ECE 2049 Lecture 12

Administrivia

- HW5: Due today by 11:59PM EDT

- HW6: Out after class, due Tues 6/28

- Lecture 13 (Tuesday) will be posted as a recording (no live class) — but it will be good!
  - Will add office hours — update soon.

- Lab2: Now due next Wed, 6/29 by end of office hours

> IF YOU HAVE OTHER OUTSTANDING LABS (LAB0, LAB1) YOU SHOULD CONTACT ME ASAP TO LET ME KNOW YOUR STATUS (IF YOU HAVE NOT DONE SO ALREADY)

I CAN BE FLEXIBLE, BUT YOU MUST ATTEMPT ALL REQUIRED LABS (LAB0, LAB1, LAB2) TO RECEIVE A PASSING GRADE!

- Lab3: Extra bonus lab, out next week, due by Wed, 7/5

Content: Lectures 10-11
Due: Thursday, June 23 by 11:59pm EDT

Submission notes:
- For full credit, please show your work and denote your answers with a circle or a box.
- Always write or draw your diagrams neatly! We cannot be expected to GUESS what you meant to write! Some problems (such as those involving code) must be typed to be graded—the others may be handwritten (neatly!) or typed.
- Points for each problem are as indicated. Some portions of problems are marked as "BONUS," which count as extra credit.

1. (5 pts) Please complete the "Interim Course Survey" located in the Quizzes section of our course page on Canvas, located here: https://canvas.wpi.edu/courses/35856/quizzes/48849

   This survey is designed to help me plan the remainder of term. Your feedback is extremely useful for helping me accommodate everyone and create a comfortable learning environment. **Your responses are anonymous, unless you choose to include your contact information—otherwise, Canvas will just record that you completed the survey.**

2. (10 pts) For a certain application, Timer A2 has been configured as shown below with the goal of creating periodic interrupts every 0.005 seconds.

```
void runtimerA2(void)
{
    TA2CTL = TASSEL_2 | MC_1 | ID_2;
    TA2CCR0 = 1309;
    TA2CCTL0 = CCIE; // Enable timer A2 interrupt
}
```

$t_{INT} = 0.005 s$

   a. Assuming that ACLK, SMCLK, and MCLK are running at their default settings, what is the **exact time** between interrupts, t_{INT}? (Your answer should be close to 0.005 sec.)

   b. If the system clock and timer settings from this problem are used to implement some kind of time-critical system, how long until the time count is off by 0.005 seconds? Will it be fast or slow? How do you know?

   c. Write an interrupt service routine for Timer A2 for this application, using a single level of leap counting to keep the display accurate for longer.

creating periodic interrupts every 0.005 seconds.

```
void runtimerA2(void)
{
    TA2CTL = TASSEL_2 | MC_1 | ID_2;
    TA2CCR0 = 1309;
    TA2CCTL0 = CCIE; // Enable timer A2 interrupt
}
```

A | SMCLK,
UP MODE
DIVIDE BY 4

$$ACLK \Rightarrow 32768 \ Hz$$
$$SMCLK \Rightarrow 1048576 \ Hz$$

$$A_{INT} = 0.005 = \frac{MAX - CNT + 1}{(f \ SMCLK / 4)}$$

$$A_{INT, ACTUAL} = \frac{1309 + 1}{(1048576/4)} \quad \boxed{\checkmark \ 0.0049972534}$$

$$A_{INT, REPORTED} = 0.005_A$$

$$ACTUAL < REPORTED \Rightarrow \boxed{FAST,}$$

b.

$$0.005_A = \left| (X \ INTERRUPTS)(A_{INT, ACTUAL} - A_{INT, REPORTED}) \right|$$

$$0.005_A = \left| (X \ INTS)(0.005 - 0.0049723 \underline{4}) \right|$$

$$X = 1820.44 \rightarrow 1820 \ INTERRUPTS$$

$$(1820 \ INTERRUPTS)(.005_A / INT) = \boxed{9.10_A}$$

DEVICE IS FAST => SKIP A COUNT

```
TIMER = 0
LEAP_CNT = 0
ISR( ) {

    IF (LEAP_CNT < 1820){
        TIMER++;
        LEAP_CNT++;
    } ELSE {
        LEAP_CNT = 0
        //DON'T CHANGE TIMER (SKIP!)
    }
}
```

3. (5 pts) Consider the following scenario involving a configuration for a timer with periodic interrupts.

a. What is the smallest time interval, t_{INT}, that you could theoretically measure with Timer A2 using ACLK or SMCLK, assuming the default clock settings?

b. Is it a good idea to set the timer to that interval? Speculate on why or why not. (We will discuss the details in class—you don't need to have a correct answer, just think about it.)

$$TA2CCR0 = 0'$$

SMCLK IS FASTEST CLOCK

1.048576 MHz

1 TICK OF SMCLK,

$$t_{INT} = \frac{0+1}{1048576 \text{ Hz}} \approx 9.02 \times 10^{-7} \text{s}$$

$\Rightarrow$ BAD IDEA: INTERRUPT OCCURS EVERY CLOCK CYCLE!

$$f_{MCLK} = f_{SMCLK}$$

— MAIN() WILL NEVER GET TO RUN BECAUSE THERE'S ALWAYS AN INTERRUPT!

— ISR WON'T FINISH BEFORE NEXT ONE ARRIVES!

