Application Note

Configuring Analog-To-Digital APIs on the Z51F3220 MCU

AN034701-0313

Abstract

This application note demonstrates how to use the Analog-to-Digital Converter (ADC) functionality of the Z51F3220 MCU, a member of Zilog's Z8051 family of advanced CMOS 8-bit microcontrollers. Included is a discussion of the ADC's Polling and Interrupt modes of operation.

Application programming interfaces (APIs) are provided to initialize and begin using the Z51F3220 ADC.

Note: Two sample code sets using these ADC APIs are included in this application. One code set is compiled and tested using the Keil compiler; the other code set is compiled using the Small Device C Compiler (SDCC). Both of these codesets are contained in the AN0347-SC01.zip file, which is available free for download from the Zilog website. Subsequent releases of the Keil and SDCC tools may require you to modify the code supplied with this application note.

Discussion

The Z51F3220 MCU's ADC peripheral facilitates analog-to-digital data conversion through a successive approximation (SAR) method of converting an analog signal into 12-bit digital data. Analog input to this SAR ADC is received through a multiplexer capable of handling 16 analog input signals.

ADC Operation

The Z51F3220 ADC can be operated in two modes: polling or interrupt. In Polling Mode, after the ADC is initialized, A/D conversion will start; i.e., the ADST bit of the ADC Control Low Register is enabled. A/D conversion continues until the AFLAG bit (bit 4 of the ADCCRL Register) goes to a logic 1. The ADCDRH and ADCDRL registers contain the digitally-converted ADC data; Figure 1 illustrates ADC operation while in Polling Mode.

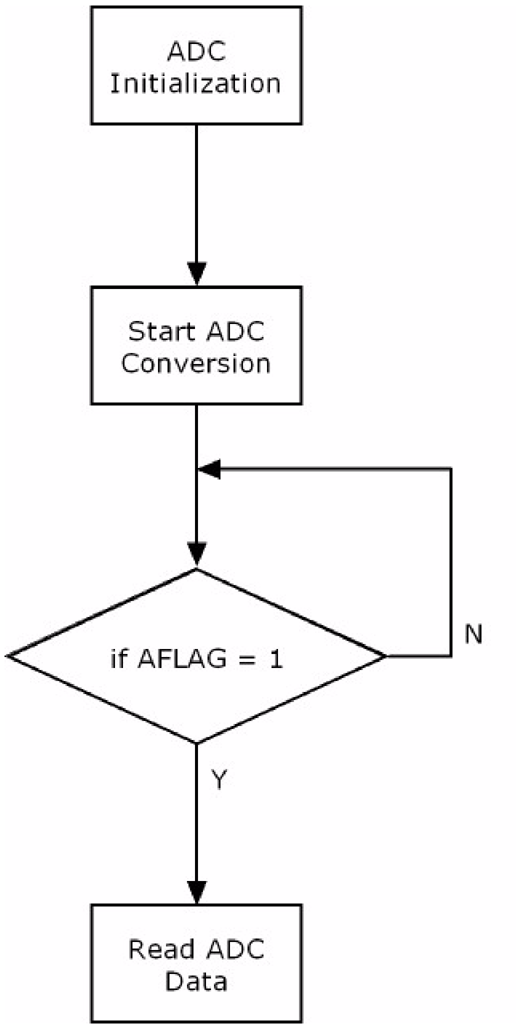


Figure 1. ADC Polling Mode Operation

In Interrupt Mode, after the ADC is initialized, A/D conversion will start, and interrupts will be enabled. When the A/D conversion is complete, an interrupt will be generated. A generated interrupt indicates that ADC data on ADCDRH and ADCDRL is available for reading. Figure 2 illustrates this operation.

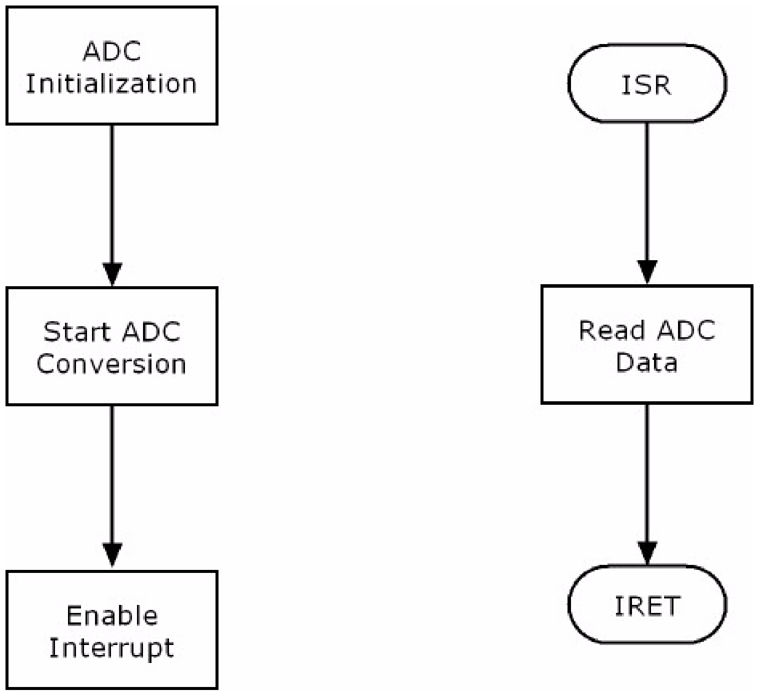


Figure 2. ADC Interrupt Mode Operation

ADC Initialization

Table 1 lists the functions that must be initialized prior to using the ADC.

