C2000 *Teaching Materials*

GETTING STARTED WITH THE TMS320F24x PROCESSOR

Tutorial 6: Analog to Digital Conversion

New Instructions Introduced

```
SFL
SFR
```

Introduction

The TMS320F243 has a built-in 8-channel analog-to-digital converter (ADC). This allows up to 8 different analog signals to be measured and converted into digital values.

Note that the ADCs of some other devices in the TMS320F24x family work in a different way to the TMS320F243, therefore many of the instructions in this tutorial do not apply to the TMS320LF2407.

ADC Options

The ADC can measure analog inputs on 8 different Port pins. These are referred to as Channel 0 to Channel 7. Only one channel can be active at any time.

The TMS320F243 has physically only one analog-to-digital converter. However, a point of potential confusion is that in the databooks, there are references to two analog-to-digital converters, ADC1 and ADC2.

For compatibility of software, all devices in the TMS320F24x family have an ADC1 and ADC2. In the case of the TMS320F243, both ADC1 and ADC2 control the same hardware.

This means we can use either ADC1 or ADC2, which can be useful when measuring two inputs.

Conversion Modes

The analog-to-digital conversion can be done in two different ways. The first way is what is known as *continuous conversion,* whereby the input value is being repeatedly measured and converted to its digital equivalent.

The second way is *single conversion*, whereby only one conversion is done and the ADC waits to be prompted before it makes the next measurement. Typically, a timer would be used to start conversions at specific time intervals.

Setting up the ADC

In order to use the analog-to-digital converter, we must first configure certain dedicated registers. Amongst these are ADC Control Register 1 (ADCTRL1 at 7032h) and ADC Control Register 2 (ADCTRL2 at 7034h).

Note that the register ADCTRL1 is not dedicated solely to ADC1. It is used by both ADC1 and ADC2. The same applies to the register ADCTRL2

Configuring ADCTRL1

ADCTRL1 allows us to choose which one of the 8 input channels to use, ADC1 or ADC2, single or continuous conversion etc. Each of these options is controlled by specific bits in ADCTRL1. The power-on default for ADCTRL1 is for the ADC1 and ADC2 to be turned off.

Let us take an example. Suppose we wish to use Channel 0 of ADC1 with conversion occurring all of the time. The bits in ADCTRL1 in which we are interested are:

	Bit 11	1 = ADC Enabled. 0 = ADC Disabled.
	Bit 10	1 = Continuous conversion mode. 0 = Single conversion.
	Bits 3-1	Select channel. Channel 0 = 000 to Channel 7 = 111.
	Bit 0	1 = Start converting.

All other bits we shall leave as zero. The binary value to be written to ADCTRL1 is therefore:

Figure 6-1. Configuration of ADCTRL1.

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1

By *enabled* we mean that the ADC is internally connected, but conversion has not necessarily started. The *start converting* flag (bit 0) controls conversions.

We shall use 0C01h as the value written to ADCTRL1.

Configuring ADCTRL2

The bits of ADCTRL2 in which we are most interested are bits 3, 2 and 1. These three bits control the *pre-scaler*. This value is used to reconcile the timings of the chip with the timings of the ADC. We shall set this value to maximum, which gives ADC plenty of time to settle between readings and therefore best accuracy.

Figure 6-2. Configuration of ADCTRL2.

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

This means we can use a value of 0007h to configure ADCTRL2

Code to Configure the ADC

To set up and use the analog-to-digital converter, we must first write appropriate values to the registers ADCTRL1 and ADCTRL2. We shall use the values from Figure 6-1 and Figure 6-2.

Example 6-1.

	`.setsect ".text",`	`8800h`

`start``:`	`LDP #224`	`; Page 224. Gain access to data`
		`; memory addresses 7000h to`
		`; 707Fh.`

	`SPLK #0C01h, 32h`	`; Write 0C01h to register`
		`; ADCTRL1 at 7000h + 32h = 7032h.`

	`SPLK #0007h, 34h`	`; Write 0007h to register ADCTRL2`
		`; at 7000h + 34h = 7034h.`

	`B `*`start`*	`; Go round again.`

A stylistically better way to write the code in Example 6-1 is to use the `.set` assembler directive. This allows us to work using meaningful symbols rather than just numbers, which could mean many different things.

Example 6-2.

	.setsect ".text",	8800h
DP_ADC	.set 224	; Page for ADC access.
ADCTRL1	.set 32h	; Offset for ADCTRL1
ADCTRL2	.set 34h	; Offset for ADCTRL2
start:	LDP #DP_ADC	; Page 224 for ADC. Gain ; access to data memory ; addresses 7000h to 707Fh.
	SPLK #0C01h, ADCTRL1	; Write 0C01h to register ; ADCTRL1 at address ; 7000h + 32h = 7032h.
	SPLK #0007h, ADCTRL2	; Write 0007h to register ; ADCTRL2 at address ; 7000h + 34h = 7034h.
	B *start*	; Go round again.