Module 9. Analog to Digital Conversion

Topics

- More on timers
- Starting Analog to Digital Conversion

Warmup: Analyzing a timer configuration

1. What is the period of the timer with the configuration below? (How often are interrupts generated?)

```
void runtimerA2(void)
{
    TA2CTL = TASSEL_2 | MC_1 | ID_3;
    TA2CCR0 = 32767;
    TA2CCTL0 = CCIE; // Enable timer A2 interrupt
}
```

$ISR()$"

$TIMER++;$

S

2. The ISR for the timer above increments a counter called `timer` on each interrupt. If `timer` = 2447, how much time has elapsed since the timer was started?

$$*_{INT} = ?$$

$$*_{INT} = \frac{32767 + 1}{1048576/8} = \frac{MAX\_CNT + 1}{f_{CLK}}$$

$$= \boxed{.25\,\text{ms}}$$

$TASSEL\_2 \Rightarrow SMCLK$
$MC\_1 \Rightarrow UP\ MODE$
$ID.3 \Rightarrow DIVIDE\ BY\ 8$

2. FIND ELAPSED TIME

2447 INTERRUPTS HAVE OCCURRED

$$(2447\ INTERVALS)(.25\,\text{ms}/INTERRUPT)$$

$$\neq \boxed{611.75\,\text{ms}}$$

Analog to Digital Converters (ADCs)

Analog to Digital Converters (ADCs), or A/D converters, have become ubiquitous in embedded applications.

- ADCs return a binary code to represent a measured voltage from within a fixed range of voltages
- Small voltages return "low valued" codes, greater voltages return "larger" codes

[handwritten annotations: '1' —— 3.3V ✓ DISCRETE ... 3.3V ANALOG CONTINUOUS ... '0' —— 0V DIGITAL ... 0V]
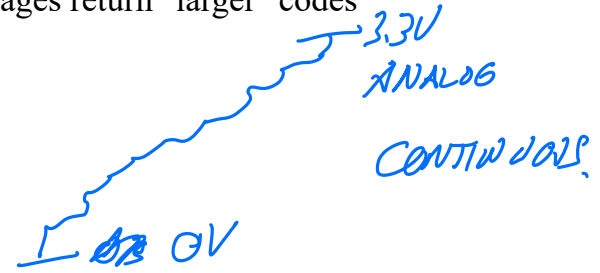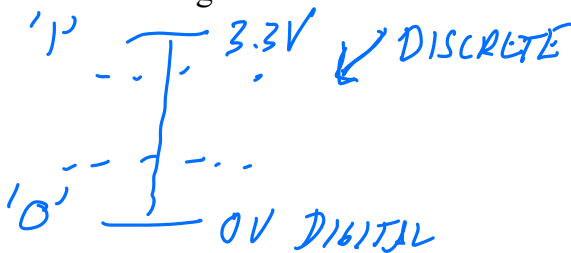
For example, within the range of 0–3V, a 10-bit ADC could return codes like these:

```
00 0000 0000b = 000h = 0d
01 1111 1111b = 1FFh = 512d
11 1111 1111b = 3FFh = 1023d
```

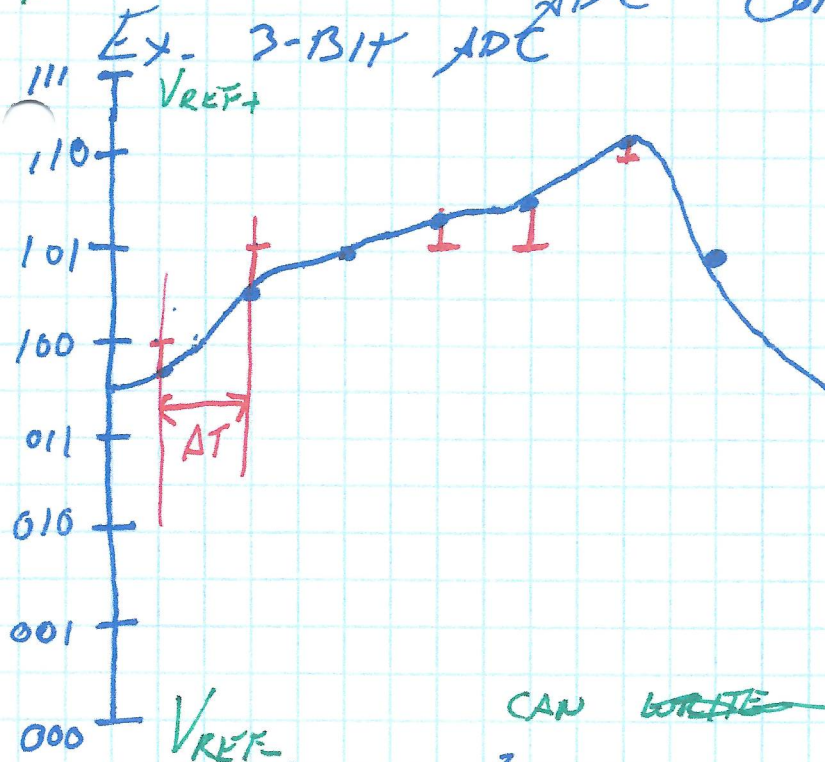*[handwritten: = > 0V ; = > 1.5V ; = > 3.0V]*

[handwritten: $2^{10} = 1024$ POSSIBLE CODES]

- It is **very** likely that you will use an ADC when you take ECE2799, do your MQP, or work on a robotics project!

[handwritten: = > SENSORS.]

EX. 3-BIT ADC $000 \rightarrow 111$

V_{REF+}

V_{REF-}

ΔT

(y-axis labels: 111, 110, 101, 100, 011, 010, 001, 000)

CAN ~~WRITE~~ FIND DIGITAL CODES FOR SAMPLES:

$$V_{OUT}[n] = \{100, 101, 101, 101, 101, 110, 101\}$$

$\Longrightarrow$ PROBLEM?

$000 = V_{REF-}$

$111 = V_{REF+}$

INPUT VOLTAGE

$$\text{CODE VALUE} = \text{FLOOR}\left(\left[\frac{V_{IN} - V_{REF-}}{V_{REF+} - V_{REF-}}\right] * (2^K - 1)\right)$$

WHERE $K = \#$ OF BITS IN ADC

$\Rightarrow$ CREATES LINEAR MAPPING ~~BETWEEN~~

FOR VOLTAGES BETWEEN V_{REF-} AND V_{REF+}

Realities of ADCs

No ADC is perfect–there will always be some error between the analog voltage and the one measured with the ADC. There are several reasons for this, including:

- Output codes are "quantized": the closest ADC code will differ somewhat from the analog voltage, depending on the resolution

- Our MSP430 uses a "sample and hold" type of ADC, which means the analog circuitry that samples the waveform can "hold" the analog value at a certain level—this means that it might miss certain changes in the waveform

- Transients from switching circuitry inside the ADC can affect the output code, which may introduce some non-linearity in the output values

WANT TO LEARN MORE? DATASHEET!

