APPLICATION NOTE

MIKE PAPPAS
ECE 492 WINTER 2015
GROUP 2

Texas Instruments MSP430-FR5969 12-Bit ADC Setup Guide

March 2015

Table of Contents

Preface

The Texas Instruments MSP-EXP430FR5969 development board features a 12-bit resolution Analog to Digital Converter (ADC) that is capable of performing both single and multiple channel conversions on up to 16 external channel inputs. This guide is intended to outline the steps needed to get the ADC working on both single and multi-channel conversions. For more MSP430 12-bit ADC information, including information on the register information and autoscan functionality, consult Chapter 25 of the Texas Instruments MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide. In addition, various ADC examples demonstrating different capabilities of the MSP430FR5969 hardware can be found here. This application note assumes the user has Code Composer Studio (an Eclipse-based development studio) installed, running on the Windows or Linux environment. Instructions for installing Code Composer Studio on Linux can be found on the Texas Instruments Wiki page here.

Pin Assignments

In order to access the ADC using the MSP-EXP430FR5969 development board, the user must first select an analog channel from the available pins. Unfortunately, only 6 of the available 16 analog channel are accessible from the development board. The following table summarizes the available analog channels:

Channel	Pin Mapping	BoosterPack Pin Number
2	P1.2	19
3	P1.3	11
4	P1.4	12
5	P1.5	13
11	P4.3	5
12	P3.0	18

Table 1 - MSP-EXP430FR5969 ADC Pin Mappings [1]

Once the desired channels have been selected, they can be mapped in the program as follows:

```
// Temperature Input Setup
P1SEL1 |= BIT3;                          // Configure P1.3 for ADC
P1SEL0 |= BIT3;

// Light Sensor Input Setup
P1SEL1 |= BIT4;                          // Configure P1.4 for ADC
P1SEL0 |= BIT4;
```

Where the left hand side of the pin assignment refers to the decimal portion of the pin mapping, and the right hand side refers to the fractional value. In this case, pins P1.3 and P1.4 are selected. Both PxSELx assignments are required to enable GPIO on that pin. All macro values can be found in the msp430.h header file.

Configuring the ADC

In order to configure the ADC12 to operate in a desired manner, three main registers must be addressed: three ADC control registers, the desired interrupt registers, and the desired memory control registers. The code below shows an example of a single channel configuration:

```
// Configure ADC12
ADC12CTL0 = ADC12SHT0_2 | ADC12ON;        // Cycle Sample Time, ADC On
ADC12CTL1 = ADC12SHP;                     // Source clock is sample timer
ADC12CTL2 |= ADC12RES_2;                  // 12-bit conversion
ADC12IER0 |= ADC12IE0;                    // Interrupt MEM0
ADC12MCTL0 |= ADC12INCH_4 | ADC12VRSEL_3; // Select A4, Vref = 2.5V
```

The above code should be sufficient for any single channel sampling, however the final ADC12MCTL0 (which refers to memory channel 0) line may need to be modified depending on the selected input channel and desired reference voltage. The ADC is capable of operating at internal reference voltages of 1.2V, 2.0V and 2.5V but external options are available should those prove insufficient [2]. As with the pin assignments, all macro values can be found in the msp40.h header file.

For multi-channel sampling, some minor modifications are required. The following example shows the configuration:

```
// Configure ADC12

// Turn on ADC and enable multiple conversions
ADC12CTL0 = ADC12SHT0_2 | ADC12ON | ADC12MSC;
// Sampling timer, single sequence
ADC12CTL1 |= ADC12SHP | ADC12CONSEQ_1;
// 12-bit conversion
ADC12CTL2 |= ADC12RES_2;
// Enable ADC  interrupt on MEM1
ADC12IER0 |= ADC12IE1;
// A3 select, Vref=1.2V
ADC12MCTL0 |= ADC12INCH_3 | ADC12VRSEL_1;
// A4 select, Vref=1.2V, End of Sequence
ADC12MCTL1 |= ADC12INCH_4 | ADC12VRSEL_1 | ADC12EOS;
```

The first difference between the single and multi-channel examples is the ADC12MSC option, which notifies the ADC that more than one channel will be in use. The following ADC12CONSEQ_1 option sets the ADC to single sequence mode – meaning the ADC will only perform one pass through the channels before stopping. It is interesting to note that the interrupt is set to trigger on only the flag of memory register 1 (even though both 0 and 1 are in use) – this is because the MSP430 interrupt only needs to trigger once the final conversion is done, so an interrupt on the first register would be unnecessary. A good rule of thumb is to enable the interrupt of the last memory channel that is to be sampled from. Finally, the ADC12EOS on the final input channel informs the ADC to stop sampling after reaching that channel. More information about the register values can be found in the user guide [2].

Sampling from the ADC

Once the ADC has been properly configured the analog sampling can take place. To do so, the ADC must first be told to enable and start conversions. The following code example demonstrates this process:

```
ADC12CTL0 |= ADC12ENC | ADC12SC;          // Sampling and conversion start
__bis_SR_register(LPM0_bits + GIE);      // LPM0, ADC12_ISR will force exit
__no_operation();                         // For debug only
```

The__bis_SR_register() command enables the low power mode in the status register after returning from the ADC interrupt routine, and enables the general interrupt to wake up the MSP430 upon another ADC sampling.

ADC Interrupt Routine

In order to do work with the results from the ADC, a proper interrupt routine must be utilized. As mentioned previously, it is only necessary to trigger on the flag of the last memory register to be sampled from, however there is no harm from performing an action within each case. The following code demonstrates how to save the conversion results from two input channels onto global values, then exit the interrupt routine:

```
// ADC Vector function adapted from http://www.ti.com/lit/zip/slac536
// using MSP430FR59xx_adc12_01.c
// Written by: T. Witt / P. Thanigai, Texas Instruments Inc., November 2013
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector = ADC12_VECTOR
__interrupt void ADC12_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(ADC12_VECTOR))) ADC12_ISR (void)
#else
#error Compiler not supported!
#endif
{
  switch (__even_in_range(ADC12IV, ADC12IV_ADC12RDYIFG))
  {
    case ADC12IV_ADC12IFG1:                          // ADC12MEM1 Interrupt
        ADC_value = ADC12MEM0;                       // Save MEM0
        ADC_value2 = ADC12MEM1;                      // Save MEM1
      __bic_SR_register_on_exit(LPM0_bits | GIE);    // Exit CPU, clear interrupts
      break;
    default: break;
  }
}
```

For performing multiple channel conversions it is important that the __bic_SR_register_on_exit() command is only issued in the end of sequence channel interrupt defined in the configuration, otherwise the conversion process will be truncated.

Conversion Results

Once the conversion results have been saved and stored, they must be manually converted by the user back into meaningful results. To perform the conversion, the following formula can be used:

$$Measured\ Voltage = ADC\ Value * \frac{V_{ref}(mV)}{2^{ADC\ Resolution}}$$

For the case of this example, 12 bit ADC resolution was used making the denominator 4096. If more precise values are needed (down to fractions) a more detailed formula can be found in the user's guide [2].

References

[1] Texas Instruments Incorporated, "MSP-EXP430FR5969 LaunchPad™ Development Kit User's Guide," June 2014. [Online]. Available: http://www.ti.com/lit/ug/slau535a/slau535a.pdf. [Accessed 12 3 2015].

[2] Texas Instruments Incorporated, "MSP430FR58xx, MSP430FR59xx, MSP430FR68xx, and MSP430FR69xx Family User's Guide," January 2015. [Online]. Available: http://www.ti.com/lit/ug/slau367f/slau367f.pdf. [Accessed 12 3 2015].