We have used the convention that `DP_` stands for a data pointer. For example, `DP_ADC` indicates the Data Page for the ADC. Similarly, `DP_GP` would be the data page for general-purpose memory. The name `DP_ADC` is a *label*, and must therefore go in the left-most column.

Reading the Measured Value

The measured result of carrying out an analog-to-digital conversion is stored in either ADCFIFO1 at 7036h or ADCFIF02 at 7038h. For the purpose of this tutorial we shall treat them as simple registers.

Figure 6-3. ADC Value in ADCFIFO1 or ADCFIF02

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0	0	0

The way in which the result is stored may at first appear a little strange.

The 10 bits of the analog-to-digital conversion are stored in bits 15 to 6 of ADCFIFO1 and ADCFIFO2. They are indicated in Figure 6-3 as D9 to D0. The remaining bits 5 to 0 are filled with zeroes.

It might seem more obvious to store the 10-bit result in bits 9 to 0; that is to right-justify it. So why has the result been stored in this way? The reasons are two-fold.

First it is standard practice in mathematics to work to more decimal places than required and then to round the result after the calculations have been done. Putting zeroes at the end of the value increases the resolution of any subsequent operations and reduces rounding errors. This is especially important when carrying out divisions or shifts to the right where digits are lost.

Second, the result of the analog-to-digital conversion lies in the range 0 to 1023. This may represent an input of 0-5V, 0-1A, 0-100%, 4-20mA etc. depending upon the system. In order to *scale* the input, that is to convert 0 to 1023 to the units we wish to measure, a simple way is to multiply the result of the analog-to-digital conversion by a *scaling factor*. We shall see in a later tutorial how left-justifying the result of the analog-to-digital conversion simplifies this multiplication.

Example 6-3.

	.setsect ".text",	8800h
DP_ADC	.set 224	; Data page for ADC
ADCTRL1	.set 32h	; ADC Control Register 1.
ADCTRL2	.set 34h	; ADC Control Register 2.
ADCFIFO1	.set 36h	; Result of ADC conversion.
start:	LDP #DP_ADC	; Page 224. Gain access to ; data memory addresses ; 7000h to 707Fh.
	SPLK #0C01h , ADCTRL1	; Setup as per Figure 6-1.
	SPLK #0007h , ADCTRL2	; Setup as per Figure 6-2.
	LACC ADCFIFO1	; Copy left-justified result ; from ADCFIFO1 to the ; accumulator. The maximum ; value the accumulator ; contains is 0000B300h.
	B *start*	; Branch to the label start.

`DP_ADC` uses the symbol #. This is because the assembler takes any number greater than 127 to be an out-of-range direct address and will truncate it to 7 bits. The symbol # indicates that the value 224 is a number, not a direct address.

Right Justifying the Result of an ADC Conversion

Suppose we wish the value stored in ADCFIFO1 to represent 0 to 1023. We would need to shift the value six places to the right i.e. lose the right-most six zeroes.

For each shift, we use the instruction `SFR` (shift accumulator right). This instruction does not require an operand. Each shift to the right is equivalent to dividing by two. Therefore, six shifts to the right are equivalent to dividing by 64.

Let us assume that the value measured by the analog-to-digital conversion is B300, which is stored in ADCFIFO1.

Example 6-4.

	.setsect ".text",	8800h
DP_ADC	.set 224	; Data page for ADC.
ADCTRL1	.set 32h	; ADC Control Register 1.
ADCTRL2	.set 34h	; ADC Control Register 2.
ADCFIFO1	.set 36h	; Result of ADC conversion.
start:	LDP #DP_ADC	; Page 224. Gain access to ; data memory addresses ; 7000h to 707Fh.
	SPLK #0C01h, ADCTRL1	; Setup as per Figure 6-1.
	SPLK #0007h, ADCTRL2	; Setup as per Figure 6-2.
	LACC ADCFIFO1	; Copy left-justified result ; from ADCFIFO1 to the ; accumulator. The ; accumulator contains ; 0000B300h.
	SFR	; The accumulator contains ; 00005980h.
	SFR	; The accumulator contains ; 00002CC0h.
	SFR	; The accumulator contains ; 00001660h.
	SFR	; The accumulator contains ; 00000B30h.
	SFR	; The accumulator contains ; 00000598h.
	SFR	; The accumulator contains ; 000002CCh.
	B *start*	; Branch to the label start.