When sampling at faster rates, these effects tend to get worse!

We will not deal with these issues much in this class, but it is important to know they exist.

ADCs on Microcontrollers

Small ADCs like the 12-bit "Sample and Hold" type ADC on our MSP430 often come standard on small microcontrollers.
Are they any good?

ON MSP430: ADC12

KIND OF.

These ADCs are best suited to measuring from sensors with low to moderate data rates with a *fixed dynamic range*. Some examples:

— TEMPERATURE
— ACCELEROMETER

SAMPLE UP TO 200K/ SAMPLES/SEC

These small ADCs are likely not suitable for applications with higher data rates or a larger dynamic range. Examples:

— AUDIO PROCESSING

CD QUALITY A
16 BITS @ 44.1k SAMPLES/SEC
(44.1kHz)

As always, however, the application will determine the type of ADC you need!
What is *dynamic range*, anyway?

ADC Concepts

When an analog signal is "read" by an ADC, the analog value of the signal is *sampled* to obtain a k-bit digital representation of that value at that time.

Key Concepts for using Analog to Digital Converters
(or performing any measurements)

1. Full Scale Range (FSR): The maximum range of analog values that can be represented

This is defined as the total range of voltages between V_{REF+} and V_{REF-}
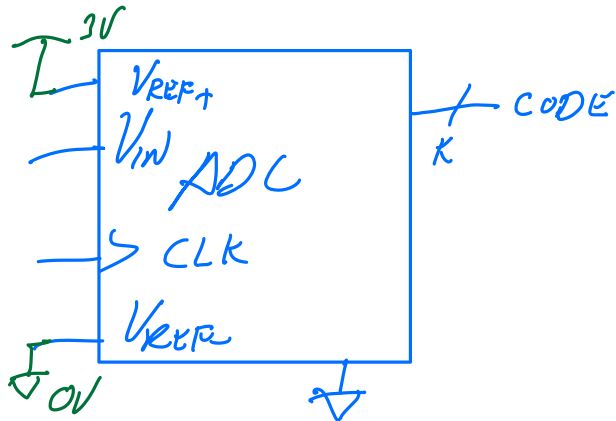
$$FSR = V_{REF+} - V_{REF-}$$

$$Ex. \quad FSR = 3V - 0V$$
$$= 3V$$

2. Resolution (for a single bit): The smallest change in value that can be measured

You can think of this as the "value of 1 bit" in an output code.

$$RESOLUTION = \frac{FSR}{2^K}, \quad K. \text{ IS NUMBER OF BITS IN ADC}$$

$$Ex. \quad FSR \text{ OF } 3.2V$$
$$K = 12$$

$$RES = \frac{3.2V}{2^{12}} = \frac{3.2V}{4096} = 0.78 \text{ mV/BIT}$$

FROM CODE N TO N+1, DIFFERENCE IS ONE UNIT OF RESOLUTION

3. Dynamic range: Ratio of largest to smallest values that can be measured

The dynamic range is usually expressed in decibels (dB), and can be computed as follows:

