Application Note

ADC Compensation in Z8 Encore! XP® MCUs

AN028402-1208

Z8 Encore! XP® Flash Microcontrollers

Abstract

Zilog's Z8 Encore! XP® F082A Series features an optional 8-channel, 10-bit Analog-to-Digital Converter (ADC). The ADC converts an analog input signal to its digital representation. ADCs invariably exhibit error due to offset and non-unity gain. The Z8 Encore! XP Microcontroller Unit (MCU) can compensate for this error by providing factory-set values stored in the information page of the Flash Memory.

This application note describes a method of obtaining accurate ADC readings using the factory-set gain and offset calibration values. ADC calibration steps based on *Z8 Encore! XP F082A Series Product Specification (PS0228)* are implemented on Z8F04A28100KITG development board. HyperTerminal for display, and SINGLE-ENDED 1 x buffered with 2 V internal Vref ADC settings are used.

Note: *The source code file associated with this application note, (*`AN0284-SC01.zip`*) is available for download at* www.zilog.com.

Overview of Z8 Encore! XP F082A Series

Zilog's Z8 Encore!® MCU family of products are the first in a line of Zilog® microcontroller products based upon the 8-bit eZ8 CPU. Z8 Encore! XP F082A Series products expand upon Zilog's extensive line of 8-bit MCUs. The Flash in-circuit programming capability allows for faster development time and program changes in the field. The new eZ8 CPU is upward compatible with the existing Z8® instructions. The rich peripheral set of the Z8 Encore! XP F082A Series makes it suitable for a variety of applications including motor control, security systems, home appliances, personal electronic devices, and sensors.

Features

The key features of sigma-delta ADC include:

- 11-bit native resolution in DIFFERENTIAL mode
- 10-bit native resolution in SINGLE-ENDED mode
- Eight SINGLE-ENDED analog input sources are multiplexed with general-purpose input/output (GPIO) ports
- Ninth analog input obtained from temperature sensor peripheral
- 11 configurations of differential pairs are also multiplexed with GPIO ports
- Low-power operational amplifier (LPO)
- Interrupt on conversion complete
- Bandgap generated internal voltage reference with two selectable levels
- Manual in-circuit calibration for employing user code (offset calibration)
- Factory calibrated for in-circuit error compensation

Hardware Architecture

Figure 1 displays the major functional blocks of the ADC. An analog multiplexer network selects the ADC input from the available analog pins, ANA0 through ANA7. The input stage of the ADC allows differential gain and buffering. The following input options are available:

- Unbuffered input (SINGLE-ENDED and DIFFERENTIAL modes)
- Buffered input with unity gain (SINGLE-ENDED and DIFFERENTIAL modes)
- LPO amplifier output with full pin access to the feedback path