Table 1. Initialized ADC Functions

Function	Register	Bit Position	Description
ADC Interrupt Generation	ADCCRH	bit7	Configure ADC operation mode: polling or interrupt.
ADC Trigger		bit[5:3]	Select ADC trigger signal.
ADC Align		bit2	Select ADC data alignment: MSB or LSB.
ADC Clock		bit[1:0]	Select ADC clock source to meet the minimum 20µs AD conversion time.
ADC Reference Voltage	ADCCRL	bit5	Select ADC reference voltage: internal or external.
Interrupt Priority	IP	bit0	If the ADC is operating in Interrupt Mode, select ADC interrupt priority.
	IP1	bit0	
Interrupt Enable Register	IE3	bit0	If the ADC is operating in Interrupt Mode, enable the ADC interrupt.

ADC Timing Requirements

The Z51F3220 MCU's ADC process requires 58 clock cycles to complete a 12-bit conversion. The first 10 clock cycles set up the ADC block; the next 4 clock cycles convert each bit, or 48 clock cycles, for a 12-bit ADC. Figure 3 illustrates these timing requirements.

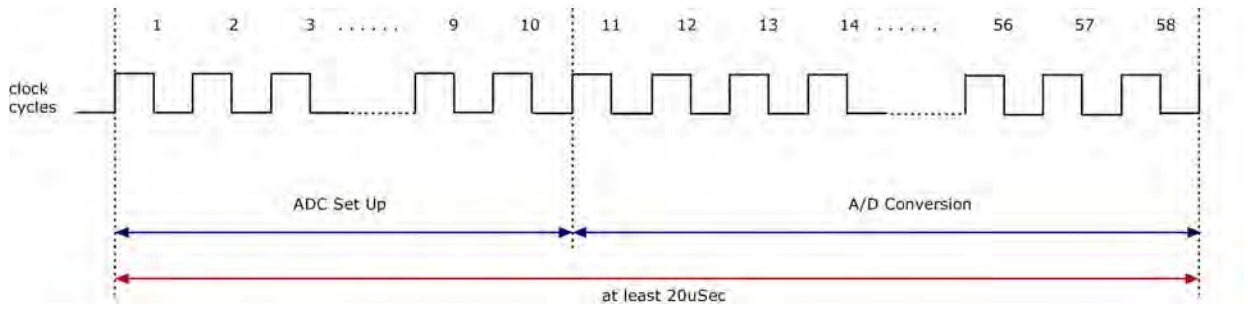


Figure 3. ADC Timing

These 58 clock cycles for the A/D conversion must be completed within 20 µsec, at minimum. The A/D conversion time is controlled by setting the ADC clock value through CKSEL[1:0] of the ADC Control High Register (ADCCRH). The duration for these 58 clock cycles can be determined using the following equation:

$$\text{Time (58 clock cycles)} = \frac{1}{\text{ADC\_Clock\_Value}} * 58$$

In this equation, the `ADC_Clock_Value` variable employs a system clock frequency divided by 1, 2, 4 or 8. When computing this equation, ensure that the duration for these 58 clock cycles is at least 20 µsec.

For example, if a 16 MHz system clock (f_X) is used, the minimum required ADC clock time of 20 µsec can be achieved by using an ADC clock value of $f_X \div 8$, as indicated in Table 2.

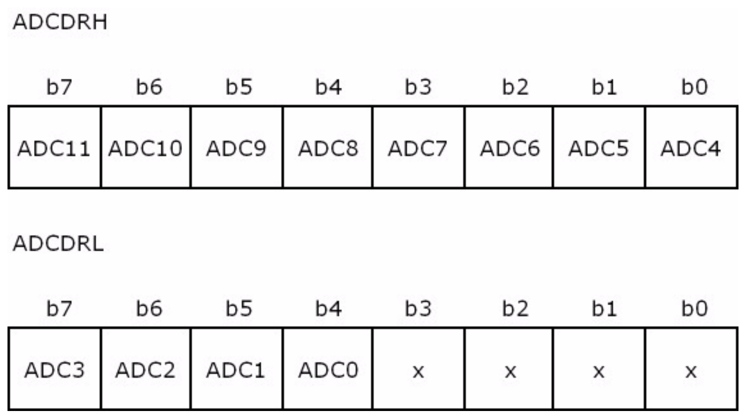
Table 2. ADC Clock Selector

CKSEL1	CKSEL0	ADC Clock Value	Time in µsec (58 ADC Clock Cycles @ 16 MHz SYSCLK)
0	0	$f_X \div 1$	3.625
0	1	$f_X \div 2$	7.25
1	0	$f_X \div 4$	14.5
1	1	$f_X \div 8$	29

ADC Alignment

After A/D conversion is complete, the 12-bit ADC data is available in the ADC Data Register High (ADCDRH) and ADC Data Register Low (ADCDRL). The data in these registers can be formatted to either MSB align or LSB align. To set this ADC data as MSB align, bit 2 of ADCCRH should be a logic 0. If bit 2 of ADCCRH is a logic 1, the ADC data is set as LSB align.