Example 6-4 shows how the instruction `SFR` (shift right accumulator) is used. It is used a total of six times to right-justify the ADC reading.

Using an Analog Input to Control the Brightness of the XF LED

We shall now use an input voltage at channel 0 of the ADC to control the brightness of the XF LED. To show how the shifts work, we shall again assume that the value in ADFIFO1 is B300h.

Example 6-5.

	.setsect ".text",	8800h
DP_ADC	.set 224	; Data page for ADC.
DP_GP	.set 6	; Data page general purpose.
ADCTRL1	.set 32h	; ADC Control Register 1.
ADCTLR2	.set 34h	; ADC Control Register 2.
ADCFIF0	.set 36h	; ADC Results Register.
temp	.set 0h	; Temporary storage.
start:	LDP #DP_ADC	; Page 224. Gain access to ; data memory addresses ; 7000h to 707Fh.
	SPLK #0C01h, ADCTRL1	; Write 0C01h to register ; ADCTRL1 at 7032h.
	SPLK #0007h, ADCTRL2	; Write 0007h to register ; ADCTRL2 at 7034h.
loop:	LACC ADCFIFO1	; Copy left-justified result ; from ADCFIFO1 to ; accumulator. The ; accumulator contains ; 0000B300h.
	SFR	; The accumulator now ; contains 00005980h.
	SFR	; The accumulator now ; contains 00002CC0h.
	SFR	; The accumulator now ; contains 00001660h.
	SFR	; The accumulator now ; contains 00000B30h.
	SFR	; The accumulator now ; contains 00000598h.
	SFR	; The accumulator now ; contains 000002CCh.
	LDP #DP_GP	; General purpose memory.
	SACL temp	; Make copy of the ADC ; reading.
	SETC XF	; Turn on XF LED.
loop1:	B *done1*, EQ	; Terminate when zero.
	SUB #1	; Decrement the value in the ; accumulator.
	B *loop1*	; Loop round again.
done1:	CLRC XF	; Turn off XF LED.

	LACC #1023	; Get copied value.
	SUB temp	; 1023 - temp.
loop2:	B *done2*, EQ	; Terminate when zero.
	SUB #1	; Decrement accumulator.
	B *loop2*	; Continue looping.
done2:	B *loop*	; Go round loop again.

The code at the label `loop` sets up the ADC. Then we read in the value at ADC channel 0 and shift it 5 places to the right, before temporarily saving it in `temp`.

We then carry out the code at `loop1` to turn the XF LED on for a time proportional to `temp`. The number of loops will be determined by the value of *temp* and will lie in the range 0 and 1023.

The code at `loop2` turns off the XF LED. The number of loops is determined by 1023 - `temp` times. This means that the total number of loops executed at loop1 and loop2 will be:

`temp` + 1023 – `temp` = 1023.

Note that we have saved the value of ADCFIFO1 at the variable `temp` before using it to control the two loops. This is because with continuous conversion, the ADC value could change and therefore a different value of ADCFIFO1 could be used for `loop1` and `loop2`.

Selecting the ADC Channel

In order to select the ADC channel to be used, we must put the number of the channel in the register ADCTRL1. This determines which of the eight physical input pins is measured.

The channel number for ADC1 is offset one place from the right. For ADC2 it is offset four places from the. This is shown in Figure 6-4. For clarity, other fields are shown as blanks.

Figure 6-4. Bits of Register ADCTRL1 used to control ADC1 and ADC2

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
									ADC2			ADC1			

In order to configure ADC1, we need to put the channel number (in range 0 to 7) into bits 3, 2 and 1.

Similarly, in order to configure ADC2, we need to put the channel number (in range 0 to 7) into bits 6, 5 and 4.

Logical OR with Shift

Before we can setup ADCTRL1, we need to use a logical OR with shift. We have already used the instruction `OR` (logical OR) to set particular bits of the accumulator. The instruction `OR` can also take a second operand to specify a *shift* to the left.

For example:

1111h shifted one place to the left is 2222h.
1111h shifted two places to the left is 4444h.
1111h shifted three places to the left is 8888h.

In code, a shift of three places to the left becomes:

Example 6-6.

	.setsect ".text",	8800h
start:	LACC #5432h	; Accumulator contains ; 0101 0100 0011 0010 binary.
	OR #1111h, 3	; Perform logical OR of ; accumulator with ; 1000 1000 1000 1000 binary. The ; accumulator now contains ; 1101 1100 1011 1010 binary ; which is DCBAh.
	B *start*	; Loop round again.

