Analog Interfacing

These lecture notes created by Mr. James B. (Jim) Carlson, NCSU

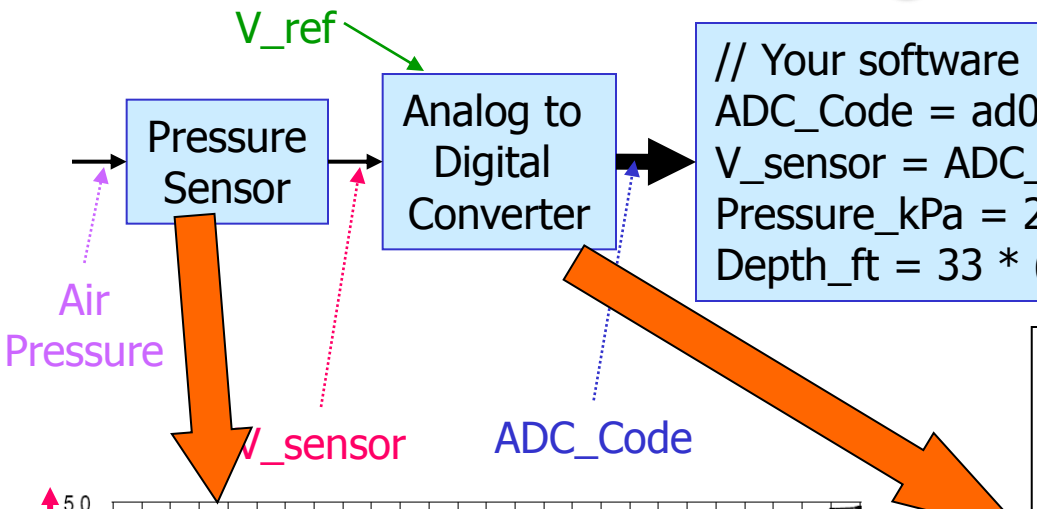
In These Notes . . .

- Analog to Digital Converters
  - ADC architectures
  - Sampling/Aliasing
  - Quantization
  - Inputs
  - MSP430 ADC Peripheral
- Digital to Analog Conversion

Why It's Needed

- Embedded systems often need to measure values of physical parameters
- These parameters are usually continuous (*analog*) and not in a digital form which computers (which operate on discrete data values) can process
- Temperature
  - Thermometer (do you have a fever?)
  - Thermostat for building, fridge, freezer
  - Car engine controller
  - Chemical reaction monitor
  - Safety (e.g. microprocessor processor thermal management)
- Light (or infrared or ultraviolet) intensity
  - Digital camera
  - IR remote control receiver
  - Tanning bed
  - UV monitor
- Rotary position
  - Wind gauge
  - Knobs
- Pressure
  - Blood pressure monitor
  - Altimeter
  - Car engine controller
  - Scuba dive computer
  - Tsunami detector
- Acceleration
  - Air bag controller
  - Vehicle stability
  - Video game remote
- Mechanical strain
- Other
  - Touch screen controller
  - EKG, EEG
  - Breathalyzer

The Big Picture




```
// Your software
ADC_Code = ad0;
V_sensor = ADC_code*V_ref/1023;
Pressure_kPa = 250 * (V_sensor/V_supply+0.04);
Depth_ft = 33 * (Pressure_kPa – Atmos_Press_kPa)/101.3;
```

Voltages	ADC Output Codes
V_ref	111..111 111..110 111..101 111..100
V_sensor	ADC_Code
Ground	000..001 000..000