$$DR = 20 \log_{10}(2^K)$$
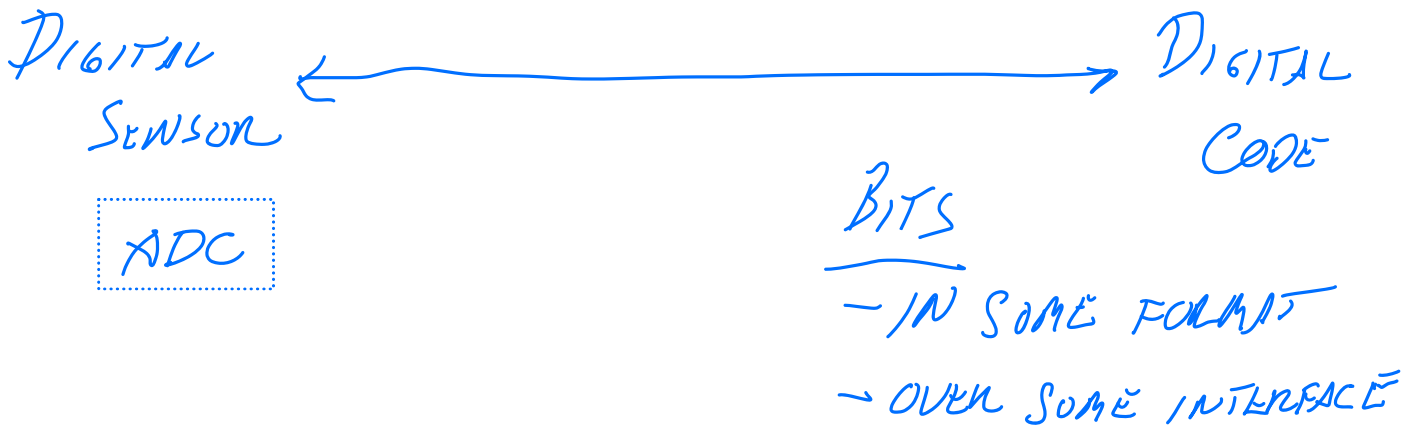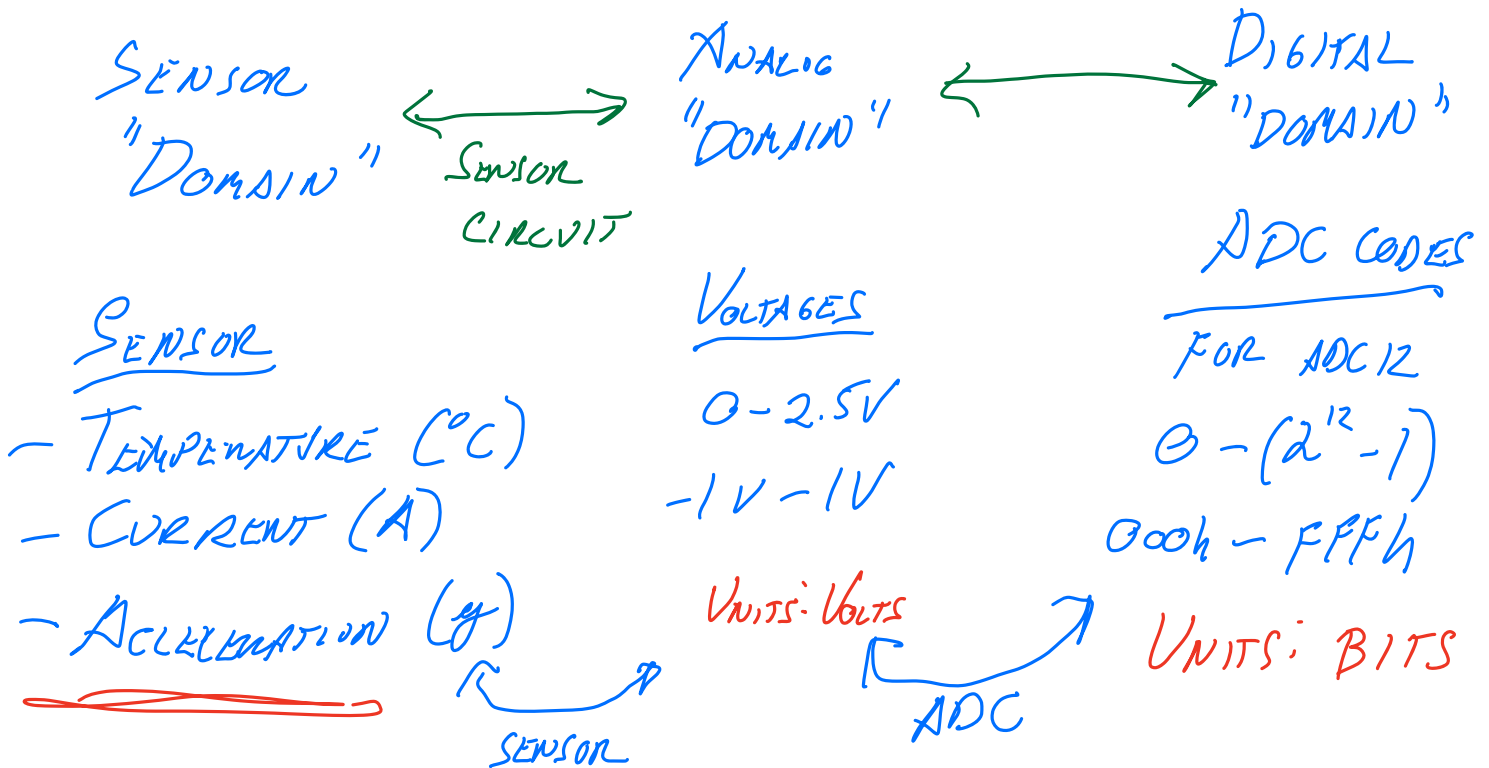
$$Ex, \quad FOR \; ADC \; 12, \quad K = 12$$

$$DR = 20 \log_{10}(2^{12})$$

$$= 72.21 dB$$

Thinking about data representations

As an embedded systems engineer, you get to decide how to make your sensors interface with the ADC! Knowing how your external sensor works and how to "map" it to the ADC you're using is as critical as knowing how to make the MSP430 read the value!

Here's a way to think about how sensor measurements are represented as digital values:

SENSOR "DOMAIN" ⟷ SENSOR CIRCUIT → ANALOG "DOMAIN" ⟷ DIGITAL "DOMAIN"

SENSOR
— TEMPERATURE (°C)
— CURRENT (A)
— ACCELERATION (g)

