Lab 2 – Analog-to-Digital Converter
on
TI TMS320F28027 Piccolo Microcontroller
and "Real-Time Emulation Mode"

NOTE: THIS DOC IS WRITTEN ASSUMING A LAUNCHPAD IS BEING USED; IF YOU HAVE ANOTHER PLATFORM, THERE SHOULD BE ONLY MINOR DIFFERENCES TO WATCH OUT FOR

NOTE: SOME OF THE PICS IN THIS DOC WERE TAKEN WITH CCS VERSION 5; VERSION 7 SHOULD BE SIMILAR

In this lab you will:

- Record part-specific information for your device including:
  - PARTID
  - CLASSID
  - REVID
  - ADCREFTRIM
  - ADCOFFTRIM
  - temperature sensor slope
  - temperature sensor offset
- Gain experience using the on-chip A-D converter by writing code to:
  - initialize the A-D
  - monitor the junction temperature with the on-chip temperature sensor
    - use a repeated series of single triggers via software control
- Experiment with the Real-Time Emulation mode:
  - observe global variables in the Expressions window update on-the-fly
  - observe Graph of signal versus time:
    - temperature sensor reading stored in a single variable
- Understand write-protection of peripheral registers and how to control it by use of the EALLOW and EDIS macros
- Familiarize yourself with an erratum from the Errata document and account for it in the temperature sensor monitoring code you write

Documents to refer to (on D2L under c2000 Documentation):

sprz292m (or newer) - TMS320F2802x Silicon Errata.pdf

spruge5f (or newer) - TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator Reference Guide.pdf

sprs523k (or newer) - Piccolo Microcontrollers Datasheet.pdf

Other documents to refer to (on D2L under Lab 2):

spru430f (or newer) - TMS320C28x CPU and Instruction Set Reference Guide pp41-42.pdf

sprufn3d (or newer) - TMS320F2802xTMS320F Piccolo System Control and Interrupts Reference Guide 32.pdf

sprs523k (or newer) - Piccolo Microcontrollers Datasheet 53-54.pdf

What to submit to D2L (use .zip file):

include in Word doc

- answers to any questions posed in this handout
- screenshot(s) showing part-specific information
- screenshot showing Graph Properties for real-time graph of temperature sensor versus time
- screenshot showing real-time graph of temperature sensor versus time for one SOC (SOC0)
- screenshot showing real-time graph of temperature sensor versus time for two SOC's (SOC0 and SOC1)
- any code you write or modify, ***with appropriate commenting***

Create Project and Record Part-Specific Information

1. Create a CCS Project, call it "ELEX7820-Lab2tsensor".

2. Add the original Project Files from Lab 1 on D2L (rename any "Lab 1" to "Lab 2").

3. Build the project, enter the Debugger, and run the code to verify that the code works by flashing the LED once per second.

4. Use the Registers window to find and record the following information for your device:
    a. PARTID
    b. CLASSID
    c. REVID

What are the addresses of these registers - where can you look to find them?

What do the values of the above registers signify?

5. Use the Registers window to find and record values in decimal of the following for your device:
    a. ADCREFTRIM
    b. ADCOFFTRIM

and also record their addresses.

What do the values of the above registers signify?

6. Modify your **main** program to view the calibration data for the on-chip temperature sensor:
    a. Add these defines:

```
#define getTempSlope (*(int (*)(void))0x3d7e80)
#define getTempOffset (*(int (*)(void))0x3d7e83)
```

    b. Declare two global variables of type **`int16`** called **`temp_slope`** and **`temp_offset`**
    c. Add this code to the initialization area:

```
temp_slope = getTempSlope();

temp_offset = getTempOffset();
```

    d. Build and run your code again. Observe and record the values of the slope and offset correction factors.

How do the values compare to the typical ones in the datasheet?

In what format can you observe `temp_slope` that is more meaningful than "integer"?

Use A-D's SOC0 to Monitor Processor's Junction Temperature

1. Modify your **DevInit** code to:
   a. Enable the A-D clock by setting ADCENCLK under the PERIPHERAL CLOCK ENABLES section.
   b. After the GPIO CONFIG section, add code to do in the following order:
      i. simultaneously power up ADC's analog circuitry, bandgap, and reference buffer
      ii. generate INT pulse on end of conversion (not start of conversion)
      iii. enable ADC
      iv. wait 1 ms after power-up before using the ADC by inserting:

```
DelayUs(1000);
```

      and adding at the top of the file after the #include section:

```
extern void DelayUs(Uint16);
```

      and Adding File **DelayUs.asm** to your Project.

      v. configure A-D to sample the on-chip temperature sensor by setting:
         - TEMPCONV
         - CHSEL for SOC0
         - ACQPS for SOC0
         - INT1SEL to connect interrupt ADCINT1 to EOC0
         - INT1E to enable interrupt ADCINT1

Where can you look to read up on the meaning of these registers?

Make sure the registers are temporarily not-write-protected when you write to them.

2. Modify your **main** code to:
   a. Change the LED flashing rate from 1000 ms to 100 msec.
   b. Add another global variable of type **`int16`** called **`temp_reading`**.
   c. In the section of code where the LED toggles, add code to:
      i. start the conversion of SOC0 via the ADCSOCFRC1 register
      ii. wait for the ADCINT1 bit to activate
      iii. clear the ADCINT1 bit
      iv. read the result for SOC0

3. Observe the temperature reading in the Expressions window. You need to Resume-Suspend, Resume-Suspend, etc to see it changing.

Use Real-Time Emulation Mode to Monitor Junction Temperature in the Expressions Window

1. Suspend the processor.

2. To invoke the Real-Time mode:
   a. Scripts-Realtime Emulation Control-Run_Realtime_with_Restart ← Might not need to do in CCSv7
   b. Run. ← Might not need to do in CCSv7
   c. In the Expressions window, press the Continuous Refresh button.