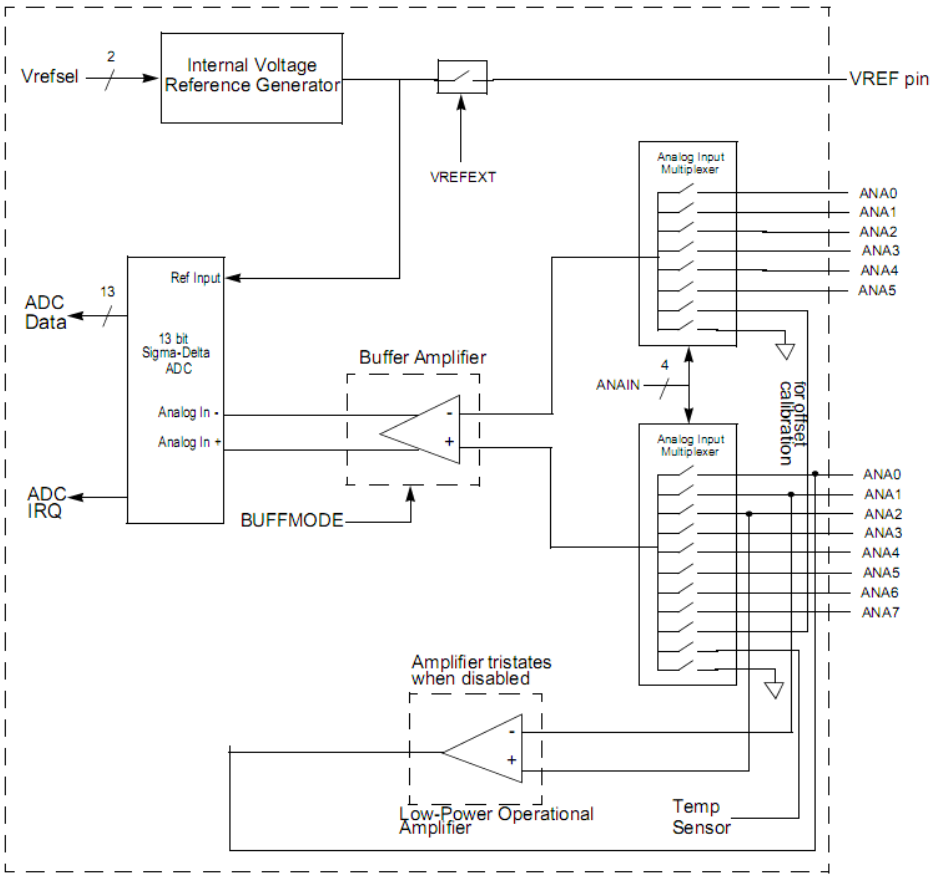

Figure 1. Analog-to-Digital Converter Block Diagram

In SINGLE-ENDED mode and DIFFERENTIAL mode, the output of the ADC is a 11-bit, signed, two's-complement digital value. In the SINGLE-ENDED mode, the ADC output generally ranges from 0 to +1023. In the DIFFERENTIAL mode, the ADC output values across the entire 11-bit range varying from -1024 to +1023. The ADC registers return 13 bits of data, of which two least-significant bits (lsb's) are reserved for the compensation of the ADC output. During compensation of the 13-bit raw ADC value, two bits of resolution are lost as a result of rounding error. Therefore, the final ADC output is 11-bit value.

Software Implementation (in C Language)

Reading Offset and Gain Calibration Bytes

The following code snippet (from `adc.c`) illustrates a method of reading the offset and gain calibration bytes from the Flash Information Area for a SINGLE-ENDED, 1 x buffered input with an internal reference voltage of 2 V. The purpose of this routine is to initialize the `ADC_offset`, `ADC_gain`, and `ADC_gainCal` variables that are used later in the ADC Compensation Algorithm. This routine only needs to be performed once.

```
unsigned int ADC_gainCal;
unsigned char ADC_offset;
unsigned int ADC_gain;
void ADC_calibration (void)
{

    rom unsigned char *ptr;
    ADC_gainCal = 0;

    FPS = 0x80;  // Enable access information area. set bit 7
    ptr = (rom unsigned char*)(0xFE69);  //offset correction byte
    ADC_offset = *ptr;
    ptr = (rom unsigned char*)(0xFE0E); //ADC_gain_hi_byte
    ADC_gain = *ptr;
    ADC_gain = ADC_gain <<8;
    ptr = (rom unsigned char*) (0xFE0F); //ADC_gain_lo_byte
    ADC_gain |= *ptr;
    ADC_gainCal = ADC_gain;
    FPS = 0x00;  //close information area
}
```

Reading Sampled ADC Bytes

The ADC is enabled to start conversion. After conversion is completed, ADC high byte data and low byte data are stored to `VInHigh` and `VInLow`, respectively, and combined as 16-bits in `ADC_data`. 12-bit ADC raw data (`ADCD_H [6.0]` and `ADCD_L [7:3]`) are extracted and remain in `ADC_data`.

```
ADCCTL0 |= 0x80;             //enable ADC
while (ADCCTL0 & 0x80);    //wait for conversion to be completed
VInHigh=ADCD_H;             //ADC high byte
VInLow=ADCD_L;              //ADC low  low byte
ADC_data = (VInHigh <<8)| (VInLow);  //ADC output word
ADC_data = (ADC_data >> 3) & 0xFFF;
```

ADC Compensation Algorithm

After getting the ADC_data, ADC offset byte, and ADC gain calibration byte, the ADC compensation process follows. This ADC compensation is based on the formula mentioned in the Theory of Operation on page 9.