In Example 6-6, the value 1111h is shifted three places to the left *before* the logical OR is performed. In this case it is easier to see the effect of the shift using binary than it is using hexadecimal.

The purpose of this shift allows us to put a number into a data word, but offset from the right.

Changing the Selected ADC Channel

So far, we have only worked using channel 0 of the ADC converter. Say we have a system whereby we wish to change from channel 0 to channel 4. To do so, we need to alter bits 3, 2 and 1 of the ADC control register (ADCTRL1) *but* leave all the other bits unchanged.

In C code we might write:

Example 6-7.

	unsigned int ADCTRL1;	
	ADCTRL = (ADCTRL1 & 0xFFF1) | (0x0004 << 1);	

What we have done in Example 6-7 is to clear bits 3, 2 and 1 of ADCTRL1 to zero using the `&` operator. All the other bits of ADCTRL1 remain unchanged. We then shift our channel number (here channel 4) one place to the right then perform a logical or with ADCTRL1.

The assembly language equivalent is:

Example 6-8.

	.setsect ".text",	8800h
DP_ADC	.set 224	; Page 224. Gain access to ; data memory addresses 7000h ; to 707Fh.
ADCTRL1	.set 32h	; ADC Control Register 1.
ADCTRL2	.set 34h	; ADC Control Register 2.
start:	LDP #DP_ADC	; Data Page for analog-to- ; digital converter.
	SPLK #0C01h, ADCTRL1	; Write 0C01h to register ; ADCTRL1 at 7032h.
	SPLK #0007h, ADCTRL2	; Write 0007h to register ; ADCTRL2 at 7034h.
loop:	LACC ADCTRL1	; Load contents of ADCTRL1 ; into the accumulator.
	AND #0FFF1h	; 1111 1111 1111 0001 binary. ; Clear bits 3, 2 and 1.
	OR #4, 1	; Shift the channel number ; one place to the left. ; 0000 0000 0000 1000 binary. ; Logical OR with ; accumulator.
	SACL ADCTRL1	; Store new value in ADCTRL1.
	B *loop*	; Branch to the label start.

The channel number in the range 0 to 7 is put into bits 3, 2 and 1 of ADCTRL1.

There is however, a complication. The instruction `SPLK #0C01h, ADCTRL1` writes the value 0C01h to ADCTRL1. When the instruction `LACC ADCTRL1` is executed, the value read back is 0D80h.Why has the value changed?

Two of the bits in ADCTRL1 are status flags. Their values depend on the current status of the ADC, and may therefore be a value other than 0.

Another way to write code to set the channel number is shown in Example 6-9.

Example 6-9.

	`unsigned int ADCTRL1;`	
	`ADCTRL = (0x0C07 | (0x0004 << 1));`	

The assembly language equivalent is:

Example 6-10.

loop:	`LACC #0C0lh`	`; Load the accumulator with`
		`; the base value to be written`
		`; to ADCTRL1.`

	`OR #4, 1`	`; Shift the channel number one`
		`; place to the left.`
		`; 0000 0000 0000 1000 binary.`
		`; Logical OR with accumulator.`

	`SACL ADCTRL1`	`; Store new value in ADCTRL1.`

In this case, a logical OR is performed with the constant #0C0lh.

Using ADC2

Should we be using ADC2, the bits of ADCTRL1 used to select the channel are bits 6, 5 and 4. To select channel 5 using ADC2 we would write:

Example 6-10.

	`unsigned int ADCTRL1;`	
	`ADCTRL = ((ADCTRL1 & 0xFF8F)|(0x0005 << 4));`	

This time we start by clearing bits 6, 5 and 4 to zero using the operator &. This clears the bits used to configure ADC2, but leaves all the other bits unaffected. We then take the channel number, here 5, and shift it 4 places to the left. Finally we perform a logical OR to put the shifted channel number into ADCTRL1.

The assembly language equivalent is:

Example 6-11.

	.setsect ".text",	8800h
DP_ADC	.set 224	; Data page for ADC.
ADCTRL1	.set 32h	; ADC Control Register 1.
start:	LDP #DP_ADC	; Page 224. Gain access to data ; memory addresses 7000h to ; 707Fh.
	LACC ADCTRL1	; Load contents of ADCTRL1 into ; the accumulator.
	AND #0FF8Fh	; 1111 1111 1000 1111 binary. ; Clear bits 7, 6 and 5.
	OR #5, 4	; OR in new channel number ; shifted four places left. ; 0000 0000 0101 0000 binary.
	SACL ADCTRL1	; Store new value in ADCTRL1.
	B *start*	; Branch to the label start.

When using ADC2, the number of the channel must first be shifted four places to the right (multiplied by 16).

