

Analog Peripherals

Introduction

In this section we'll take a look at the MSP430 analog peripherals. It's not possible in this limited amount of time to give you a complete overview of the possible analog inputs, but hopefully this introduction will guide you in the right direction

Objectives

- Comparators
- ADC10 & 12
- SD16 & SD16A
- DAC12
- DTC
- Timer triggers
- Lab

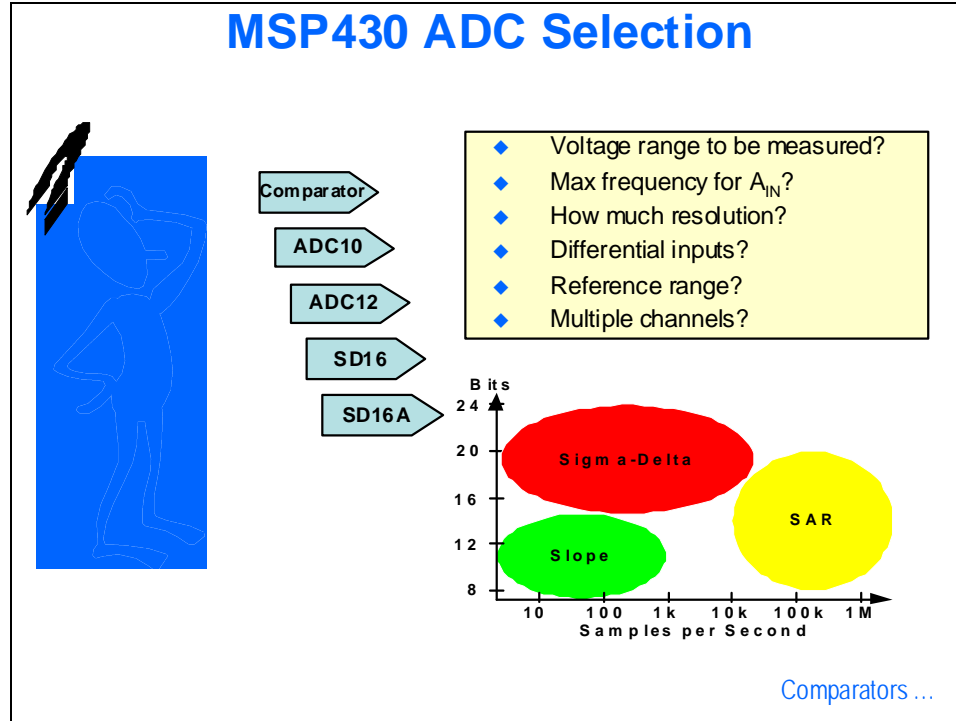
*** This page intentionally left _____ ***