For the ADC compensation process, first, ADC offset is tested if positive or negative. If it is negative, take the two's complement of it, then add to ADC_data. If ADC offset is positive, then ADC_data is subtracted by ADC offset. Second, ADC_gainCal is multiplied with ADC_data result from first step. After multiplication, it is rounded off by adding 32768 or 0x8000, then divided by 65536, or shifted by 16 bits position to the right. Finally, the result from last step is added to ADC_data from the first step, then shifted by 2 bits position to the right and took the 'AND' with 0x3FF to get the 10-bit ADC compensated data.

```
if ((ADC_offset & 0x80) == 0x80)
      {
      ADC_offset = ~ADC_offset + 1;
      ADC_data = ADC_data + ADC_offset;
      }
else
      {
       ADC_data = ADC_data - ADC_offset;
      }
ADC_product = ADC_data * ADC_gainCal;
ADC_product = (ADC_product + 0x8000)>> 16;
ADC_compensated = ((ADC_data + ADC_product)>>2)&0x03FF;
```

Software Implementation (in Assembly Language)

Reading Offset and Gain Calibration Bytes

The following code snippet (from adc_comp.asm) illustrates a method of reading the offset and gain calibration bytes from the Flash Information Area for a SINGLE-ENDED, 1x buffered input with an internal reference voltage of 2 V. The purpose of this routine is to initialize the adc_offset, lsb_gain, and msb_gain variables that are used later in the ADC Compensation Algorithm on page 6. Perform this routine once.

```
orx    FPS, #80h          ; enable FLASH page with
ld     ptr_h, #0feh       ; ADC offset FLASH address
ld     ptr_l, #69h        ; adc unbuf, 2v int vr offset @ FE69h
ldc    adc_offset, @ptr   ; load offset from flash
ld     ptr_l, #0Eh        ; adc unbuf, 2v int vr gain hi @ FE0Eh
ldc    msb_gain, @ptr     ; load gain hi cal byte from flash
ld     ptr_l, #0Fh        ; adc unbuf, 2v int vr gain lo @ FE0Fh
ldc    lsb_gain, @ptr     ; load gain lo cal byte from flash
ldx    FPS, #0             ; close FLASH page
```

Reading Sampled ADC Bytes

The ADC is enabled to start conversion. After conversion completed, ADC high byte data and low byte data are stored to `hi_adc` and `lo_adc`, respectively.

```
          orx ADCCTL0, #80h
convert:  tmx ADCCTL0, #80h
          jp nz, convert
          ldx hi_adc, ADCD_H
          ldx lo_adc, ADCD_L
```

The code below extracts the 12 bits of the ADC data (`ADCD_H` and `ADCD_L`). The top 4 bits are stored on `hi_adc` and the remaining 8 bits on `lo_adc`.

```
          ; lo_adc >>3
          ld r12, #3
here:     srl lo_adc
          djnz r12, here

          ; hi_adc << 5
          ld r11, hi_adc
          ld r12, #5
here1:    rcf
          rlc r11
          djnz r12, here1

          orx lo_adc, r11      ;8 bits ADC [b7:b0]

          ; hi_adc >>3
          ld r12, #3
here2:    srl hi_adc
          djnz r12, here2

          andx hi_adc,#0Fh    ; top 4 bits [b11:b8]
```

ADC Compensation Algorithm

After getting the ADC data (hi_adc and lo_adc), ADC offset byte and ADC gain calibration byte, the ADC compensation process follows. This ADC compensation is based on the formula mentioned in the Theory of Operation on page 9.

For the ADC compensation process, first, ADC offset is tested if positive or negative. If it is negative, take the two's complement of it, then add to ADC_data. If ADC offset is positive, ADC_data is subtracted by ADC offset.

```
          tm adc_offset, #80h
          jp z, sub_offset
          com adc_offset                    ;one's complement
          inc adc_offset                    ;two's complement
          add lo_adc, adc_offset
          adc hi_adc, #0
          jr multiply

sub_offset:rcf
          sub lo_adc, adc_offset
          sbc hi_adc, #0
```

The second step is to multiply the ADC gain and the ADC data. 16 bits by 16 bits multiplication in assembly can be done using the following procedures:

(a) Multiply lo_adc by lsb_gain, leaving the 16-bit result in r10 and r11.
(b) Multiply hi_adc by lsb_gain, adding the 16-bit result to r9 and r10.
(c) Multiply lo_adc by msb_gain, adding the 16-bit result to r9 and r10.
(d) Multiply hi_adc by msb_gain, adding the 16-bit result to r8 and r9.

Registers r14 and r15 are used in the operation as temporary registers.

Table 1 and Table 2 on page 7 lists an illustration of multiplication.