Figures 4 and 5 illustrate the alignment of 12-bit ADC data (ADC[11:0]) in the ADCDRH and ADCDRL registers.



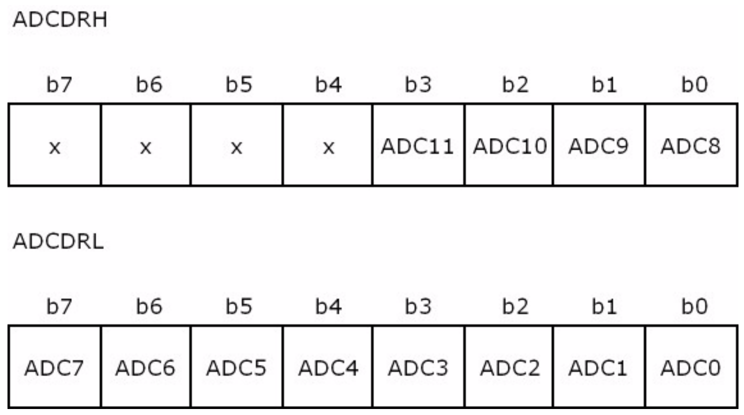
ADCDRH

b7	b6	b5	b4	b3	b2	b1	b0
ADC11	ADC10	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4

ADCDRL

b7	b6	b5	b4	b3	b2	b1	b0
ADC3	ADC2	ADC1	ADC0	x	x	x	x

Figure 4. ADC Data MSB Align



ADCDRH

b7	b6	b5	b4	b3	b2	b1	b0
x	x	x	x	ADC11	ADC10	ADC9	ADC8

ADCDRL

b7	b6	b5	b4	b3	b2	b1	b0
ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0

Figure 5. ADC Data LSB Align

Firmware Design

Only two files need to be included when using the Z51F3220 MCU's ADC for your project: `ADC_API.c` and `ADC_API.h`; the `ADC_API.h` file contains all of the definitions and prototypes for `ADC_API.c`. The default values for the ADC trigger signal, ADC clock, and ADC reference voltage are also provided in the `ADC_API.h` file; these default values can be changed by the user.

Additionally, the `ADC_API.h` file contains the `#define USE_ADC_INTERRUPT` and `#define USE_ADC_ALIGN_LSB` preprocessors. The `#define USE_ADC_INTERRUPT` statement should be removed if the user operates the ADC in Polling Mode, but should be included if the ADC will operate in Interrupt Mode.

Inside the `ADC_Init` file, the following lines of code will only be executed when the ADC Interrupt Mode of operation is selected:

```
ADCCRH |= ADC_INT;  // ADC control register ADCFIR: interrupt generation
IP |= 0x01;         // Set priority
IP1 |= 0x01;
IE3 |= INT18E;    // Interrupt Enable Register
```

The `#define` statement `USE_ADC_ALIGN_LSB` should be removed if the user prefers the ADC data MSB alignment format; it should be included if the user prefers the ADC data LSB alignment format. Additionally, one of the objectives of the `ADC_Init` function is to execute the following line of code if the ADC data LSB alignment format is selected.

```
ADCCRH = ui8ADC_TRIG | ui8ADC_CKSEL | ADC_ALIGN_LSB;
```

However, if the ADC data MSB alignment format is selected, the following line of code will be executed instead:

```
ADCCRH = ui8ADC_TRIG | ui8ADC_CKSEL;
```

Note: The default setting for the ADC is Polling Mode; the default ADC data alignment format is MSB.

Available APIs

The ADC_Init function accepts three parameters from the user: ADC Trigger Signal, ADC Clock and ADC Reference voltage source, as described below.

ADC Trigger Signal. Indicates the source of events that will trigger to start A/D conversion. The default value is the ADST (bit 6 of the ADCCRL Register), meaning that setting the ADST bit will start an A/D conversion.

The other five trigger signal sources to start an A/D conversion are:

- Timer 1 A match signal
- Timer 4 overflow event signal
- Timer 4 A match event signal

- Timer 4 B match event signal
- Timer 4 C match event signal

An example of how these five trigger signal sources are used is shown below.

```
// ADC trigger signal selection
#define ADC_TRIG_ADST   0x00    // ADST
#define ADC_TRIG_T1A    0x08    // Timer1 A match signal
#define ADC_TRIG_T4OV   0x10    // Timer4 overflow event signal
#define ADC_TRIG_T4A    0x18    // Timer4 A match event signal
#define ADC_TRIG_T4B    0x20    // Timer4 B match event signal
#define ADC_TRIG_T4C    0x28    // Timer4 C match event signal
```

ADC Clock. Required to configure to achieve the required A/D conversion time of at least 20 µs. The choices are: $f_X \div 1$, $f_X \div 2$, $f_X \div 4$ and $f_X \div 8$, in which f_X is the system clock frequency; see the ADC Timing Requirements section on page 4 for a more thorough description.