Again, when a value is written to ADCTRL1, a different value may be read back.

TMS320F243 DSK EXPERIMENTS

Equipment Required

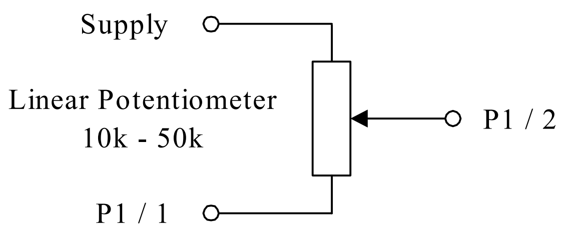
TMS320F243 DSK
Linear Potentiometer connected as per Figure 6-5.
Frequency meter or oscilloscope
Analog voltmeter with range 0 to 5V D.C.

Experiment 6-1.

Objective: To determine the extremes of ADC measurement

Connect the potentiometer to connector P1 of the TMS320F243 DSK as shown in Figure 6-5. One end is connected to ground (P1/1) and the other is connected to 5V P2/1). Should a 3.3V device in the TMS320F24x family be used such as the TMS320LF2407, then the maximum allowed voltage at the potentiometer should be 3.3V, not 5V.

Figure 6-5. Connection of Potentiometer to TMS320F243 DSK



For the TMS320F243 DSK, the supply can be between 3.3V and 5V. However, if a 3.3V device is being used such as the TMS320LFf2407, then the maximum allowed supply is 3.3V.

Enter the code in Example 6-5 into a text editor, save it, assemble it and run it on the debugger. Make a note of the values in the accumulator when the potentiometer is at each extreme (2 readings) and in the center. The values should lie between a minimum of 0 and a maximum 3B0h.

Experiment 6-2.

Objective: To use the ADC to control the D.C. output on XF

Again using the code in Example 6-5. Set the potentiometer to its left-most extreme. Measure the voltage between 0V (Connector P1, Pin1) and XF (Connector P1, Pin 21) using the analog voltmeter.

Repeat using the potentiometer set to near the center of its travel and at the other extreme. Sweep the potentiometer from one end to the other and observe how the voltage changes.

Experiment 6-3.

Objective: To measure the output frequency produced at XF

Set the potentiometer to its left-most extreme. Connect the frequency meter probes between connector P1, Pin1) and XF (Connector P1, Pin 21). Slowly sweep the potentiometer from one end of its range to the other, continually monitoring the frequency meter. The frequency should not change. Make a note of this frequency.

Example 6-4.

Objective: To make the input potentiometer work in the reverse direction

Modify the code in Example 6-5 so that the potentiometer operates in the reverse direction.

Design Problem

Write a program to write to measure each channel 0 to 7 in turn and store the values at successive data memory addresses 400h to 407h. Use ADC2.

Self-Test Questions.. *Click Here To View* Answers!

1.	How many physical ADCs are there on TMS320F243?
2.	When the TMS320F243 powers up, is the ADC on or off?
3.	ADCTRL1 is dedicated to ADC1 and ADCTRL2 is dedicated to ADC2. Is this true or false?
4.	What is the difference in function between the *enable* and *start conversion* bits in ADCTRL1?
5.	What is meant by the term *channel* when applied to an analog-to-digital converter?
6.	Why is the result of an analog to digital conversion, as stored in ADCFIFO1 or ADCFIFO2 left justified?

7.	Which syntax of the following is correct?
	`a) SFR`
	`b) SFR 1`
	`c) SFR -1`

8.	What single instruction can be used to perform a multiply by 2?
9.	The instruction `OR #3, 4` performs which of the following? a) Logical OR of accumulator with 0003h then left shift of accumulator by 4 b) Logical OR of accumulator with 0003h then right shift by 4. c) Logical OR of accumulator with 0003h shifted left four places. c) Logical OR of accumulator with 0003h then stored at 4h.
10.	What is the maximum value that ADCFIFO1 or ADCFIFO2 can contain?
11.	What is the minimum value that ADCFIFO1 or ADCFIFO2 can contain?
12.	When we use the directive .set with a value greater than 7Fh, why must we use the symbol #?
13.	If we write the value #0C0lh to `ADCTRL1`, why might it read back as 0D80h?

References

TMS320F/C24x DSP Controllers. CPU and Instruction Set. Reference number SPRU160

TMS320F/C240 DSP Controllers. Peripheral Library and Specific Devices. Reference Number SPRU161.

TMS320F243, TMS320F241 DSP Controllers. SPRS064

CLICK TO VIEW
Tutorials
1 2 3 4 5 6 7 8 9 10

Click here to view.......... Route Map