Table 1. Illustration of Multiplication

Operation	Byte 4	Byte 3	Byte 2	Byte 1
			hi_adc	lo_adc
*			msb_gain	lsb_gain
=				
			r10	r11
+		r9	r10	
+		r9	r10	
+	r8	r9		
=	r8	r9	r10	r11

Table 2. Multiplication Example

Operation	Byte 4	Byte 3	Byte 2	Byte 1
			03	BD
*			12	11
=				
			0C	8D
+		00	33	
+		0D	4A	
+	00	36		
=	00	43	89	8D

```
multiply: ld r10, lsb_gain
          ld r11, lo_adc
          mult rr10

          ld r14, hi_adc
          ld r15, lsb_gain
          mult rr14
          add r10, r15
          clr r9
          adc r9, r14
          clr r8
          adc r8,#00h

          ld r14, msb_gain
          ld r15, lo_adc
          mult rr14
          add r10, r15
          adc r9, r14
          adc r8, #00h

          ld r14, msb_gain
          ld r15, hi_adc
          mult rr14
          add r9, r15
          adc r8, r14

          add r10, #80h
          adc r9, #0
          adc r8, #0
```

The third step is adding 0x8000 to the product for rounding off. This is equivalent to adding 0x80 to r10, r9 add with carry to 0, and r8 add with carry to 0, as shown in the following table. Adding 0 to r11 is disregarded since it has no effect on the process.

	r8	r9	r10	r11
+	0x00	0x00	0x80	0x00

The fourth step is dividing the result by 2^{16}. In C language, this can be done by dividing the result with 65536 or by shifting 16 bits position to right. In assembly language, this is done by using the upper two bytes (r8 and r9) and disregarding the lower two bytes (r10 and r11).

The fifth step is adding the data of r8 and r9 to hi_adc and lo_adc, respectively.

```
          add lo_adc, r9
          adc hi_adc, r8
```

And finally, extract the lower 10 bits from the hi_adc and lo_adc to get the ADC compensated value.

```
          ; lo_adc >>2
          ld r12, #2
here3:    srl lo_adc
          djnz r12, here3

          ; hi_adc << 6
          ld r11, hi_adc
          ld r12, #6
here4:    rcf
          rlc r11
          djnz r12, here4

          orx lo_adc, r11      ; 8bits of ADC [b7:b0]

          ; hi_adc >>2
          ld r12, #2
here5:    srl hi_adc
          djnz r12, here5

          andx hi_adc,#03h    ; top 2bits [b9:b8]
```

Theory of Operation

The ADC generates an interrupt under one of the following three conditions:

1. After the analog-to-digital conversion is complete.
2. For samples greater than the programmable high threshold.
3. For samples lesser than the programmable low threshold.

The ADC High Threshold register (ADCTH) and the ADC Low Threshold register (ADCTL) are used to set the programmable high and low thresholds, respectively. The ADC interrupt must be set in the IRQ0 Enable High Bit register (IRQ0ENH) and the IRQ0 Enable Low Bit register (IRQ0ENL).

Note: *For detailed information about ADC interrupts, refer to the Z8 Encore! XP® F082A Series Product Specification (PS0228).*

ADC Compensation Algorithm

ADCcomp= (ADCuncomp — OFFCAL) + ((((ADCuncomp — OFFCAL) x GAINCAL) + 32768)/2^{16})

Where,

- ADCcomp is the compensated 10-bit ADC value
- ADCuncomp is the raw, uncompensated 12-bit ADC output value
- OFFCAL is a sign-extended 8-bit value stored in the Flash Memory
- GAINCAL is a sign-extended 16-bit value stored in the Flash Memory

The Z8 Encore! XP® MCUs are factory-tested to determine the unique gain and offset characteristics of a specific device. Error-correction coefficients are calculated using these test results, and stored in the information page of the Flash Memory. These correction coefficients are used in the compensation algorithm, and are not affected by the mass erasure of the program memory area.

Offset and gain error are not only influenced by the ADC characteristics, but also largely depend on the mode of operation of the ADC and the reference voltage levels. Therefore, separate error-correction coefficients are used for the different modes of operation.

As the information page of the Z8 Encore! XP MCU Flash Memory is of limited size, the ADC output is calibrated only for the 10 operational modes listed in Table 3 on page 10.