SENSOR →

VOLTAGES
0 - 2.5V
-1V - 1V

UNITS: VOLTS

ADC →

ADC CODES
FOR ADC12
$0 - (2^{12} - 1)$
000h - FFFh

UNITS: BITS

DIGITAL SENSOR ⟵⟶ DIGITAL CODE

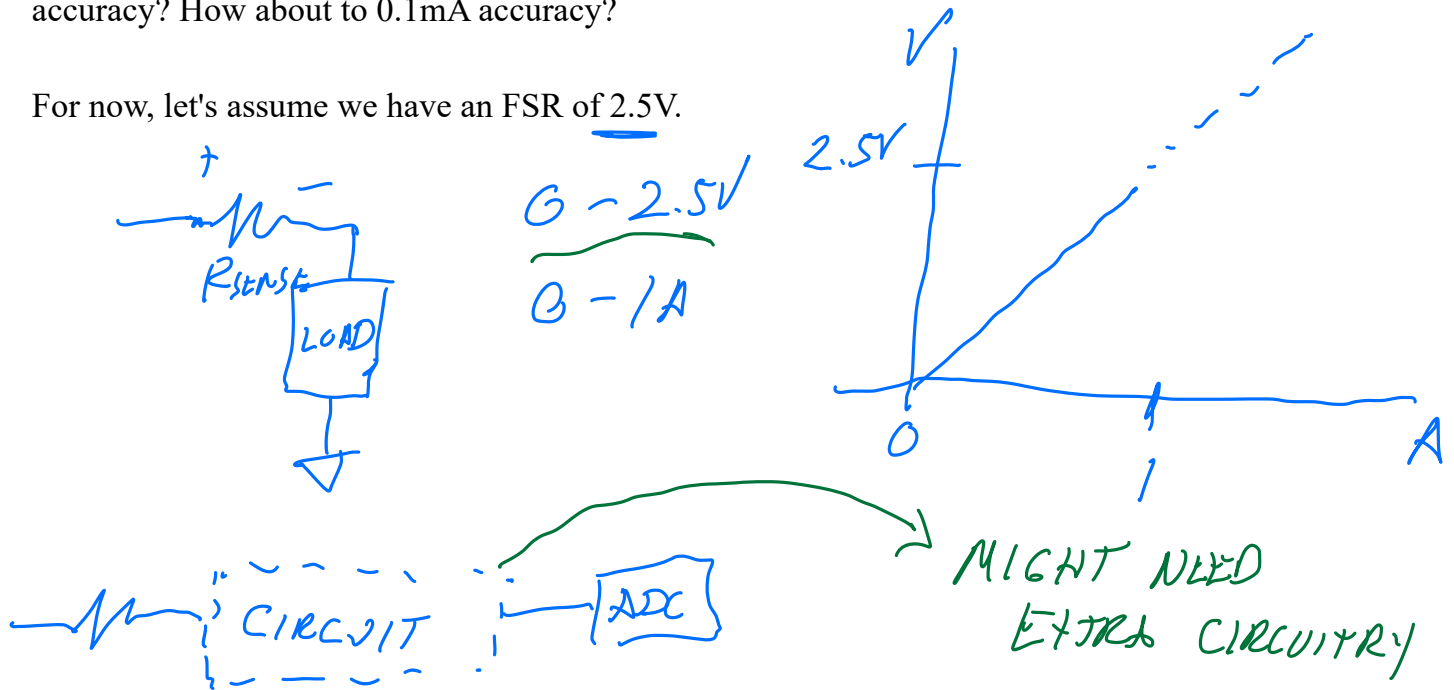ADC

BITS
— IN SOME FORMAT
— OVER SOME INTERFACE

Example: Current sensor

You can make a simple digital current meter by measuring the voltage across a small sensing resistor.

Can we use the ADC12 on the MSP430 to measure current in the range 0–1A with 1mA accuracy? How about to 0.1mA accuracy?

For now, let's assume we have an FSR of 2.5V.

$0 \sim 2.5V$
$\overline{}$
$0 - 1A$

MIGHT NEED
EXTRA CIRCUITRY

CAN WE GET 1mA ACCURACY WITH ADC12?

FSR = 2.5V   ↳ RESOLUTION

$$RES = \frac{FSR}{2^k} = \frac{2.5V}{2^{12}} = .61 \, mV/BIT$$

$$\frac{1.0 - 0A}{2^{12}} = .244 \, mA/BIT$$
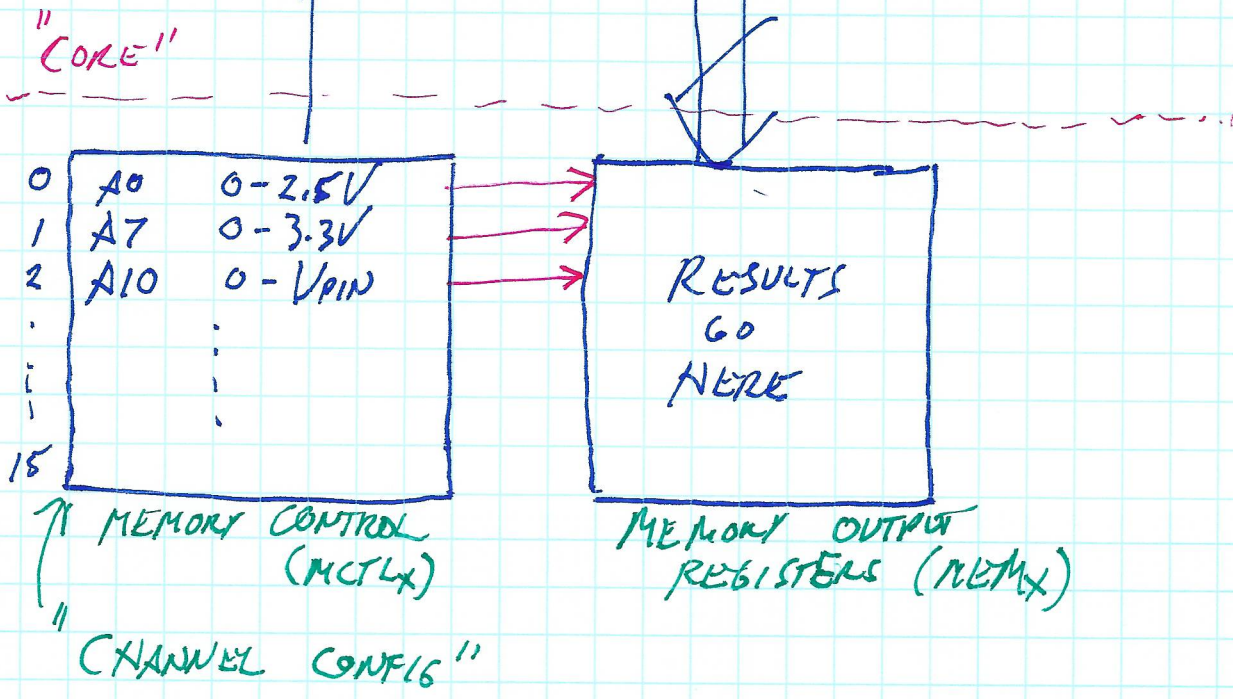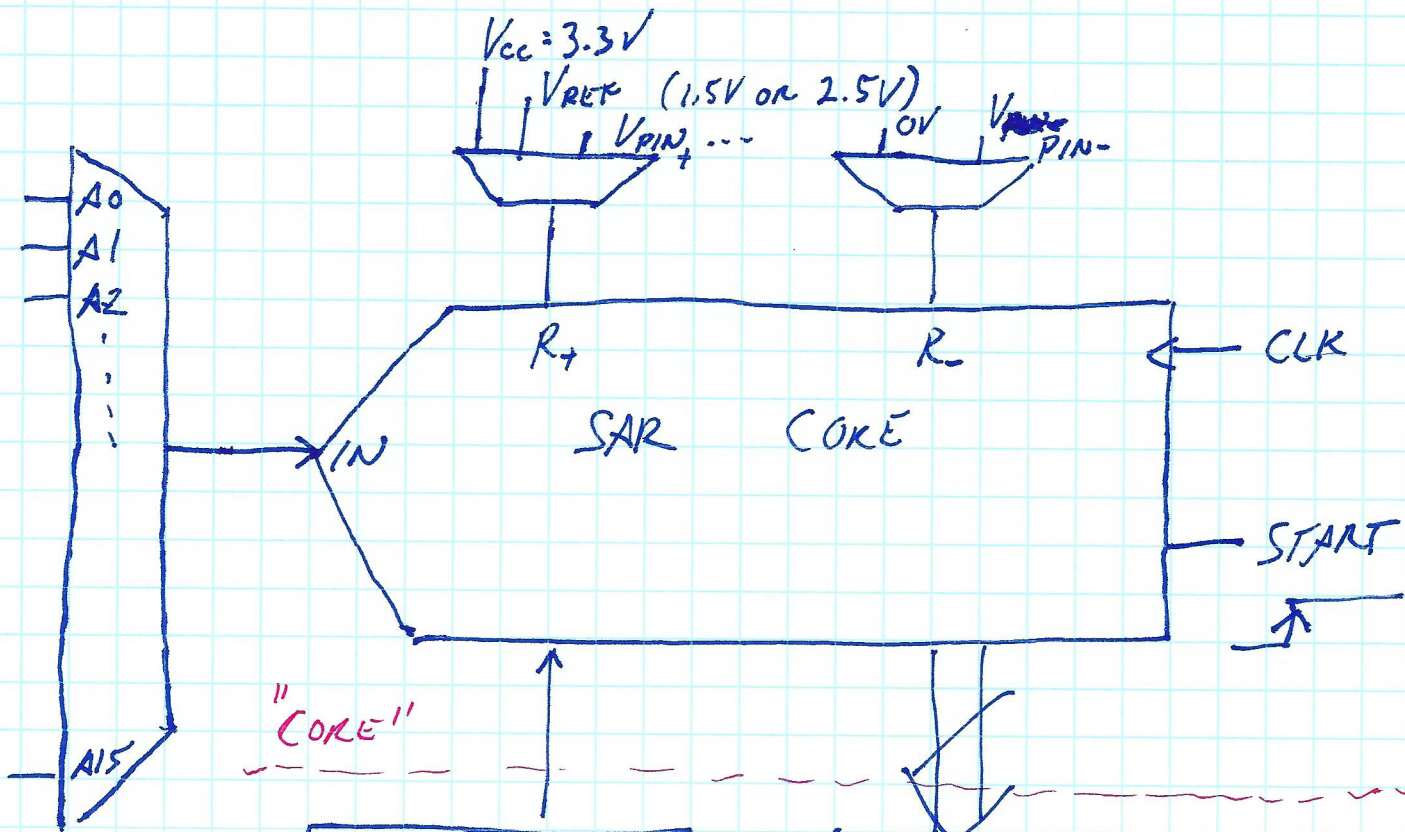
$< 1 \, mA/BIT, \therefore YES$