Module Topics

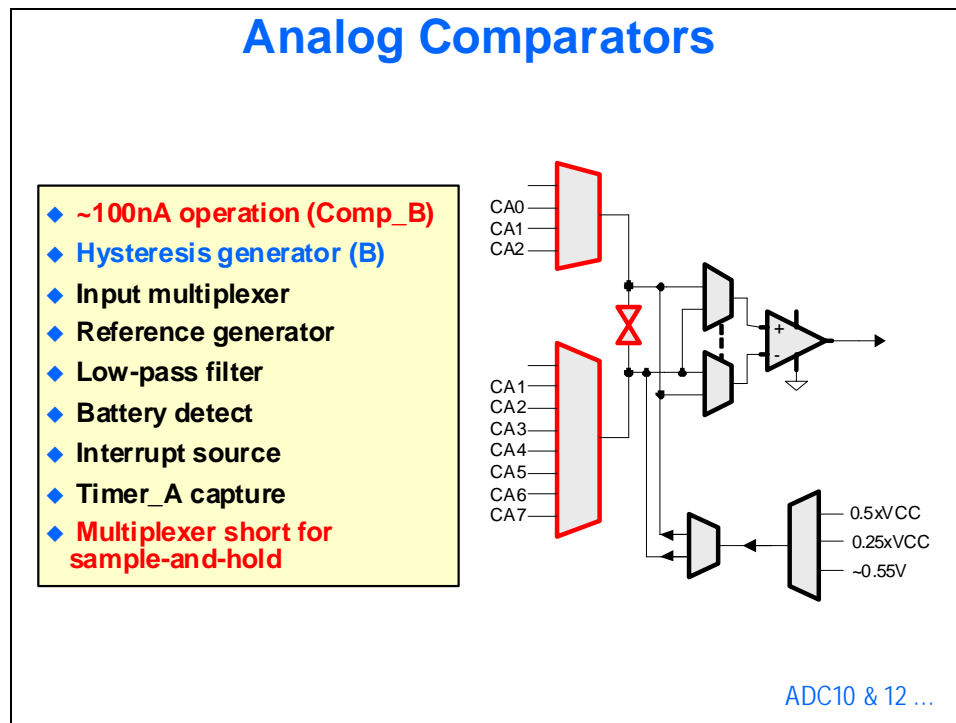
Analog Peripherals.....	3-1
<i>Module Topics.....</i>	3-3
ADC Selection.....	3-5
Comparators	3-5
ADC10 & ADC12	3-6
Conversion Memory and Control	3-6
ADC10 DTC.....	3-7
Timer Triggers.....	3-7
SD16.....	3-8
SD16A	3-8
SD16 & SD16A Input Range	3-9
DAC12.....	3-9
<i>Lab 4 – Using the ADC12.....</i>	3-11
Hardware list:	3-12
Software list:.....	3-12
<i>IAR Kickstart Procedure.....</i>	3-13
Start Up.....	3-13
Add Source File.....	3-13
Complete the Code	3-13
Test Your Work.....	3-16
Shut Down	3-16
<i>Code Composer Studio 4.1 Procedure.....</i>	3-17
Start Up.....	3-17
Add Source File.....	3-17
Complete the Code	3-17
Test Your Work.....	3-20
Shut Down	3-20
<i>Review Questions.....</i>	3-21