An example of these ADC Clock selections is shown below.

```
// ADC clock selection
#define ADC_CKSEL_FX1   0x00    // fX ÷ 1
#define ADC_CKSEL_FX2   0x01    // fX ÷ 2
#define ADC_CKSEL_FX4   0x02    // fX ÷ 4
#define ADC_CKSEL_FX8   0x03    // fX ÷ 8
```

ADC Reference Voltage. The internal reference voltage is equivalent to the V_{DD} value, while the external reference voltage is equivalent to the voltage value on the AV_{REF} pin, as shown in the following code example.

```
// ADC reference voltage selection
#define ADC_REFSEL_INT  0x00    // Internal reference (VDD)
#define ADC_REFSEL_EXT  0x20    // External reference (AVREF)
```

The following example shows how to initialize the ADC:

```
ADC_Init(ADC_TRIG_ADST, ADC_CKSEL_FX8, ADC_REFSEL_INT);
```

ADC_Start. This function accepts one analog input channel and starts A/D conversion of the signal on that channel. There are 16 analog input channels available for the Z51F3220 MCU, and each can be selected using this function.

One purpose of the `ADC_Start` function is to enable the ADC module, set the port as an analog input channel, and start an A/D conversion; Table 3 indicates these three ADC Start functions.

Table 3. ADC Start Functions

Function	Register	Bit Position	Description
ADC Module Enable	ADCCRL	bit7	Enable the ADC.
ADC Start Conversion		bit6	Start A/D conversion.
ADC Analog Input		bit[3:0]	Select the ADC input channel.
Port Analog Function	P0FSRL	bit[6:1]	Select the Port0 as analog input, ANA[2:0].
	P0FSRH	bit[5:0]	Select the Port0 as analog input, ANA[5:3].
	P1FSRH	bit[7:0]	Select the Port1 as analog input, ANA[9:6].
	P1FSRL	bit[7:0]	Select the Port1 as analog input, ANA[13:10].
	P2FSRL	bit[3:0]	Select the Port2 as analog input, ANA[15:14].

The following line of code is an example of how to start an A/D conversion of the signal on analog input AN0:

```
ADC_Start (ADC_ANA0);
```

Macros for all of the analog input channels are listed below, and are available in the `ADC_API.h` function.

```
// ADC channel input selection
#define ADC_ANA0        0x00    // Analog input 0
#define ADC_ANA1        0x01    // Analog input 1
#define ADC_ANA2        0x02    // Analog input 2
#define ADC_ANA3        0x03    // Analog input 3
#define ADC_ANA4        0x04    // Analog input 4
#define ADC_ANA5        0x05    // Analog input 5
#define ADC_ANA6        0x06    // Analog input 6
#define ADC_ANA7        0x07    // Analog input 7
#define ADC_ANA8        0x08    // Analog input 8
#define ADC_ANA9        0x09    // Analog input 9
#define ADC_ANA10       0x0A    // Analog input 10
#define ADC_ANA11       0x0B    // Analog input 11
#define ADC_ANA12       0x0C    // Analog input 12
#define ADC_ANA13       0x0D    // Analog input 13
#define ADC_ANA14       0x0E    // Analog input 14
#define ADC_ANA15       0x0F    // Analog input 15
```

ADC_GetData. This function returns 12-bit ADC data from the ADCDRH and ADCDRL registers if the A/D conversion is completed. If the A/D conversion is ongoing, it returns `0xFFFF`.

The following line of code is an example of how to use the `ADC_GetData` function:

```
ADCVariableName = ADC_GetData();
```

Note: The `ADC_GetData` function returns the data from the last analog input channel that was passed to the `ADC_Start` function.

Sample Application Code

Sample application code to demonstrate the use of the ADC APIs is included with this application. This application code reads ADC values and displays them to the HyperTerminal console.

In addition to the `adc_api.c` file, there are two additional files used for the demonstration, `main.c` and `uart.c`. The function of `main.c` is to initialize all of the peripherals to be used (i.e., the oscillator, the ADC, the UART and the timer) to read the ADC and display the ADC values to the HyperTerminal console through UART0.

One important segment of the `main.c` code appears below.

```
#ifdef USE_ADC_TIMER1_MATCH   // if Timer1A match as ADC trigger
ADC_Init(ADC_TRIG_T1A, ADC_CKSEL_DEF, ADC_REFSEL_DEF);
TIMER1_Init();
#else
ADC_Init(ADC_TRIG_DEF, ADC_CKSEL_DEF, ADC_REFSEL_DEF);
#endif                        // USE_ADC_TIMER1_MATCH
```

If `USE_ADC_TIMER1_MATCH` is defined (i.e., located in the `adc_api.h` file), the ADC triggering signal is the Timer1 A match. As a result, the ADC executes a conversion when the Timer1 period expires.

By default, `USE_ADC_TIMER1_MATCH` is not defined. Within this setup, ADST is the ADC trigger signal source.

Equipment Used

The tools used to create and test this application are:

- Keil Compiler µVision, version 4.24.00.0
- SDCC Compiler, version 3.1.0
- Z51F3220 Development Kit
- Z8051 On-Chip Debugger, version 1.147
- Z8051 On-Chip Debugger cable
- HyperTerminal or any equivalent communications/terminal emulation program

Testing/Demonstrating the Application

This application includes sample code that shows how to use the API; this sample code is contained in the `main.c` file. There are two methods for compiling and testing this code:

- Use the Keil compiler for compiling and debugging
- Use the SDCC compiler for compiling the code, plus use Zilog's Z8051 OCD software to download a hex file to the Z8051 MCU

The remainder of this section lists procedures for enabling these two methods.