COULD WE GET .1 mA ACCURACY?

No.

$$.1 \, mA/BIT = \frac{FSR}{2^k}$$

∴ NO — TO IMPROVE ACCURACY, WOULD NEED TO
MEASURE OVER SMALLER RANGE, OR GET A BETTER
ADC.

On the MSP430: Using the ADC12

Our MSP430 provides includes a 12-bit ADC, called the ADC12.

About the ADC12

- 16 channel, 12-bit sample-and-hold ADC
- Maximum sample rate of 200k samples/second
- 12 External analog inputs A0-A7, A12-A15; shared with Digital I/O ports 6 and 7
- You configure and use them by setting values in various control registers

Overall ADC operation

An ADC's job is to perform a *conversion* by sampling an analog voltage into a digital value.

The ADC12 has the following components:

- Inputs from analog input channels
- Core unit to perform conversions
- *Core configuration registers* that configure how the conversion happens
- Can define multiple *channels* to perform multiple conversions at once
  - *Memory control registers* that configure how each channel should be converted
  - *Memory registers* that store the conversion results for each channel

ADC12 Control and Data registers

You can find the ADC12 register definitions in the MSP430 User's Guide (Ch. 28).

(E21)

IN THIS CLASS, YOU DO NOT
NEED TO READ/WRITE ADC12
REGISTER CONFIGURATIONS, BUT YOU
SHOULD BE FAMILIAR W/ THE CONCEPTS
(OF WORKING W/ ADCs!

Core configuration registers
The ADC12 conversion core is configured using ADC12CTL0 and ADC12CTL1.

ADC12CTL0 controls the following options:
- Sample and Hold Time (ADC12SHT1x, ADC12SHT0x): Controls sampling period

- Multiple sample conversion method (ADC12MSC)

- **Reference voltages (ADC12REF2_5V and ADC12REF_ON)**
- **ADC12ON bit**: Turns on the ADC12! (It's off by default!)

- **Enable conversions (ADC12ENC)**: Must be set to 1 before ADC will perform conversions! When set to 0, ADC can be configured.

- **Start conversion (ADC12SC)**: Starts a conversion!

- Overflow/conversion time interrupt enables (ADC12OVIE, ADC12TVIE)

ADC12CTL1 controls the following options:
- Conversion start address (ADC12STARTADDx)

- Sample and hold source select (ADC12SHS):
- Sample and hold pulse mode select (ADC12SHP): Always set this to 1
- Invert signal sample and hold (ADC12ISSH)

- ADC12 clock divider (ADC12DIVx): Typically use 1
- ADC12 clock source select (ADC12SSELx):

- Conversion mode select (ADC12CONSEQx): Can select single, multi-channel, or repeated conversions

- **ADC12 busy bit (ADC12BUSY)**

Results from each channel are stored in the low 12 bits of one 16 bit **Conversion Memory Register (ADC12MEMx)**.

Each memory register has a corresponding **Conversion Memory Control Register (ADC12MCTLx)**.