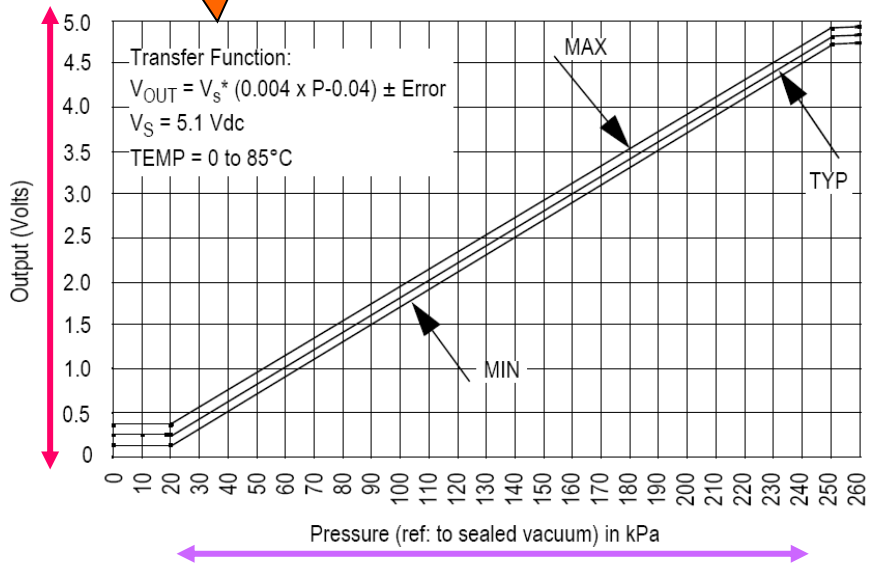
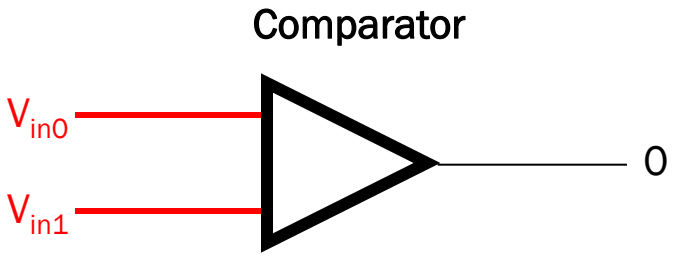


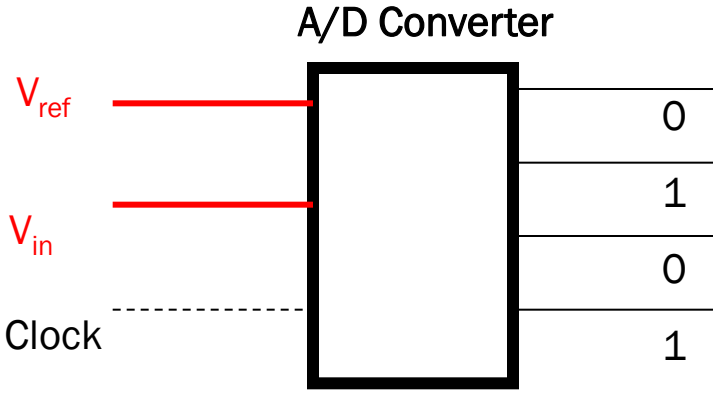
Figure 4. Output vs. Absolute Pressure

1. Sensor detects *air pressure* and generates a proportional *output voltage* V_sensor
2. ADC generates a proportional digital *integer* (code) based on V_sensor and V_ref
3. Code can convert that integer to a something more useful
    1. first a float representing the *voltage*,
    2. then another float representing *pressure*,
    3. finally another float representing *depth*

Getting From Analog to Digital

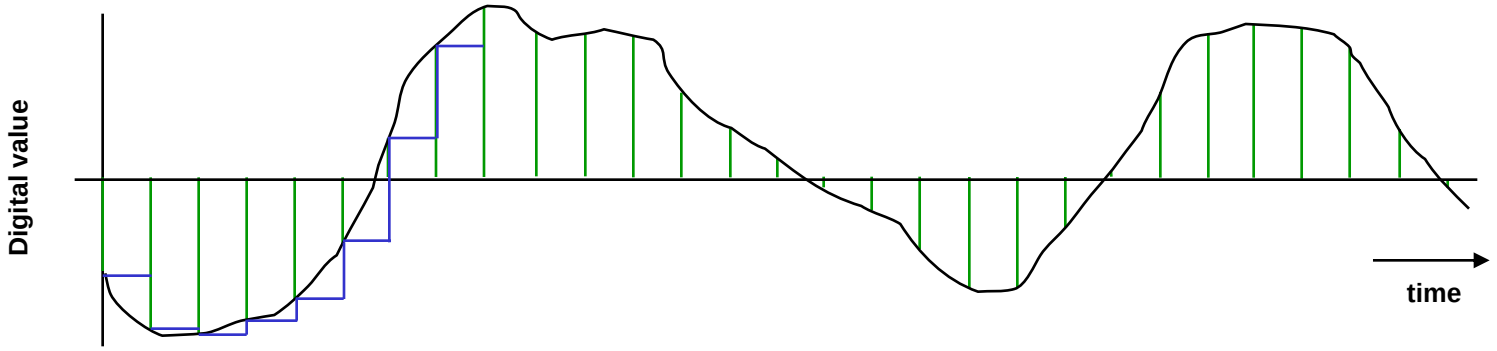
- A **Comparator** is a circuit which compares an analog input voltage with a reference voltage and determines which is larger, returning a 1-bit number
- An **Analog to Digital converter** [AD or ADC] is a circuit which accepts an analog input signal (usually a voltage) and produces a corresponding multi-bit number at the output.