Test The Code Using The Keil Compiler

Observe the following steps to test the code using the Keil compiler.

1. Connect the potentiometers, as indicated in Figure 6, to the Z51F3220 Development Board.

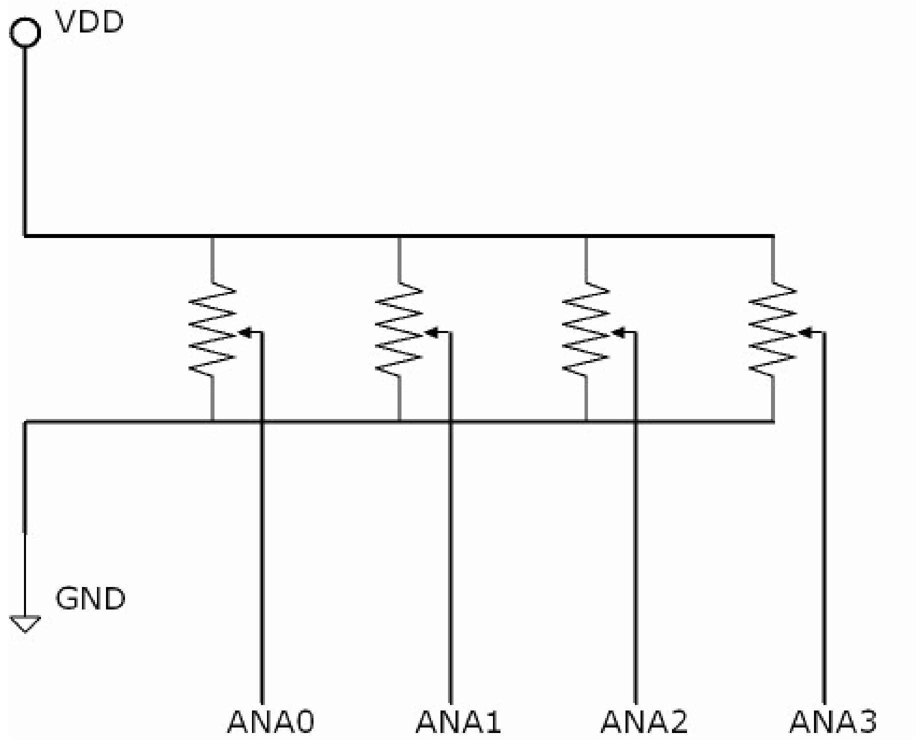


Figure 6. Potentiometers as Analog Input Sources

2. Connect the Z51F3220 Development Board to a PC using the Z8051 On-Chip Debugger cable.

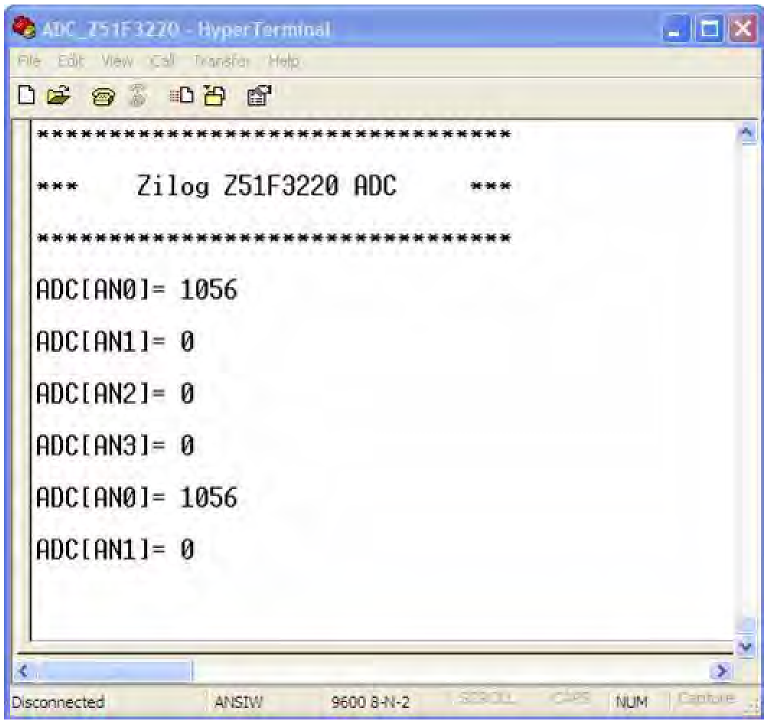
3. Connect one end of the USB A (male) to Mini-B cable (which is contained in the Z51F3220 Development Kit) to the Z51F3220 Development Board, and connect the other end of this cable to the PC's USB port. The indicator LED near the USB Mini port of the Board should illuminate, indicating that the Board is powered up.
4. Check to determine if the Header J16 jumpers at pins 1–2 and 3–4 of the Development Board are in place. These jumpers will connect the UART0 TXD and RXD pins to the FTDI UART-to-USB chip.
5. Download and unzip the AN0347-SC01.zip file to a convenient location on your PC.
6. Launch the Keil compiler. Select **Open Project** from the Keil **Project** menu, and locate the AN0347-SC01 folder from where you saved it on your PC. Open the `ADC_Z51F3220.uvproj` project, which is located in the following filepath:

   <PC Directory>\AN0347-SC01\ADC_8051_Keil

Note: In this path, <PC Directory> is the location of the unzipped AN0347-SC01 file.