Each **ADC12MCTLx** controls one channel on which a conversion can occur. The conversion parameters for channel x is controlled by Memory Control Register x, and the result gets stored in memory register x.

Each ADC12MCTLx controls the following options:
- **Reference voltage select (ADC12SREFx)**: Important settings are as follows:

- **Analog input channel select (ADC12INCH_x)**:

- **End of Sequence (EOS)**: Set to 1 if this channel is the end of a sequence of channels. Used for multi-channel conversions.

So, as a programmer, what do you need to use the ADC12?

ADC configuration: Key steps

Step 0: Disable the ADC for configuration
- Before you can modify any ADC12 register settings, conversions must be disabled by setting ADC12ENC = 0.

Step 1: Select ADC core behavior (ADC12CTL0 and ADC12CTL1)
- Set clock source and divider
- Configure sample and hold behavior   ⎤ *DEFAULTS.*
- Select trigger source (ADC12SHS)   ⎦

- **Reference voltages**

COMMON CHOICES ~~VREFA~~ FOR V_{REF+}

$-V_{CC} = 3.3V$

$-1.5V$ ⎤ *INTERNAL REFERENCE*
$-2.5V$ ⎦ *GENERATOR* V_{REF}

$V_{REF-} = 0V$

Step 2: Select conversion mode for your application
- Configure using ADC12CONSEQx in ADC12CTL1 register
- There are four possible conversion modes:

Table 28-2. Conversion Mode Summary

ADC12CONSEQx	Mode	Operation
00	Single-channel single-conversion	A single channel is converted once.
01	Sequence-of-channels (autoscan)	A sequence of channels is converted once.
10	Repeat-single-channel	A single channel is converted repeatedly.
11	Repeat-sequence-of-channels (repeated autoscan)	A sequence of channels is converted repeatedly.

Step 3: Select input channel(s)
- What analog inputs do we need to read?
- Configure using ADC12INCHx in ADC12MCTLx registers

— SELECT WHICH PIN ON THE CHIP TO USE
A0-A7, A12-A15 ARE AVAILABLE
ON EXTERNAL PINS
— ~~FIRST~~ NEED TO SET PIN FOR FUNCTION MODE

- Analog inputs A0–A7 and A12–A15 are external analog inputs–these are **multiplexed** with Digital I/O pins on Port 6 and Port 7!
    - To use them, we need to configure the digital I/O pins for **function mode!** Ex. P6SEL | BIT7|BIT6;

- Analog inputs 8, 9, 11 are connected to the various on-chip reference voltages–you can use these to monitor the "health" of the microcontroller
- Input channel 10 is connected to an internal temperature sensor (ADC12INCH_10)

Step 4: Enable ADC interrupts, if desired (ADC12IE register)
- Using interrupts is NOT required, but useful if you are doing repeated measurements
- Also need to write ISR

Step 5: Enable ADC and start conversions
- Need to re-enable ADC so it will perform conversions (opposite of step 0)
- Start conversion process by setting ADC12SC.
- If not using interrupts, need to **poll** ADC12BUSY bit in ADC12CTL1 until conversion has finished!

ALL CONVERSIONS TAKE TIME! (≈ 250μs)

Example: Current measurement sensor
You can make a simple digital current meter by measuring the voltage across a small sensing resistor. Can we use the ADC12 on the MSP430 to measure current in the range 0–1A with 1mA accuracy? (Yes!) How about to 0.1mA accuracy? (No!)
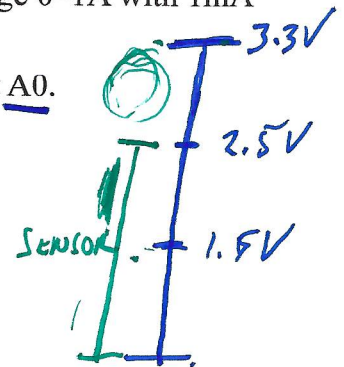
Assume we have an FSR of 2.5V and the analog voltage is connected to input A0.

What parameters do we need?

$0 - 1A \Rightarrow 0 - 2.5V$

INPUT CHANNEL: A0

REFERENCE: 1.5V, [2.5V], 3.3V

OTHER EXAMPLES

$0 - 2.1V \Rightarrow 2.5V$

$0 - 2.6 \Rightarrow 3.3V$

↘ MEETS SENSOR RANGE WHILE MAINTAINING BEST RESOLUTION.

3.3V

2.5V

SENSOR 1.5V

```c
// Current sensor conversion example
void config_adc(void) {
  /* ***** Core configuration ***** */

  // Reset REFMSTR to enable control of reference voltages by ADC12
  REFCTL0 &= ~REFMSTR;

  /*
   * Initialize control register ADC12CTL0
   * STH0x   = 9 => 384 clock cycles; MSC = 0 => no multisample mode
   * REF2_5V = 1 => Reference is 2.5V, REFON = 1 => Use internel reference generator
   * ADC12ON = 1 => Turn on ADC12
   */
  ADC12CTL0 = ADC12SHT0_9 | ADC12REFON | ADC12REF2_5V | ADC12ON;

  /*
   * Initialize control register ADC12CTL1
   * STARTADDx = 0 => Start conversion at ADC12MEM0
   * SHSx      = 0 => Conversion trigger:  Start when ADC12SC is set to 1
   * SHP       = 1 => SAMPCON sourced from sampling timer (default)
   * ISSH      = 0 => Input signal not inverted
   * SSEL      = 0 => ADC12clock = ADC12OSC (~5 MHz)
   * DIVx      = 0 => Divide ADC12CLK by 1
   * CONSEQx   = 0 => Single channel, single conversion mode
   */
  ADC12CTL1 = ADC12SHP;

  /* ***** Channel configuration ***** */

  // Set conversion memory control register ADC12MCTL0
  // SREF = 001b => Voltage refs:
  // EOS  =   0  => End of sequence not set (not a multi-channel conversion, so ig-
  nore)
  ADC12MCTL0 = ADC12SREF_1 | ADC12INCH_0;

  // Set P6.0 to FUNCTION mode
  // This connects the physical pin P6.0/A0 to the ADC input A0
  P6SEL |= BIT0;

  // Enable the ADC.  This means we are done configuring it,
  // so we can start the conversion.
  ADC12CTL0 |= ADC12ENC;
}
```

```
unsigned int read_adc(void) {
  // Input voltage has range 0-2.5V, which corresponds to 0 to 1A.
  unsigned int in_value;

  ADC12CTL0 &= ~ADC12SC;
  // Enable and start a single conversion
  ADC12CTL0 |= ADC12SC;
```