- Think of looking out a window, and that which you can see is the output of the comparator.

Waveform Sampling and Quantization

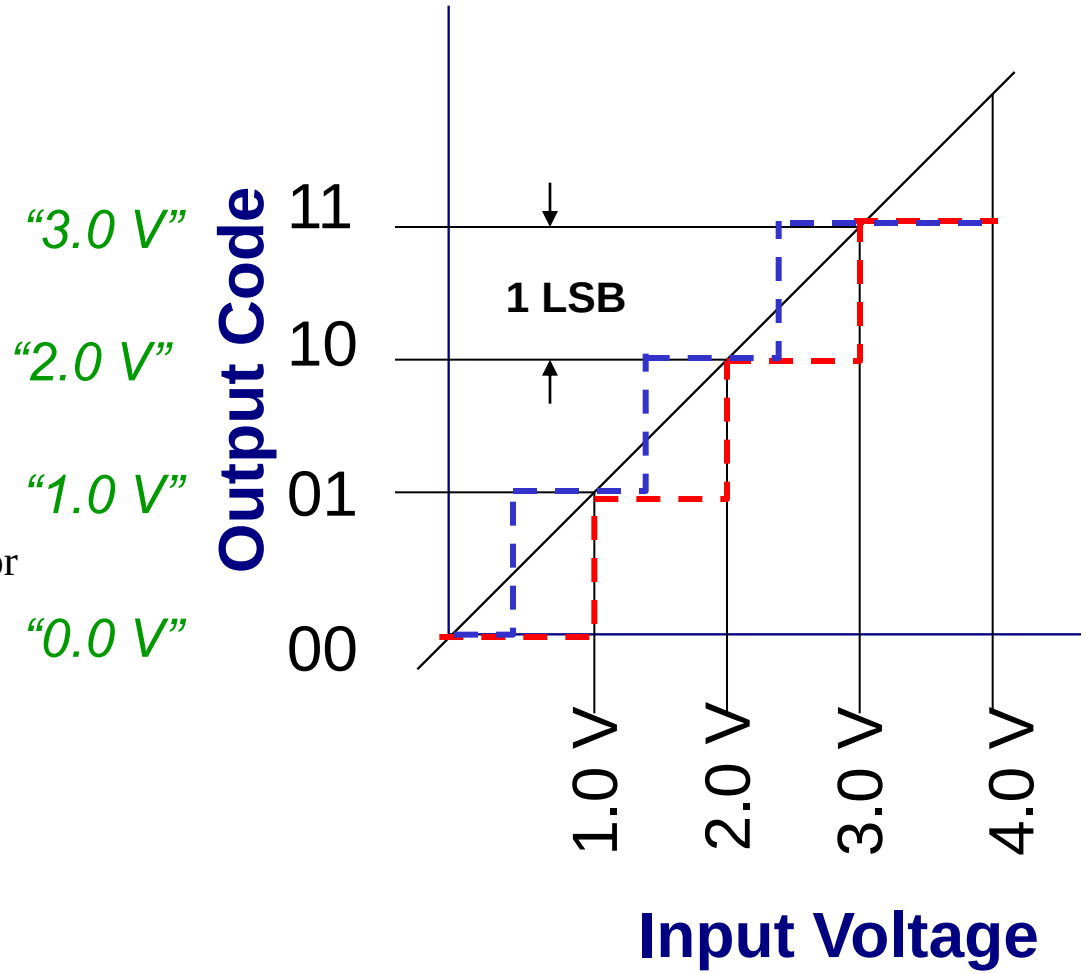


- A waveform is **sampled** at a constant rate – every Δ_t
  - Each such sample represents the instantaneous amplitude at the instant of sampling
  - *"At 37 ms, the input is 1.913419145134514512343 11… V"*
  - Sampling converts a **continuous time** signal to a **discrete time** signal
- The sample can now be **quantized** (converted) into a digital value
  - Quantization represents a **continuous** (analog) value with the closest **discrete** (digital) value
  - *"The sampled input voltage of 1.913419145134514512343 11… V is best represented by the code 0x018, since it is in the range of 1.901 to 1.9980 V which corresponds to code 0x018."*

Transfer Function

- The ADC produces a given output code for all voltages within a specific range
- The ideal transfer function A/D converter is a stair-step function.