7. In the application software, the UART is set to communicate via the following parameters:
   - 9600 baud
   - 8 bits data frame
   - No parity bits
   - 2 stop bits

   Configure HyperTerminal (or any equivalent terminal emulation program) to these same values.
8. From the **Project** menu, select **Rebuild all target files** to recompile all project files.
9. From the **Flash** menu, select **Download** to download the code to the MCU.
10. From the **Debug** menu, select **Start/Stop Debug Session** to start debugging. Figure 7 shows an example of the initial HyperTerminal console display.


ADC_Z51F3220 - HyperTerminal
File Edit View Call Transfer Help
Disconnected ANSIW 9600 8-N-2 SCROLL CAPS NUM Capture

```
*********************************

***    Zilog Z51F3220 ADC    ***

*********************************

ADC[AN0]= 1056

ADC[AN1]= 0

ADC[AN2]= 0

ADC[AN3]= 0

ADC[AN0]= 1056

ADC[AN1]= 0
```

Figure 7. An Example Keil Compiler HyperTerminal Display

11. Rotate the shafts of the potentiometers and observe the changing of the ADC values displayed in the HyperTerminal console.

Note: Refer to the Results section on page 15 for the ADC equivalent value of an analog input voltage.

Test The Code Using The SDCC Compiler

Observe the following steps to compile the code using the SDCC compiler and download a hex file to the Z51F3220 MCU using the Z8051 OCD software.

Note: Debugging is not possible when using the SDCC compiler. The following procedure is intended merely as a guide to compiling and downloading the code to the MCU using the SDCC compiler.

1. Connect the potentiometers, as indicated in Figure 6, to the Z51F3220 Development Board.

2. Connect the Z51F3220 Development Board to a PC using the Z8051 On-Chip Debugger cable.
3. Connect one end of the USB A (male) to Mini-B cable (which is contained in the Z51F3220 Development Kit) to the Z51F3220 Development Board, and the other end of this cable to the PC's USB port. The indicator LED near the USB Mini port of the Board should illuminate, indicating that the Board is powered up.
4. Check to determine if the Header J16 jumpers at pins 1–2 and 3–4 of the Development Board are in place. These jumpers will connect the UART0 TXD and RXD pins to the FTDI UART-to-USB chip.
5. Download and unzip the AN0347-SC01.zip file to a convenient location on your PC.
6. Navigate to the following filepath and locate the `ADC_8051.bat` file.

   <PC Directory>\AN0347-SC01\ADC_8051_SDCC

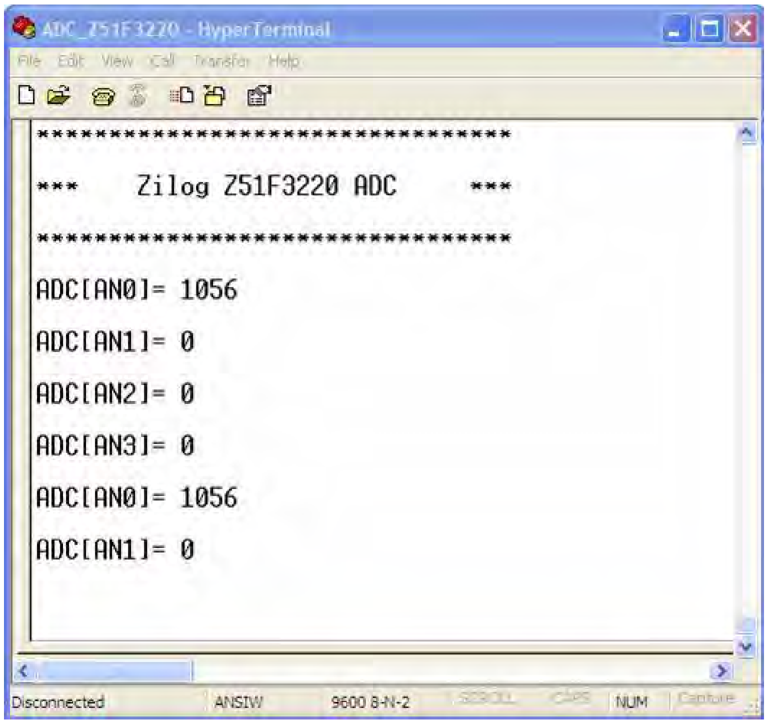
Note: In this path, <PC Directory> is the location of the unzipped AN0347-SC01 file.

7. Launch the Zilog Z8051 OCD Debugger software.
8. If a *Disconnected* message is displayed by the Debugger, turn the Z51F3220 Development Board OFF, then ON.
9. From the Debugger's **File** menu, select **Load Hex**. The Object File dialog will appear.
10. Click the **Browse...** button, then locate and open the `ADC_Z51F3220.hex` file from within the following filepath:

    <PC Directory>\AN0347-SC01\ADC_8051_SDCC

11. In the Object File dialog, click the **Download** button. The Configuration dialog box will appear.
12. Click the **Write** button to download the code to the MCU.
13. In the application software, the UART is set to communicate via the following parameters:
    - 9600 baud
    - 8 bits data frame
    - No parity bits
    - 2 stop bits

    Configure HyperTerminal (or any equivalent terminal emulation program) to these same values.
14. Disconnect the Z8051 OCD cable, then turn the Z51F3220 Development Board OFF, then ON, to run the code. Figure 8 shows an example of the initial HyperTerminal console display.