*** Blankety, blank ***

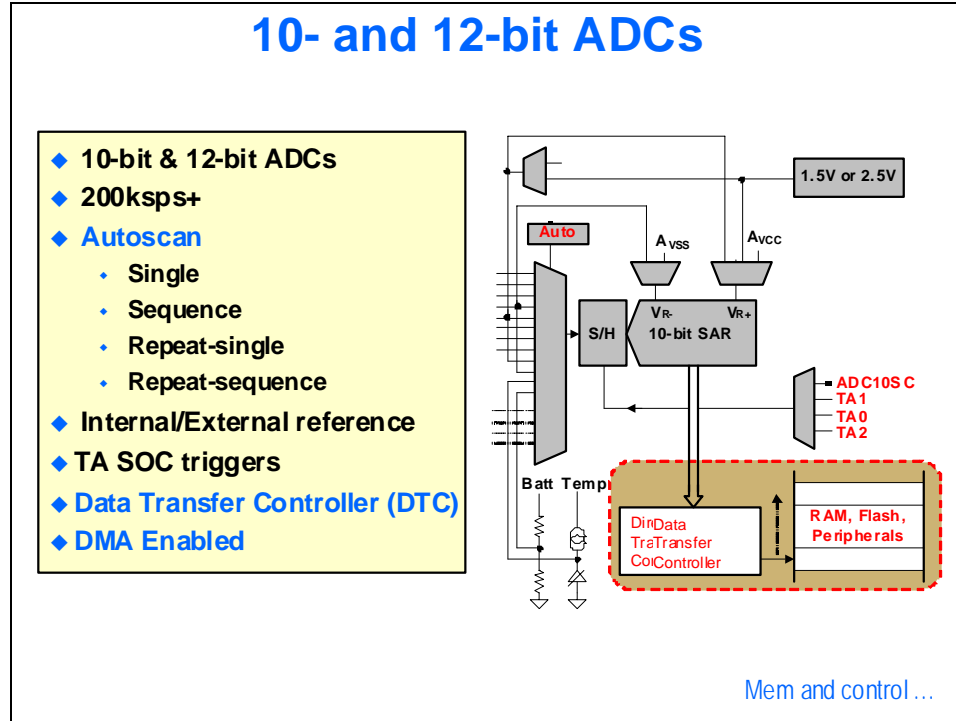
ADC Selection



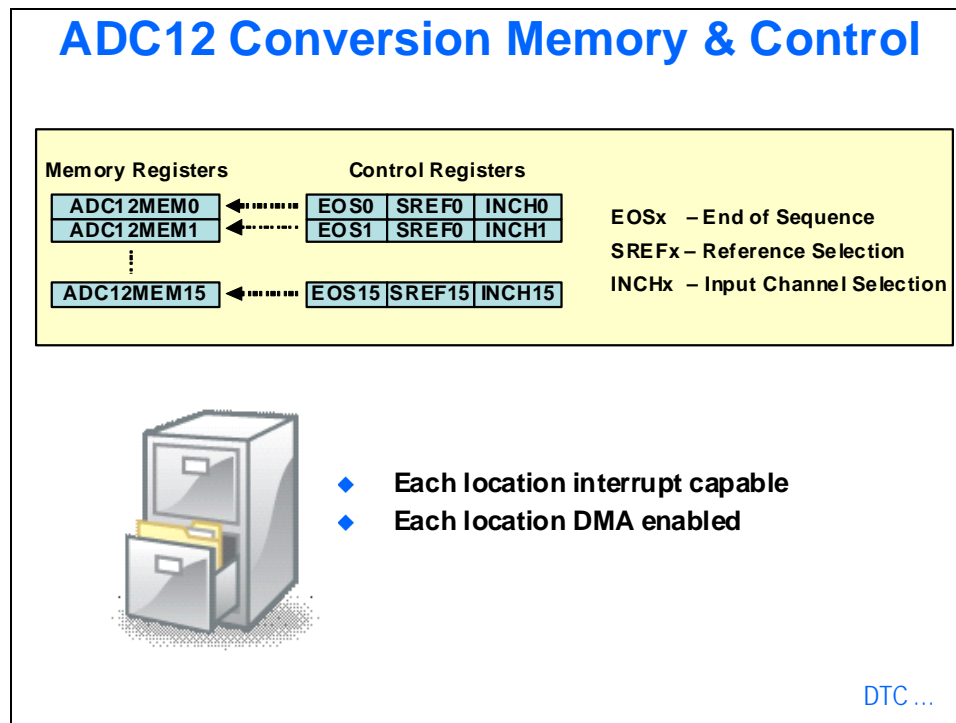
Comparators



ADC10 & ADC12



Conversion Memory and Control



ADC10 DTC

ADC10 Direct Transfer Controller (DTC)

```

// Software
Res[pRes++] = ADC10MEM;
ADC10CTL0 &= ~ENC;
if (pRes < NR_CONV)
{
  CurrINCH++;
  if (CurrINCH == 3)
    CurrINCH = 0;
  ADC10CTL1 &= ~INCH_3;
  ADC10CTL1 |= CurrINCH;
  ADC10CTL0 |= ENC+ADC10SC;
}
    
```

```

// DTC
_BIS_SR(CPUOFF);
    
```

70 cycles/Sample **Fully Automatic**

Other automated conversion methods offer similar benefits

[Timer Triggers ...](#)

Timer Triggers

Timer Triggers – Low-Power

```

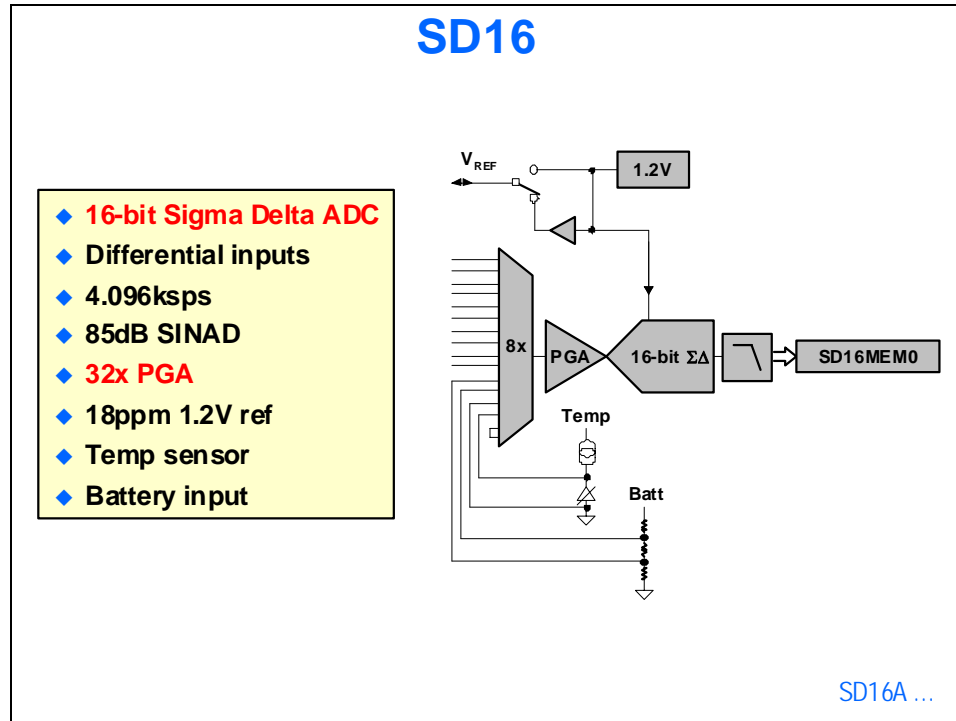
// Interrupt
; MSP430 ISR to start conversion
BIS #ADC12SC,&ADC12CTL0 ; Start conversion
RETI ; Return
;
    
```