- Consider a 2-bit ADC
  - 0 to 4 V input
  - LSB = $4/2^2 = 1$ V
- Red line
  - Truncation
  - Maximum error is -1 LSB or + 0 LSB
- Blue line
  - Rounding
  - Maximum error in conversion is usually ± 1/2 bit.
  - *Half as much error if we limit range to* $V_{ref}(1-2^N/2)$



Transfer Function Equation

General Equation

n = converted code

V_{in} = sampled input voltage

V_{+ref} = upper end of input voltage range

V_{-ref} = lower end of input voltage range

N = number of bits of resolution in ADC

$$n = \left\lfloor \frac{(V_{in} - V_{-ref})2^N}{V_{+ref} - V_{-ref}} + 1/2 \right\rfloor$$

Simplification with V_{-ref} = 0 V

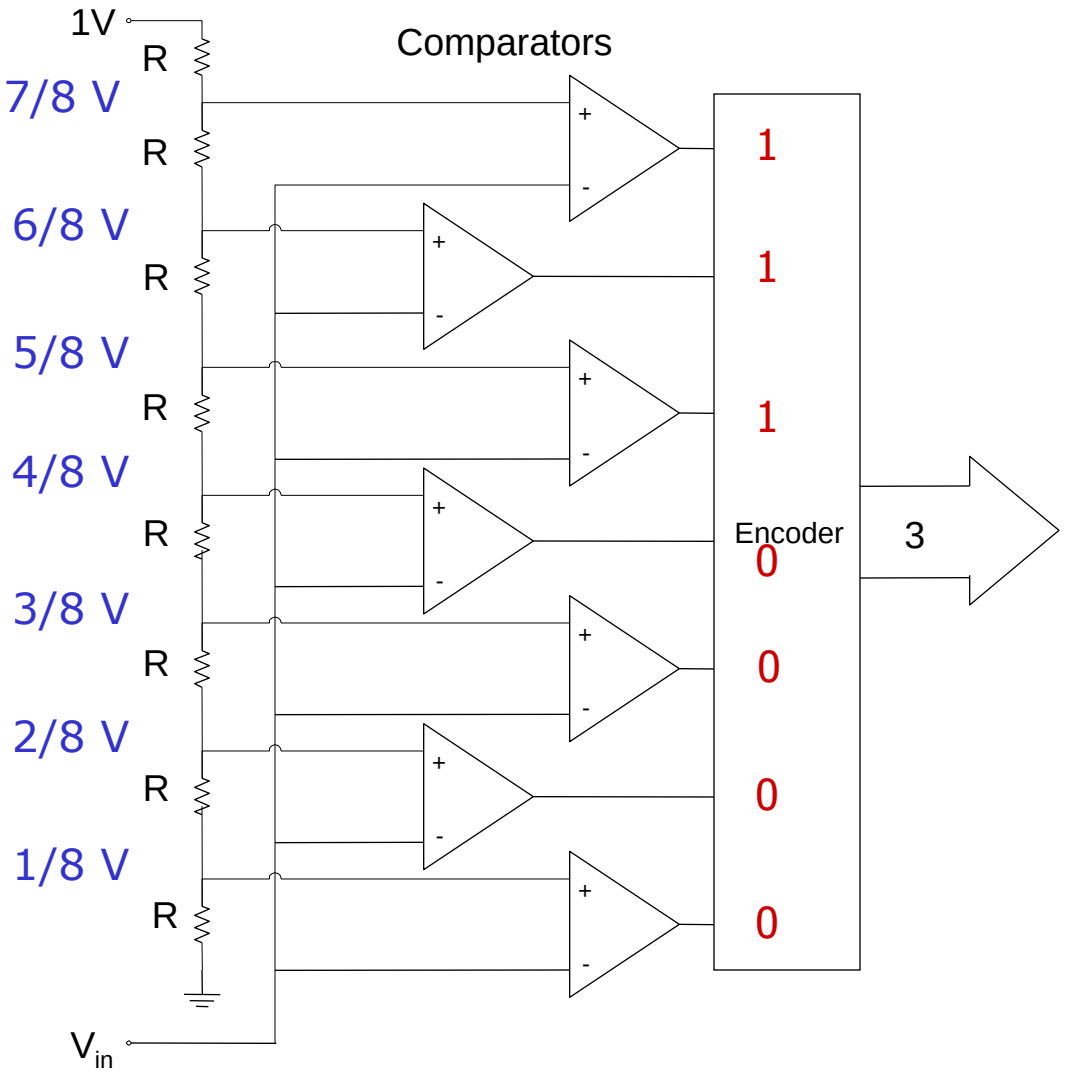
$$n = \left\lfloor \frac{(V_{in})2^N}{V_{+ref}} + 1/2 \right\rfloor$$