3. Observe that the value of the temperature reading updates automatically without a need to Suspend the processor.

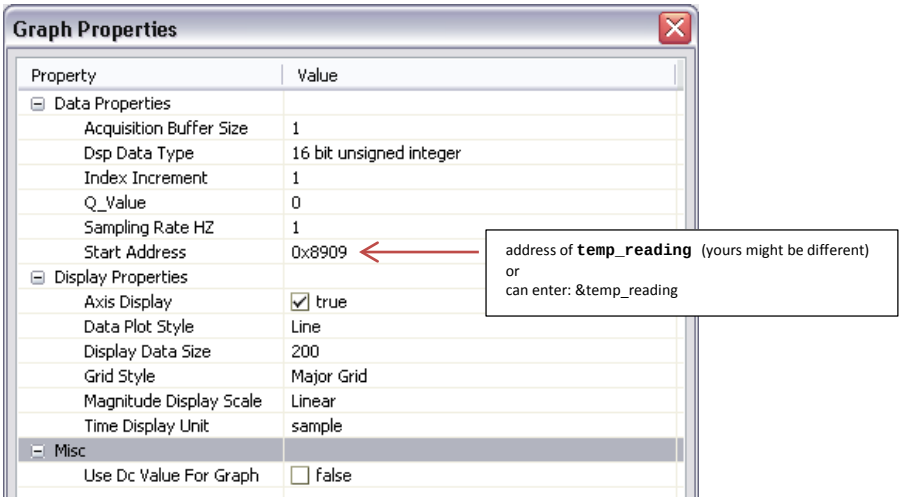
4. You can stop the Continuous Refresh by pressing it again.

5. To change the refresh period:
   a. Window-Preferences-Code Composer Studio-Debug
   and change “Continuous refresh interval”.

6. To exit Real-Time mode: ← Might not need to do in CCSv7
   a. Scripts-Realtime Emulation Control-Full_Halt (if you don’t do this I have found crazy system errors can occur later)

Use Real-Time Emulation Mode to Monitor Junction Temperature in Graph Window

1. Invoke Graph by:
   a. Tools-Graph-Single Time
   b. Modify the Graph Properties for graphing a single variable:



2. A Graph window will open.

3. Invoke the Real-Time mode:
   a. Scripts-Realtime Emulation Control-Run_Realtime_with_Restart ← Ditto
   b. Run. ← Ditto
   c. In the Graph window, press the Continuous Refresh button.

4. Observe that the value of the temperature reading updates automatically. Estimate the peak-to-peak variation in the reading.

5. You can stop the Continuous Refresh by pressing it again.

6. To change the refresh period:
   a. Window-Preferences-Code Composer Studio-Debug
   and change "Continuous refresh interval".

7. To exit Real-Time mode: ← Ditto
   a. Scripts-Realtime Emulation Control-Full_Halt (if you don't do this I have found crazy system errors can occur later)

After discharging yourself, put your thumb on the processor chip when the Graph is running. **Observe that the junction temperature changes (up or down?) a bit.**

Obtain a temperature meter and measure the case temperature of the processor.

Given the measured case temperature, how does the measured junction temperature compare to the theoretical junction temperature? (Refer to the datasheet for the thermal model result for Θ_{JC}.)

Use A-D's SOC0 *and* SOC1 to Monitor Processor's Junction Temperature

According to the Errata sheets, the A-D does not read the temperature sensor accurately on each first conversion trigger. Your program continuously re-triggers (restarts) the converter so it is always providing a "first" sample, thus the samples you observe may have a bigger error than they should. One way to circumvent this is to modify the code so that upon each new trigger, two SOCs cause samples to be converted. By default, SOC0 will be first and SOC1 will be second. If you modify your code to do both an SOC0 and an SOC1 conversion each time, you can ignore the first result and read the second result. It should be more accurate, i.e., have a lower peak-to-peak variation.

1. Create another CCS Project, call it "ELEX7820-Lab2tsensor2".
2. Add the Project Files from ELEX7820-Lab2tsensor.
3. Modify your **DevInit** code to *add* SOC1 conversion:
    a. Configure to sample on-chip temperature sensor by setting:
        a. CHSEL for SOC1 (retain SOC0 code)
        b. ACQPS for SOC1 (retain SOC0 code)
        c. INT1SEL to connect interrupt ADCINT1 to EOC1, not EOC0
        d. retain INT1E to enable interrupt ADCINT1

4. Modify your **main** code to:
    a. In the section of code where the LED toggles, modify code to:
        i. start the conversion of *both* SOC0 and SOC1 via the ADCSOCFRC1 register
        ii. retain "wait for the ADCINT1 bit to activate" code
        iii. retain "clear the ADCINT1 bit" code
        iv. read the result for SOC1, not SOC0

5. Observe the temperature reading in a Graph window in Real-Time Mode.

How does the peak-to-peak variation in the reading versus time compare to the previous code where only one conversion is done?

To better measure this, Export the Graph data to Excel and compute the rms values for the two cases.

Apply Calibration Data to Temperature Reading and View Celsius Value in Floating-Point

1. In your ELEX7820-Lab2tsensor2 project, add code to your **main** code to apply the calibration data to the temperature reading and thereby produce a more accurate result.
    a. Add a global variable of type **`float32`** called **`temp_celsius`**.
    b. Calculate the temperature in Celsius by applying the offset and slope values in **`temp_offset`** and **`temp_slope`**, respectively, to **`temp_reading`**.
    c. Observe the Celsius value in the Expressions window.

    You might have to think carefully about how to convert your integer value to a floating-point value in your C calculation statement.

    Is the Celsius value in the right ballpark for the physical temperature?