	<u>CPU cycles</u>
; MSP430 ISR to start conversion	6
BIS #ADC12SC,&ADC12CTL0 ; Start conversion	5
RETI ; Return	<u>5</u>
;	16

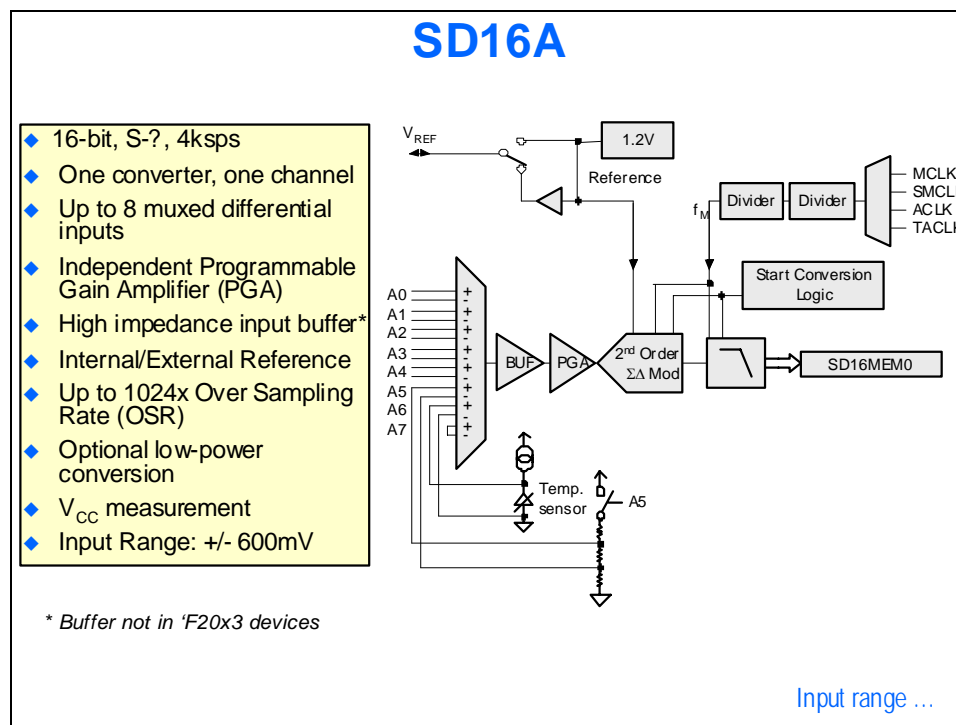
Timer triggered interrupts – no software wait loops

[SD16 ...](#)

SD16



SD16A



SD16 & SD16A Input Range

SD16/SD16A Input Range

- ◆ What is V_{REF} ?
- ◆ What is the PGA setting?

$$V_{FSR} = \frac{V_{ref} / 2}{GAIN_{PGA}}$$

- ◆ Applies to all inputs & modes
- ◆ $0V = V_{SS}$ (SD16), $0V = relative$ (SD16A)

GAIN 1 2 ... 32

+0.6V
+0.5V
0 V
-0.5V
-0.6V

+0.015V
-0.015V

DAC12 ...