The error-correction coefficients are read from the information page by the software during the initialization of the ADC, and stored for future use. Each of the 10 supported modes of operation uses 1 byte of offset calibration and 2 bytes of gain calibration.

Table 3 lists ADC calibration modes and calibration data location.

Table 3. ADC Calibration Modes and Calibration Data location

Info Page Address	Memory Address	Compensation Usage	ADC Mode	Reference Type
60	FE60	Offset	SINGLE-ENDED Unbuffered	Internal 2.0 V
08	FE08	Gain High Byte	SINGLE-ENDED Unbuffered	Internal 2.0 V
09	FE09	Gain Low Byte	SINGLE-ENDED Unbuffered	Internal 2.0 V
63	FE63	Offset	SINGLE-ENDED Unbuffered	Internal 1.0 V
0A	FE0A	Gain High Byte	SINGLE-ENDED Unbuffered	Internal 1.0 V
0B	FE0B	Gain Low Byte	SINGLE-ENDED Unbuffered	Internal 1.0 V
66	FE66	Offset	SINGLE-ENDED Unbuffered	External 2.0 V
0C	FE0C	Gain High Byte	SINGLE-ENDED Unbuffered	External 2.0 V
0D	FE0D	Gain Low Byte	SINGLE-ENDED Unbuffered	External 2.0 V
69	FE69	Offset	SINGLE-ENDED 1x Buffered	Internal 2.0 V
0E	FE0E	Gain High Byte	SINGLE-ENDED 1x Buffered	Internal 2.0 V
0F	FE0F	Gain Low Byte	SINGLE-ENDED 1x Buffered	Internal 2.0 V
6C	FE6C	Offset	SINGLE-ENDED 1x Buffered	External 2.0 V
10	FE10	Gain High Byte	SINGLE-ENDED 1x Buffered	External 2.0 V
11	FE11	Gain Low Byte	SINGLE-ENDED 1x Buffered	External 2.0 V
6F	FE6F	Offset	DIFFERENTIAL Unbuffered	Internal 2.0 V
12	FE12	Positive Gain High Byte	DIFFERENTIAL Unbuffered	Internal 2.0 V
13	FE13	Positive Gain Low Byte	DIFFERENTIAL Unbuffered	Internal 2.0 V
30	FE30	Negative Gain High Byte	DIFFERENTIAL Unbuffered	Internal 2.0 V
31	FE31	Negative Gain Low Byte	DIFFERENTIAL Unbuffered	Internal 2.0 V
72	FE72	Offset	DIFFERENTIAL Unbuffered	Internal 1.0 V
14	FE14	Positive Gain High Byte	DIFFERENTIAL Unbuffered	Internal 1.0 V
15	FE15	Positive Gain Low Byte	DIFFERENTIAL Unbuffered	Internal 1.0 V
32	FE32	Negative Gain High Byte	DIFFERENTIAL Unbuffered	Internal 1.0 V
33	FE33	Negative Gain Low Byte	DIFFERENTIAL Unbuffered	Internal 1.0 V
75	FE75	Offset	DIFFERENTIAL Unbuffered	External 2.0 V

Table 3. ADC Calibration Modes and Calibration Data location (Continued)

Info Page Address	Memory Address	Compensation Usage	ADC Mode	Reference Type
16	FE16	Positive Gain High Byte	DIFFERENTIAL Unbuffered	External 2.0 V
17	FE17	Positive Gain Low Byte	DIFFERENTIAL Unbuffered	External 2.0 V
34	FE34	Negative Gain High Byte	DIFFERENTIAL Unbuffered	External 2.0 V
35	FE35	Negative Gain Low Byte	DIFFERENTIAL Unbuffered	External 2.0 V
78	FE78	Offset	DIFFERENTIAL 1 x Buffered	Internal 2.0 V
18	FE18	Positive Gain High Byte	DIFFERENTIAL 1 x Buffered	Internal 2.0 V
19	FE19	Positive Gain Low Byte	DIFFERENTIAL 1 x Buffered	Internal 2.0 V
36	FE36	Negative Gain High Byte	DIFFERENTIAL 1 x Buffered	Internal 2.0 V
37	FE37	Negative Gain Low Byte	DIFFERENTIAL 1 x Buffered	Internal 2.0 V
7B	FE7B	Offset	DIFFERENTIAL 1 x Buffered	External 2.0 V
1A	FE1A	Positive Gain High Byte	DIFFERENTIAL 1 x Buffered	External 2.0 V
1B	FE1B	Positive Gain Low Byte	DIFFERENTIAL 1 x Buffered	External 2.0 V
38	FE38	Negative Gain High Byte	DIFFERENTIAL 1 x Buffered	External 2.0 V
39	FE39	Negative Gain Low Byte	DIFFERENTIAL 1 x Buffered	External 2.0 V

Testing

This section describes the setup to test the ADC compensation procedure. The test results of the C language code and the Assembly code are also provided.

