

## EC583 – Power Electronics

### A Brief-and-Abbreviated Analog-to-Digital Conversion Tutorial for the MSP430

#### **Introduction:**

An analog-to-digital converter (ADC, or A/D) is an electronic circuit that converts an analog voltage into its digital representation. The latter is an integer that is proportional (relative to an internal or external reference voltage) to the magnitude of the sampled analog voltage. While an analog signal has (theoretically) infinitely fine resolution, a digital signal can resolve only down to the voltage increment represented by its LSB.

The MSP 430 controller can perform A/D conversion using 10 bit or 12 bits. This tutorial is based on a 10-bit conversion mode, hence the binary number representing the result of a conversion can vary over the range 0 to one-bit-less than  $2^{10}$ , i.e.,  $1024 - 1 = 1023$ . All voltages are referenced to  $V_{SS} = 0$  V (ground). If a voltage being sampled has value  $v_{IN}$  and the reference voltage for conversion is  $V_{REF}$ , then the decimal number representing that input will be:

$$n = \frac{v_{IN}}{V_{REF}} \times 1023$$

#### **Performing A/D Conversion:**

Here are the steps you must take to perform an A/D operation:

- Define the port pins that will be used as analog input channels.
- Select a clock source for the conversion operation, or set up the Watchdog timer.
- Select a conversion mode (e.g., continuous, one-time conversion, etc).
- Select the reference voltage  $V_{REF}$ .
- Select the sample-and-hold time for the conversion.
- Enable the corresponding chosen pin for analog input.
- Switch ON the ADC functionality.
- Enable ADC (take a reading)
- Read the value after conversion; set a program variable to the reading.

#### **Control Registers**

The MSP430 architecture is governed by numerous control registers, which are fixed memory bytes inside the chip. The relevant registers for A/D conversion are Control Registers #0 and #1:

##### **ADC10CTL0 and ADC10CTL1**

Control registers are configured by setting their bits. Bits are set by a summation statement of various library constants that have different meaning to the chip (they represent control register bit positions.) The idea is to accumulate the various constants you need, then add them all together in a single statement of the form

**ADC10CTL1 = Constant1 + Constant2 + ...** or, similarly

**ADC10CTL1 = ConstantA + ConstantB + ....**

The idea is to accumulate the various constants you need, then add them all together into a single “ADC10CTL1 = ” type statement. (The relevant constants for A/D are predefined within the MSP430)

---

#### **Setting Up the Analog Input Pins**

For the MSP430 chips we use in class, any one of the pins of Port 1 can be set to be an analog input. Thus up to 8 channels are available for separate ADC inputs. These input channels are selected by adding **INCH<sub>n</sub>** constants to Control Register#1 (**ADC10CTL1**). To set channel as an analog input, the constant **INCH01** is added to the total value of **ADC10CTL1**. Similarly, suppose that we wish to set Channels 4 and 5 as analog inputs. The **ADC10CTL1** declaration statement would include the constants:

ADC10CTL1 = INCH\_4 + INCH\_5

The physical pins that correspond to these channels depends on which MSP430 chip you have, but the ones we use in class follow the table below. (Note that, in general, the various pins of the MSP430 can serve multiple functions depending on how the port is configured.)

**Analog Input Pins and Constants INCHx for MSP430-G2231 and MSP430-G2553:**

P1.0 = Channel 0 (A0) INCH00	P1.4 = Channel 4 (A4) INCH04
P1.1 = Channel 1 (A1) INCH01	P1.5 = Channel 5 (A5) INCH05
P1.2 = Channel 2 (A2) INCH02	P1.6 = Channel 6 (A6) INCH06
P1.3 = Channel 3 (A3) INCH03	P1.7 = Channel 7 (A7) INCH07

**Clock Source Selection:**

The ADC will perform conversions at a speed defined by either the Watchdog timer, or an internal clock. If the Watchdog timer is not used, it must be disabled, and one of several internal (and faster) clocks chosen. This latter selection is again made by adding library constants to **ADC10CTL1**:

- + ADC10SSEL00 (chooses the RC oscillator generated inside the chip)
- + ADC10SSEL01 (choose auxiliary clock)
- + ADC10SSEL10 (choose master clock)
- + ADC10SSEL11 (choose sub-main clock)

For most programs in this class, the RC oscillator (frequency of several MHz) will be fine.