DAC12

DAC12

- ◆ 12-bit monotonic
- ◆ 8/12-bit voltage output
- ◆ Programmable settling time versus power
- ◆ Internal/External reference
- ◆ Binary or 2's compliment
- ◆ Self-calibration
- ◆ Group sync load
- ◆ DMA enabled

ADC12 1.5V / 2.5V Reference

A_{VSS}

GND

TA1
TB2

Group Load

GND

TA1
TB2

V_{R+} V_{R-}

V_{R+} V_{R-}

DAC12_0

DAC12_1

Lab4 ...

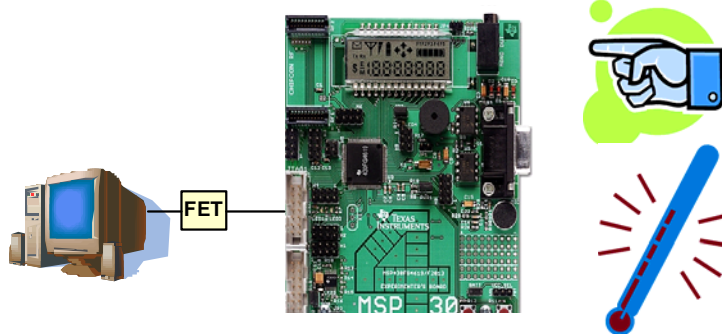
*** www.this-page-intentionally-left-blank.org no kidding***

Lab 4 – Using the ADC12

In this lab we'll configure and use the ADC12 analog input in the FG4618/9 to measure the temperature from the internal thermistor.

Lab4: Using the ADC12

- ◆ Use ADC12 integrated temperature sensor
- ◆ Set up ADC12 to perform a single conversion
- ◆ Loop continuously, converting to Degrees F and C in software
- ◆ Touch the FG4618/9 with a finger to change the temperature
- ◆ Open a watch window in the debugger to see the temperature values



[Review](#)

Hardware list:

- WinXP PC
- MSP-FET430UIF
- USB cable
- JTAG ribbon cable
- MSP430FG461x/F28xx Experimenter's Board
- Jumpers

Software list:

- IAR Kickstart for MSP430 version 4.21B
- Code Composer Studio 4.1
- Labs
- Additional pdf documentation
- Adobe™ Reader

IAR Kickstart Procedure

In this lab we'll configure and use the ADC12 analog input in the FG4618/9 to measure the temperature from the internal thermistor. Touching the MSP430 will change the temperature enough to measure it, calculate it and place it in a memory for observation.

Start Up

1. JTAG

Assure that the JTAG interface is connected to the FG4618/9 debug port.

2. New Workspace, New Project

Start up *IAR Kickstart* and **create** a new workspace, **Create** a new project named **Lab4** and save it in the **C:\MSP430\IAR Labs\Lab4** folder.

3. Configure the Project Options

Target device = **MSP430FG4618** or **MSP430FG4619**

Driver = **FET Debugger**

FET Debugger = **Texas Instrument USB-IF**

Add Source File

4. Add the source file to the project

Add **Lab4_exercise.c** from the **C:\MSP430\IAR Labs\Lab4** folder.

Complete the Code

The following lab steps will walk you through filling in the blanks in the code as shown on the facing page. You'll want to **open** the *MSP430x4xx Family User's Guide (slau056g.pdf)*, as well as the MSP430FG4618/9 datasheet (**msp430fg4618.pdf** or **msp430fg4619.pdf**). We're also going to need to look at the standard definitions in the header file:

C:\Program Files\IAR Systems\Embedded Workbench 5.0\430\inc\msp430xG46x.h.

Open that file in the *IAR Kickstart* editor.

If you want to take the easy way out, you'll find the completed code in the Addendum chapter at the end of this workbook or you can peruse the solution file in the Lab4 folder.