← START.

```
  // Wait for the conversion to finish by polling the busy bit
  // The busy bit is automatically set to 0 when the conversion is done
  while(ADC12CTL1 & ADC12BUSY) {
    __no_operation(); // Could also just leave the loop empty
  }
```

← WAIT FOR ADC TO BE DONE!

```
  // Now that the conversion has completed, we can read the result
  // from the memory register
  in_value = ADC12MEM0 & 0x0FFF; // Keep only the low 12 bits
  return in_value;
}
```

← GET VALUE FROM OUTPUT REGISTER.

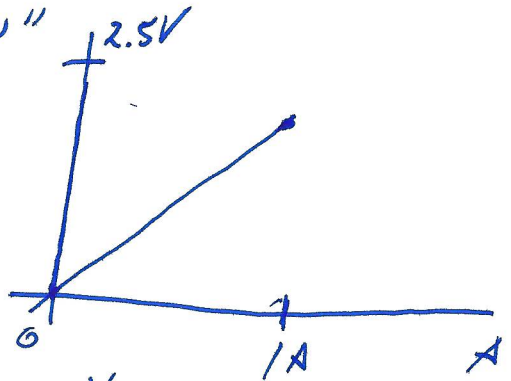Now what do we do with the return value?

IN_VALUE = 2730;   WHAT DOES THIS MEAN??

1. GET ADC CODE AS VOLTAGE "DIGITAL DOMAIN"

$$\text{CODE} = \left(\frac{V_{IN} - V_{REF-}}{V_{REF+} - V_{REF-}} \right) (2^K - 1)$$

(with annotations: V_{REF-} → 0V, V_{REF-} → 0V, $V_{REF+} - V_{REF-}$ → 2.5V)

$$2730 = \frac{V_{IN}}{2.5V} (2^{12} - 1)$$

$$V_{IN} = 1.6667 V \quad \text{ANALOG DOMAIN}$$

2. CONVERT VOLTAGE → SENSOR UNITS

$$V_{OUT} = 2.5 I$$

$$1.6667V = 2.5 I$$

$$\boxed{I = 0.666A} \quad \text{"SENSOR DOMAIN"}$$

$$y = mx + b$$

$$\frac{2.5 - 0}{1 - 0} = 2.5$$

$$V_{OUT} = 2.5(x)$$

$$= 2.5 I$$

CAN WRITE A LINEAR EQUATION FOR SENSOR BASED ON KNOWN PARAMS/GRAPH.

Example: The internal temperature sensor

To use any sensor, you need to understand how the sensor output (in this case, voltage) corresponds to the quantity it measures, which is documented by the designers.

Our MSP430 contains a built-in sensor to measure the internal chip temperature. It has a linear mapping from voltage to temperature:

A typical temperature sensor transfer function is shown in Figure 28-11 . The transfer function shown in Figure 28-11 is only an example—the device-specific data sheet contains the actual parameters for a given device. When using the temperature sensor, the sample period must be greater than 30 µs. The temperature sensor offset error can be large and may need to be calibrated for most applications. Temperature calibration values are available for use in the TLV descriptors (see the device-specific data sheet for locations).
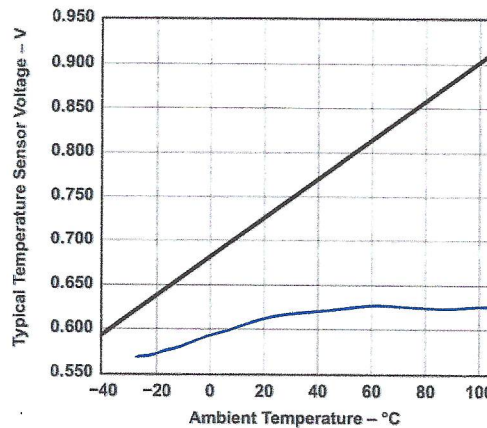

CAN USE TO GET LINEAR EQUATION FOR TEMP.

$V_{OUT} \rightarrow {}^{\circ}C$

$Y = MX + B$

(UNLIKE PREVIOUS EXAMPLE, THIS ONE HAS AN INTERCEPT.)

Figure 28-11. Typical Temperature Sensor Transfer Function

To use the sensor, we need to read some *calibration* data from the device, which we use in the computation for the resolution. This information is stored in the *Tag-Length-Value Table (TLV Table),* which is a portion of flash memory that contains some device-specific settings and constants—we need to read from the addresses specified in this table to get the calibration data. For more information on how this works, see p. 102 of the datasheet.

According to the datasheet, the calibration data provided is based on a 1.5V reference.

More ADC features

Multi-channel conversion

What if we wanted to read data from two sensors? Consider the following sensors:

- Our current sensor example from earlier (connected to input channel A0, 2.5V reference)
- A barometric pressure sensor (Input channel A4, 3.3V reference)

These sensors require different settings for reference voltages and inputs.
We *could* reconfigure the ADC every time we wanted to take a measurement, but this would be annoying. Instead, the ADC12 provides several different conversion modes to solve this problem. We will discuss the most straightforward: *multiple channel, single conversion* mode.

To perform readings from two sensors, we will need to use **two** ADC12MEMx registers, one for each channel.

Like the previous examples, we need to:

- Configure the ADC12 core, this time selecting *multiple-channel, single conversion* mode
- Configure one ADC12MEMx register for each reading we want to perform with the appropriate settings for each channel (ie, analog channel and reference voltage)