**ADC Conversion Modes:**

There are four modes available for conversion:

- Single channel, Single conversion
- Sequence of Channels
- Repeat Single Channel
- Repeat Sequence of Channels

These modes are selected by adding the appropriate **CONSEQx** bit to the **ADC10CTL1** register”

- + **CONSEQ00** (one channel and one-time conversion (once))
- + **CONSEQ01 + MSC** (sequence of channels and a single conversion of all of them)
- + **CONSEQ10** (one channel repeatedly sampled)
- + **CONSEQ11 + MSC** (sequence of channels repeatedly sampled)

Note: If you want to scan a sequence of channels, you must also add **MSC** (multiple scanned channels) to **ADC10CTL1**

**Reference Voltage Selection :**

The internal voltage reference for conversion can be set to 1.5 or 2.5 V by adding the appropriate constant to the **ADC10CTL0** register.

- + **SREF00** →  $V_{CC}$  as reference voltage relative to ground
- + **SREF01** → Internal  $V_{REF}$  as reference voltage relative to ground
- + **REFON + REF2\_5V** → Set  $V_{REF}$  to 2.5 V (REFON enables this action)

**Sample And Hold Time Selection :**

The sampling (settling) time is important in ADC conversion. The longer the sampling time, the better the accuracy of the conversion. The sampling time equates to a selected number of clock cycles:

- + **ADC10SHT\_0** → 4 Clock Cycles
- + **ADC10SHT\_1** → 8 Clock Cycles
- + **ADC10SHT\_2** → 16 Clock Cycles
- + **ADC10SHT\_3** → 64 Clock Cycles

**Enable Analog Inputs :**

Setting the value of the Analog Enable control register **ADC10AEO** (in a direct program statement) will enable a specific channel (get it ready to perform a conversion), as chosen by bit position.

- + **ADC10AEO = 1** → Enable Channel 1 P1.0
- + **ADC10AEO = 16** → Enable Channel 4 P1.3
- + **ADC10AEO = 32** → Enable Channel 5 P1.4

For example, to enable channels 1 and 5, either of the following statements could be included:

```
ADC10AEO = 1 + 32;  
ADC10AEO = 33;
```

### **Enable ADC :**

The actual conversion(s) will occur when the following constants are added to **ADC10CTL0**:

```
+ ADC100N; (last two letters spell "on")  
+ ENC; (ENC stands for ENable Conversion)
```

### **Reading The Digital Value After Conversion :**

The ADC core converts an analog input to its 10 bit digital representation and stores the result in the **ADC10MEM** register. Any program variable can be set to the momentary value of **ADC10MEM** (i.e., the result of the most recent conversion). For example:

```
X = ADC10MEM;
```

where x can now be used to perform operations in C code.

### **Disable the Watchdog Timer (If Necessary)**

The following code statement will disable the Watchdog Timer if you want ADC performed according to the chosen clock. Note: It is permissible to use the timing of the Watchdog to decide when ADC will occur.

```
WDTCTL = WDTPW + WDTHOLD;
```

### **Some Simple Sample Code**

(This code would be included in a program loop to set the variable *n* to the instantaneous value of ADC10MEM for each execution of the loop.)

```
// SETUP COMMANDS:  
  
// Turn off the WatchDog Timer  
WDTCTL = WDTPW + WDTHOLD;  
  
// Set conversion to single channel and continuous-sampling mode  
ADC10CTL1 |= CONSEQ1;  
  
//Set S/H time, 10-bit converter, and continuous sampling:  
ADC10CTL0 |= ADC10SHT_2 + ADC100N + MSC;  
  
//Choose P1.0 (channel 1) as an analog input pin:  
ADC10AEO |= 1;  
  
// Start A/D conversion; The result will appear in the memory variable "ADC10MEM"  
ADC10CTL0 |= ADC10SC + ENC;  
  
// SET VARIABLE n TO THE RESULT OF THE A/D CONVERSION:  
n = ADC10MEM;
```