If you're a glutton for punishment, ignore the following steps and do it on your own. No one's forcing you to do it our way!

```

#include "msp430xG46x.h"

volatile unsigned int i;
volatile unsigned int ADCresult;
volatile unsigned long int DegC, DegF;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;           // Stop watchdog timer
    ADC12CTL0 = _____;           // Turn ADC on, ref on. Ref = 2.5V,
                                        // Set sampling time
    ADC12CTL1 = _____;           // Use sampling timer
    ADC12MCTL0 = _____;           // Channel A10, Vref+ & AVSS
    ADC12IE = 0x01;                     // Enable ADC12IFG.0
    for (i = 0; i < 0x3600; i++);        // Delay for reference start-up
    ADC12CTL0 |= ENC;                    // Enable conversions
    __enable_interrupt();                // Enable interrupts

    while(1)
    {
        ADC12CTL0 |= _____;        // Start conversion
        __bis_SR_register(LPM0_bits);    // Enter LPM0

        // DegC = (Vsensor - 986mV)/3.55mV
        // Vsensor = (Vref)(ADCresult)/4095
        // DegC -> ((ADCresult - 1615)*704)/4095
        DegC = (((long)ADCresult-1615)*704)/4095);
        DegF = ((DegC * 9/5)+32);         // Calculate DegF
        __no_operation();                 // SET BREAKPOINT HERE
    }
}

#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR(void)
{
    ADCresult = ADC12MEM0;               // Move results, IFG is cleared
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}

```

5. **ADC12CTL0** = _____;

Search [slau056f.pdf](#) for **ADC12CTL0**. Somewhere in there you'll find the bit field layout for the register. Search the header file for the same thing. Under about the fourth occurrence you'll see the definitions for the individual bit fields.

Look around and find the following definitions:

Turn ADC12 on: _____

Turn ADC12 reference on: _____

Set the reference to 2.5V: _____

Set the sampling time: _____

This last one is a little harder than the first three. First, we need to find out how fast we can sample the temperature sensor. Search the [msp430fg4618.pdf](#) or [msp430fg4619.pdf](#) datasheet for t_{SENSOR} and you'll see the following:

$t_{\text{SENSOR(sample)}}$	Sample time required if channel 10 is selected (see Note 3)	ADC12ON = 1, INCH = 0Ah, Error of conversion result ≤ 1 LSB	2.2 V	30	μs
			3 V	30	

The ADC12 in this lab is set up to use the **ADC12OSC** as the clock source, so search the datasheet for f_{ADC12OSC} and find the following:

f_{ADC12OSC}	Internal ADC12 oscillator	ADC12DIV=0, $f_{\text{ADC12CLK}}=f_{\text{ADC12OSC}}$	$V_{\text{CC}} = 2.2 \text{ V} / 3 \text{ V}$	3.7	5	6.3	MHz
-----------------------	---------------------------	--	---	-----	---	-----	-----

So, we need to make sure that the sampling timer uses enough clock cycles in the sample period to guarantee we meet the 30 μs sampling time required by the temperature sensor. **Calculate** the clock cycles needed and **select** a sampling time that has at least that many cycles.

Take all those definitions, put + signs in between them and **type** them in the proper blank. By the way, the order doesn't matter since the definitions are all 16-bits.

6. **ADC12CTL1** = _____;

This one's easy. Look in the header file for **ADC12CTL1** and find the correct mode setting for the sample/hold field. Check the datasheet too, if necessary.

7. **ADC12MCTL0** = _____;

You should have it down by now, but this time search the header file for **ADC12CTLx**. Look in the definitions for **Input Channel 10**. You also have to select $V_{\text{REF+}}$ using the **SREFx** field. A quick look at the mux near the top of the ADC12 block diagram in [spau056g.pdf](#) will give you a clue which one to pick.

8. ADC12CTL0 |= _____;

Last one. You should find it pretty quickly if you look in the header file under **ADC12CTL0**.

Test Your Work

9. Build and Download

You know what to do by now. Correct any errors you may find. When prompted to save your workspace, save it in the **Lab4** folder as **Lab4.eww**.

10. Set a Breakpoint

Set a breakpoint on the line with the comment **//SET BREAKPOINT HERE** (wow, that was tough). If you've already looked through the code, you'll see that this line is right after the temperature calculations are complete

11. Set a Watch

Right click on the **DegC** or **DegF** variable in the code right before the breakpoint. Select **Add to Watch** from the drop down menu. Right now the value should be 0.