ADC_Z51F3220 - HyperTerminal
File Edit View Call Transfer Help
Disconnected ANSIW 9600 8-N-2 SCROLL CAPS NUM Capture

```
**********************************
***    Zilog Z51F3220 ADC     ***
**********************************
ADC[AN0]= 1056
ADC[AN1]= 0
ADC[AN2]= 0
ADC[AN3]= 0
ADC[AN0]= 1056
ADC[AN1]= 0
```

Figure 8. An Example SDCC Compiler HyperTerminal Display

15. Rotate the shafts of the potentiometers and observe the changing of the ADC values displayed in the HyperTerminal console.

Note: Inserting breakpoints using Zilog's Z8051 OCD software is not possible if the code is compiled with the SDCC compiler.

Experiment with the ADC APIs

Observe the following procedures with either compiler to experiment with the ADC interrupt, the ADC Align LSB data format, and/or the ADC Timer1 A match APIs.

Keil Compiler Procedure

1. Navigate to the following filepath and locate the `adc_api.h` file:

   <PC Directory>\AN0347-SC01\ADC_8051_Keil\include

2. Open the `adc_api.h` file using Notepad or similar text editor. Uncomment one or more of the following lines of code:

```
//#define USE_ADC_INTERRUPT
//#define USE_ADC_ALIGN_LSB
```

```
//#define USE_ADC_TIMER1_MATCH
```

3. Save the file.
4. Return to the the Test The Code Using The Keil Compiler section on page 10 and repeat the entire procedure.

SDCC Compiler Procedure

1. Navigate to the following filepath and locate the `adc_api.h` file:

   <PC Directory>\AN0347-SC01\ADC_8051_SDCC\include

2. Open the `adc_api.h` file using Notepad or similar text editor. Uncomment one or more of the following lines of code:

```
//#define USE_ADC_INTERRUPT
//#define USE_ADC_ALIGN_LSB
//#define USE_ADC_TIMER1_MATCH
```

3. Save the file.
4. Return to the the Test The Code Using The SDCC Compiler section on page 12 and repeat the entire procedure.

Results

Table 4 presents the computed ADC values and measured ADC values with respect to analog input voltage.

Table 4. Resulting ADC Values

V_{IN} (V)	Computed ADC Values	Measured ADC Values
0	0	0
0.2	171	167
0.4	341	342
0.6	512	512
0.8	683	679
1	853	848
1.2	1024	1016
1.4	1195	1187
1.6	1366	1360
1.8	1536	1521
2	1707	1703
2.2	1878	1870

Table 4. Resulting ADC Values (Continued)

V_{IN} (V)	Computed ADC Values	Measured ADC Values
2.4	2048	2048
2.6	2219	2223
2.8	2390	2385
3	2560	2567
3.2	2731	2719
3.4	2902	2903
3.6	3073	3070
3.8	3243	3248
4	3414	3423
4.2	3585	3591
4.4	3755	3759
4.6	3926	3931
4.798	4095	4095

The values listed in Table 4 can be determined with the following formula.

$$ADC = \left(2^{12} - 1\right) * \left(\frac{VIN}{VREF}\right)$$

In the above equation, V_{IN} represents the analog input voltage and $V_{REF} = 4.798\,V$.

For the computed ADC values, the analog input voltage values are generated. Conversely, for the measured ADC values, the analog input voltage values are measured on the Z51F3220 MCU's analog input pin. The measured analog input voltage is controlled by the potentiometer shown in Figure 6 on page 10.

Figure 9 shows a graphical representation of the ADC values in Table 4.