Setup

Figure 2 displays the hardware setup for ADC compensation procedure.

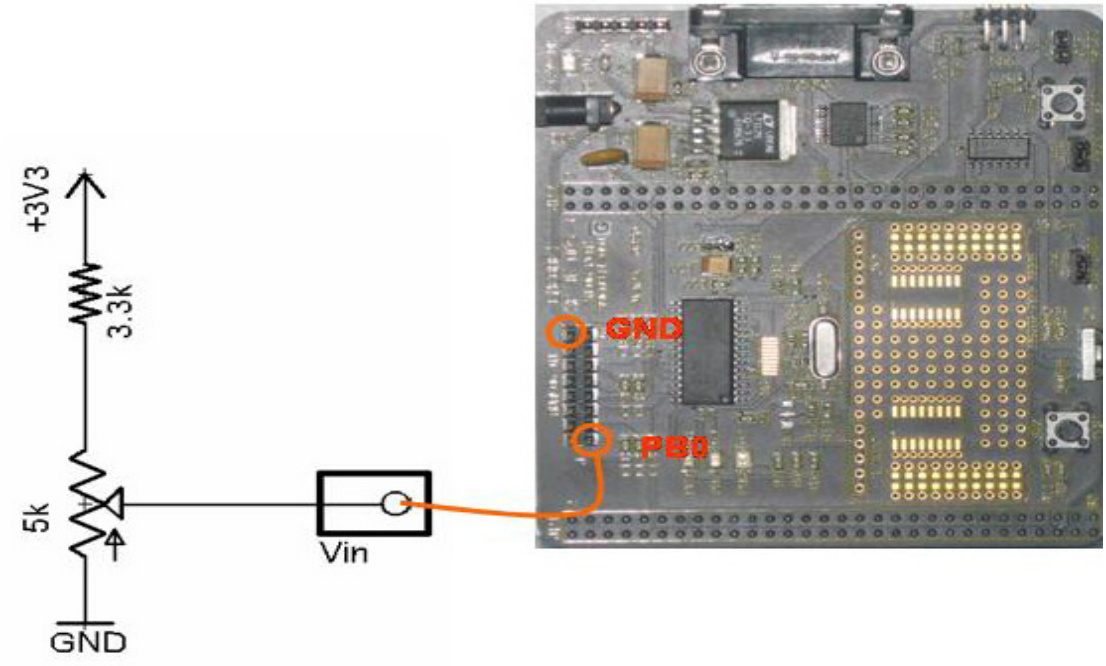

Figure 2. Vin Hardware Setup

For this application note, ADC is set as SINGLE-ENDED, 1 x buffered with 2 V internal voltage reference.

The Vin circuit with voltage divider network uses potentiometer to generate voltages as analog input of the MCU.

Results

For testing purposes, the ADC is operated in the SINGLE-ENDED, 1 x buffered mode. An internal reference voltage is used. The ADC offset byte is `0xE6` and the ADC calibration gain is `0x3BA5`.

Input voltage is measured using a FLUKE 89 IV True RMS multimeter with 3 decimal place accuracy.

Table 4 is a comparison of the input, ideal ADC value, and output ADC readings on C language and Assembly language.

Table 4. Comparison of ADC Uncompensated Value and ADC Compensated Value Using C Code and Assembly Code

Vin	Ideal = (Vin * 1023)/1.99	ADC Uncompensated		ADC Compensated	
		12-bits value	12-bits value/4	Assy	C code
0.1	51	153	38	55	55
0.2	103	306	77	102	102
0.3	154	470	118	153	153
0.4	206	638	160	204	204
0.5	257	804	201	255	255
0.6	308	969	242	306	306
0.7	360	1140	285	359	359
0.8	411	1305	326	410	410
0.9	463	1471	368	461	461
1	514	1639	410	513	513
1.1	565	1809	452	565	565
1.2	617	1977	494	617	617
1.3	668	2142	536	668	668
1.4	720	2307	577	719	719
1.5	771	2477	619	771	771
1.6	823	2643	661	822	822
1.7	874	2810	703	874	874
1.8	925	2975	744	925	925
1.9	977	3142	786	976	976
1.99	1023	3293	823	1023	1023

Figure 3 displays ADC compensated output value.