12. Run

Run the code and it will quickly stop at the breakpoint you set. **Observe** the temperature in the **Watch** window on the right of the screen. Keep clicking the **Go** button while you place your finger on the FG4618/9 and watch the temperature rise.

Unfortunately, we didn't properly calibrate the temperature before we started, so the temperature isn't very accurate. But it's close enough to understand the ADC12 functions.

13. Additional Information

Did you notice the line in the initialization with the comment **//Delay for reference start-up ?** The ADC12 module has a shortcoming, in that a 17mS delay is required after initializing the ADC in order for the reference to stabilize. A software loop is a terrible waste of cycles, but in this case we thought it would be simpler from a coding perspective.

If you have the time and the motivation, how about eliminating the loop and using Timer_A to delay those 17mS? Let your instructor know that you're going to give this a try!

Shut Down

14. Shut Down

When done, click the **Stop Debugging** button and **close IAR Embedded Workbench**..



IAR Kickstart users ... you're done. Proceed to the review questions on page 3-21

Code Composer Studio 4.1 Procedure

In this lab, we'll configure and use the ADC12 analog input in the FG4618/9 to measure the temperature from the internal thermistor. Touching the MSP430 will change the temperature enough to measure it, calculate it and place it in a memory for observation.

Start Up

1. JTAG

Assure that the JTAG interface is connected to the FG4618/9 debug port.

2. New Workspace, New Project

Start up CCS and **create** a new workspace at **C:\MSP430ODW\CCS Labs\Lab4\workspace**. **Create** a new project named **Lab4** in the newly created workspace folder. Make sure the:

Project Type = MSP430

Device Variant = **MSP430FG4618** or **MSP430FG4619**

Add Source File

3. Add the source file to the project

Add **Lab4_exercise.c** from the **C:\MSP430\CCS Labs\Lab4** folder.

Complete the Code

The following lab steps will walk you through filling in the blanks in the code as shown on the facing page. You'll want to **open** the *MSP430x4xx Family User's Guide* (**slau056g.pdf**), as well as the MSP430FG4618/9 datasheet (**msp430fg4618.pdf** or **msp430fg4619.pdf**). We're also going to need to look at the standard definitions in the **msp430xG46x.h** header file:

Find the line in the code containing **#include "msp430xG46x.h"**. **Right-click** on the line and select **Show In**, then **Outline**. In the Outline pane (on the right), **double-click** on **msp430xG46x.h** to open that file in the editor.

If you want to take the easy way out, you'll find the completed code in the Addendum chapter at the end of this workbook or you can peruse the solution file in the Lab4 folder.

If you're a glutton for punishment, ignore the following steps and do it on your own. No one's forcing you to do it our way!

```

#include "msp430xG46x.h"

volatile unsigned int i;
volatile unsigned int ADCresult;
volatile unsigned long int DegC, DegF;

void main(void)
{
    WDTCTL = WDTPW + WDTHOLD;           // Stop watchdog timer
    ADC12CTL0 = _____;           // Turn ADC on, ref on. Ref = 2.5V,
                                        // Set sampling time
    ADC12CTL1 = _____;           // Use sampling timer
    ADC12MCTL0 = _____;           // Select channel A10, Vref+
    ADC12IE = 0x01;                     // Enable ADC12IFG.0
    for (i = 0; i < 0x3600; i++);        // Delay for reference start-up
    ADC12CTL0 |= ENC;                     // Enable conversions
    __enable_interrupt();                 // Enable interrupts

    while(1)
    {
        ADC12CTL0 |= _____;       // Start conversion
        __bis_SR_register(LPM0_bits);   // Enter LPM0

        // DegC = (Vsensor - 986mV)/3.55mV
        // Vsensor = (Vref)(ADCresult)/4095
        // DegC -> ((ADCresult - 1615)*704)/4095
        DegC = (((long)ADCresult-1615)*704)/4095);
        DegF = ((DegC * 9/5)+32);        // Calculate DegF
        __no_operation();                // SET BREAKPOINT HERE
    }
}

#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR(void)
{
    ADCresult = ADC12MEM0;               // Move results, IFG is cleared
    __bic_SR_register_on_exit(LPM0_bits); // Exit LPM0
}

```

4. ADC12CTL0 = _____;

Search [slau056f.pdf](#) for **ADC12CTL0**. Somewhere in there you'll find the bit field layout for the register. Search the header file for the same thing. Under about the fourth occurrence you'll see the definitions for the individual bit fields.

