters to develop a better understanding of their functionality. This project is a good vehicle to experiment with the PGA parameters and other LCD functions. The LCD BarGraph functions are fun to experiment with and can be used to graphically indicate a voltage or sensor magnitude.

Document History

Document Title: AN2096 - PSoC® 1 - Using the ADCINC ADC

Document Number: 001-41065

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	1536344	SSFTMP4	10/03/2007	New application note.
*A	2609454	BTK/PYRS	11/18/2008	Change in title and content to refer to and describe ADCINC User Module instead of ADCINC12 User Module. Update of text and figures to show latest version of PSoC Designer software. Minor grammatical changes.
*B	3272672	MEH	06/06/2011	Changed title and abstract. Added PGA and LCD to project. Changed code to accommodate LCD. Changed how code was presented. Changed and added several images
*C	3646967	ARVI	06/15/2012	Put into new application note template Minor changes to the document
*D	4422578	ARVI	06/27/2014	Updated example project to latest version of PSoC Designer (5.4)
*E	5688150	AESATMP8	04/19/2017	Updated logo and Copyright.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6

Cypress Developer Community

Forums | WICED IOT Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2007-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.