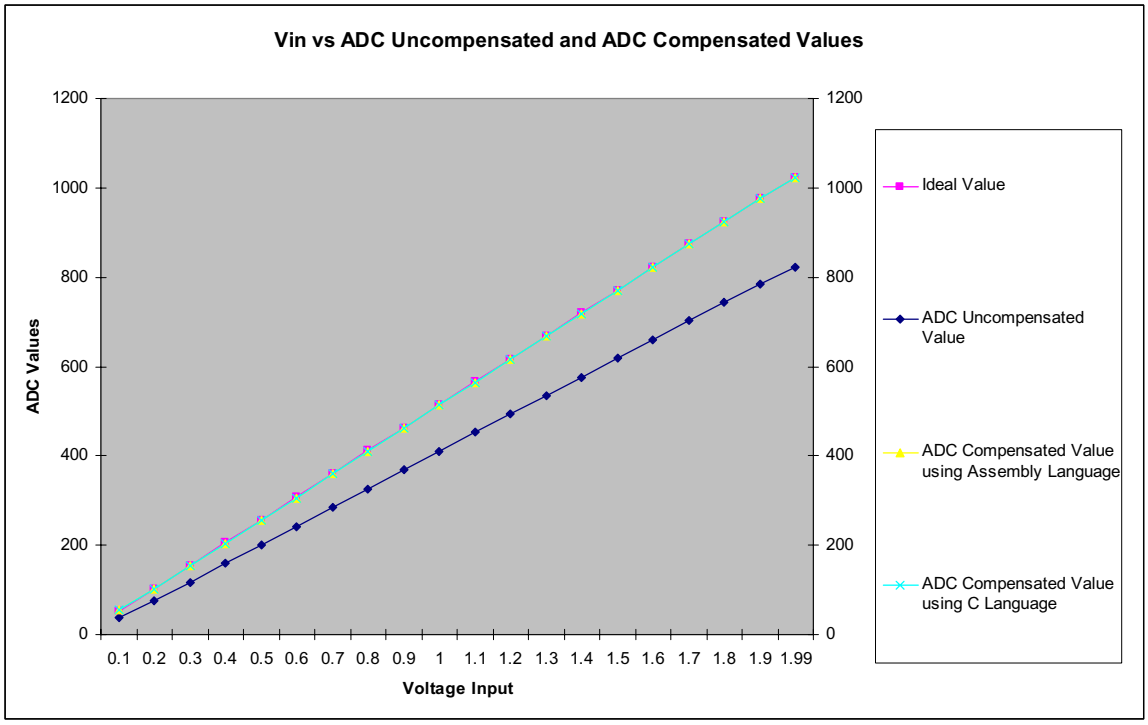

Figure 3. Vin vs. ADC Raw Data and ADC Compensated Data

Summary

This application note obtained the ADC readings using the factory set ADC gain and ADC offset calibration values. Results of the C language code and the Assembly code are equal to the ideal values.

References

The document associated with Z8 Encore! XP® MCU available on www.zilog.com is provided below:

- Z8 Encore! XP® F082A Series Product Specification (PS0228)

Appendix A—Glossary

Table 5 lists the abbreviations and definitions used in this application note.

Table 5. List of Abbreviations

Abbreviation	Definition
ADC	Analog-to-Digital Converter
GPIO	General-Purpose Input/Output
LPO	Low-Power Operational Amplifier
lsb	Least-Significant Bit
MCU	Microcontroller Unit

Warning: DO NOT USE IN LIFE SUPPORT

LIFE SUPPORT POLICY

ZILOG'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS PRIOR WRITTEN APPROVAL OF THE PRESIDENT AND GENERAL COUNSEL OF ZILOG CORPORATION.

As used herein

Life support devices or systems are devices which (a) are intended for surgical implant into the body, or (b) support or sustain life and whose failure to perform when properly used in accordance with instructions for use provided in the labeling can be reasonably expected to result in a significant injury to the user. A critical component is any component in a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

Document Disclaimer