Look around and find the following definitions:

Turn ADC12 on: _____

Turn ADC12 reference on: _____

Set the reference to 2.5V: _____

Set the sampling time: _____

This last one is a little harder than the first three. First, we need to find out how fast we can sample the temperature sensor. Search the [msp430fg4618.pdf](#) or [msp430fg4619.pdf](#) datasheet for t_{SENSOR} and you'll see the following:

$t_{\text{SENSOR(sample)}}$	Sample time required if channel 10 is selected (see Note 3)	ADC12ON = 1, INCH = 0Ah, Error of conversion result ≤ 1 LSB	2.2 V	30	μs
			3 V	30	

The ADC12 in this lab is set up to use the **ADC12OSC** as the clock source, so search the datasheet for f_{ADC12OSC} and find the following:

f_{ADC12OSC}	Internal ADC12 oscillator	ADC12DIV=0, $f_{\text{ADC12CLK}}=f_{\text{ADC12OSC}}$	$V_{\text{CC}} = 2.2 \text{ V} / 3 \text{ V}$	3.7	5	6.3	MHz
-----------------------	---------------------------	--	---	-----	---	-----	-----

So, we need to make sure that the sampling timer uses enough clock cycles in the sample period to guarantee we meet the 30 μs sampling time required by the temperature sensor. **Calculate** the clock cycles needed and **select** a sampling time that has at least that many cycles.

Take all those definitions, put + signs in between them and **type** them in the proper blank. By the way, the order doesn't matter since the definitions are all 16-bits.

5. ADC12CTL1 = _____;

This one's easy. Look in the header file for **ADC12CTL1** and find the correct mode setting for the sample/hold field. Check the datasheet too, if necessary.

6. ADC12MCTL0 = _____;

You should have it down by now, but this time search the header file for **ADC12CTLx**. Look in the definitions for **Input Channel 10**. You also have to select $V_{\text{REF+}}$ using the **SREFx** field. A quick look at the mux near the top of the ADC12 block diagram in [spau056g.pdf](#) will give you a clue which one to pick.

7. ADC12CTL0 |= _____;

Last one. You should find it pretty quickly if you look in the header file under **ADC12CTL0**.

Test Your Work

8. Build and Download

You know what to do by now. Correct any errors you may find.

9. Set a Breakpoint

Set a breakpoint on the line with the comment **//SET BREAKPOINT HERE** (wow, that was tough). If you've already looked through the code, you'll see that this line is right after the temperature calculations are complete

10. Set a Watch

Double-click on the **DegC** or **DegF** variable in the code right before the breakpoint. Right-click on the selected variable, then select **Add Watch Expression** from the drop down menu. You should see a Watch tab in the upper right pane in CCS. If the Watch pane isn't already open, click on the tab now.

11. Run

Run the code and it will quickly stop at the breakpoint you set. **Observe** the temperature in the **Watch** pane. Keep clicking the **Run** button while you place your finger on the FG4618/9 and watch the temperature rise.

Unfortunately, we didn't properly calibrate the temperature before we started, so the temperature isn't very accurate. But it's close enough to understand the ADC12 functions.

12. Additional Information

Did you notice the line in the initialization with the comment **//Delay for reference start-up ?** The ADC12 module has a shortcoming, in that a 17mS delay is required after initializing the ADC in order for the reference to stabilize. A software loop is a terrible waste of cycles, but in this case we thought it would be simpler from a coding perspective.

If you have the time and the motivation, how about eliminating the loop and using Timer_A to delay those 17mS? Let your instructor know that you're going to give this a try!

Shut Down

13. Shut Down

When done, click the **Terminate All** button and **close** *Code Composer Studio*.



Code Composer Studio users ... you're done

Review Questions

Review

- ◆ What is your lowest power option for triggering an ADC?
- ◆ Name the four ADC conversion modes:
- ◆ What is the purpose of the DTC?
- ◆ ADC10 and ADC12 can sample at what speed?

You can find the answers to these questions in the Addendum section at the end of this workbook.

*** Why can't we do this outside? ***