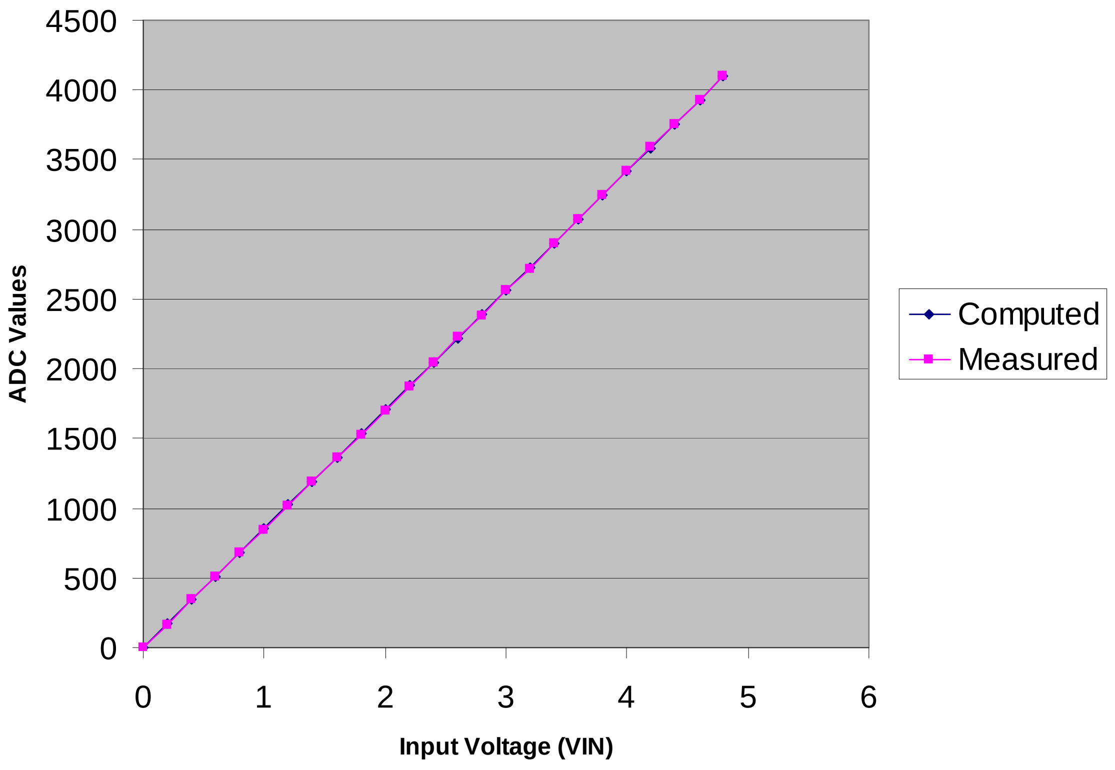


Figure 9. ADC Values vs. Input Voltage

Summary

This application provides sample code for properly setting up ADC functionality through the use of ADC APIs. There are five items to consider when setting up the ADC in this manner:

- ADC operational mode: Interrupt or Polling
- LSB or MSB data alignment format
- Select a Trigger signal
- Select an ADC clock
- Select a reference voltage source

Among the ADC APIs, `ADC_Start` enables and starts an analog-to-digital conversion, and `ADC_GetData` reads the current converted ADC data.

References

The following supporting documents are available free for download from the Zilog website.

- Z51F3220 Product Specification (PS0299)
- Z51F3220 Development Kit User Manual (UM0243)
- Z8051 Tools Product User Guide (PUG0033)

Appendix A. ADC APIs

This appendix documents the parameters and return values for the `ADC_Init`, `ADC_Start` and `ADC_GetData` APIs.

ADC_Init

Configures the ADC.

Parameters

ui8ADC_TRIG. Selects the ADC trigger signals.

ui8ADC_CKSEL. Selects the ADC clock.

ui8ADC_REFSEL. Selects the ADC reference voltage source.

Return Value(s)

None.

ADC_Start

Enables ADC module and starts A/D conversion of the specified analog channel, and is the first command before `ADC_GetData`.

Parameters

ui8ADC_Channel. ADC analog source.

Return Value(s)

None.

ADC_GetData

Reads ADC data.

Parameters

None.

Return Value(s)

12-bit ADC data or `0xFFFF` (A/D conversion is ongoing).

Appendix B. ADC Registers

Tables 5 and 6 show the ADC High and Low Control registers.

Table 5. ADC Control High Register (ADCCRH)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	ADCIFR	Reserved	TRIG2	TRIG1	TRIG0	ALIGN	CKSEL1	CKSEL0
Reset	0	–	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Address	9Dh							

Bit	**Description**
[7] ADCIFR	**ADC Interrupt** 0: ADC interrupt, no generation. 1: ADC interrupt generation.
[6]	**Reserved**
[5:3] TRIG[2:0]	**A/D Trigger Signal Selection** 000: ADST. 001: Timer 1 A match signal. 010: Timer 4 overflow event signal. 011: Timer 4 A match event signal. 100: Timer 4 B match event signal. 101: Timer 4 C match event signal.
[2] ALIGN	**A/D Converter Data Align Selection** 0: MSB Align. 1: LSB Align.
[1] CKSEL[1:0]	**A/D Converter Clock Selection** 00: $f_X \div 1$. 01: $f_X \div 2$. 10: $f_X \div 4$. 11: $f_X \div 8$.

Table 6. ADC Control Low Register (ADCCRL)

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Field	STBY	ADST	REFSEL	AFLAG	ADSEL3	ADSEL2	ADSEL1	ADSEL0
Reset	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Address	9Ch							

Bit	**Description**
[7] STBY	**Control Operation of the ADC** 0: ADC module disable. 1: ADC module enable.
[6] ADST	**Control A/D Conversion Stop/Start** 0: No effect. 1: Start A/D conversion and auto clear.
[5] REFSEL	**A/D Converter Reference Selection** 0: Internal reference (V_{DD}). 1: External reference (AV_{DD}).
[4] AFLAG	**A/D Converter Data Operation State** 0: A/D conversion in progress. 1: A/D conversion complete.
[3:0] ADSEL[3:0]	**A/D Converter Input Selection** 0000: AN0. 0001: AN1. 0010: AN2. 0011: AN3. 0100: AN4. 0101: AN5. 0110: AN6. 0111: AN7. 1000: AN8. 1001: AN9. 1010: AN10. 1011: AN11. 1100: AN12. 1101: AN13. 1110: AN14. 1111: AN15.

Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.

Warning: DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.