$$n = \left\lfloor \frac{3.30v\ 2^{10}}{5v} + 1/2 \right\rfloor = 676$$

$\lfloor X \rfloor = I$ ***floor function: nearest integer I such that I <= X***

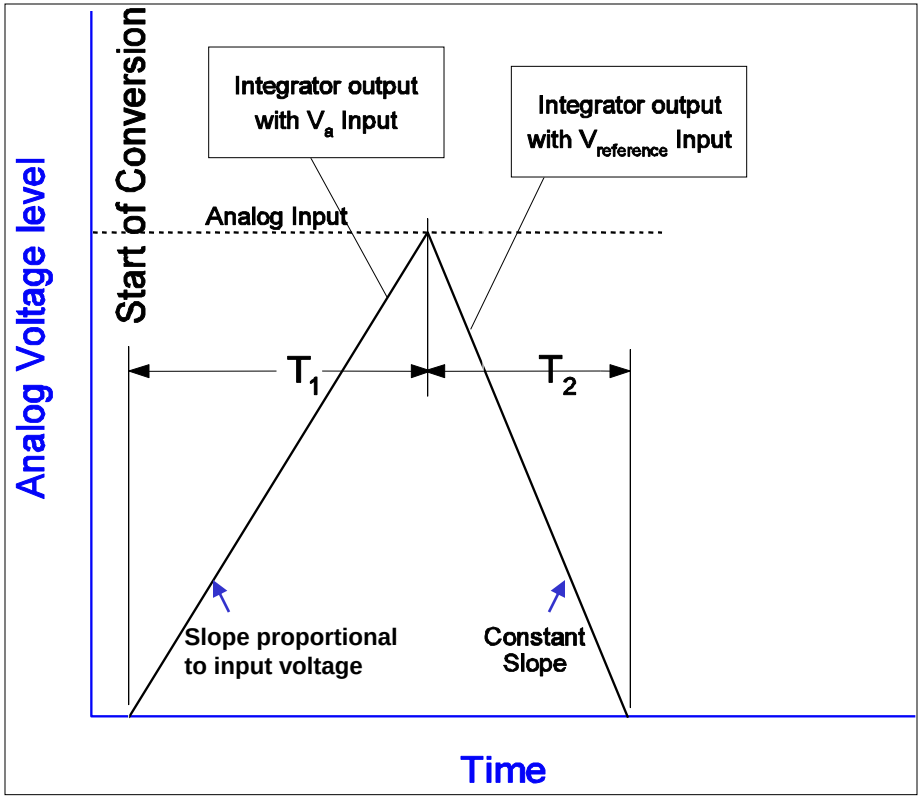
A/D – Flash Conversion

- A multi-level voltage divider is used to set voltage levels over the complete range of conversion.
- A comparator is used at each level to determine whether the voltage is lower or higher than the level.
- The series of comparator outputs are encoded to a binary number in digital logic (a priority encoder)
- Components used
  - 2^N resistors
  - 2^N-1 comparators
- Note
  - This particular resistor divider generates voltages which are *not* offset by ½ bit, so maximum error is 1 bit
  - We could change this offset voltage by using resistors of values R, 2R, 2R ... 2R, 3R (starting at bottom)



ADC - Dual Slope Integrating

- Operation
  - Input signal is integrated for a fixed time
  - Input is switched to the negative reference and the negative reference is then integrated until the integrator output is zero
  - The time required to integrate the signal back to zero is used to compute the value of the signal
  - Accuracy dependent on V_{ref} and timing
- Characteristics
  - Noise tolerant (Integrates variations in the input signal during the T_1 phase)
  - Typically slow conversion rates (Hz to few kHz)

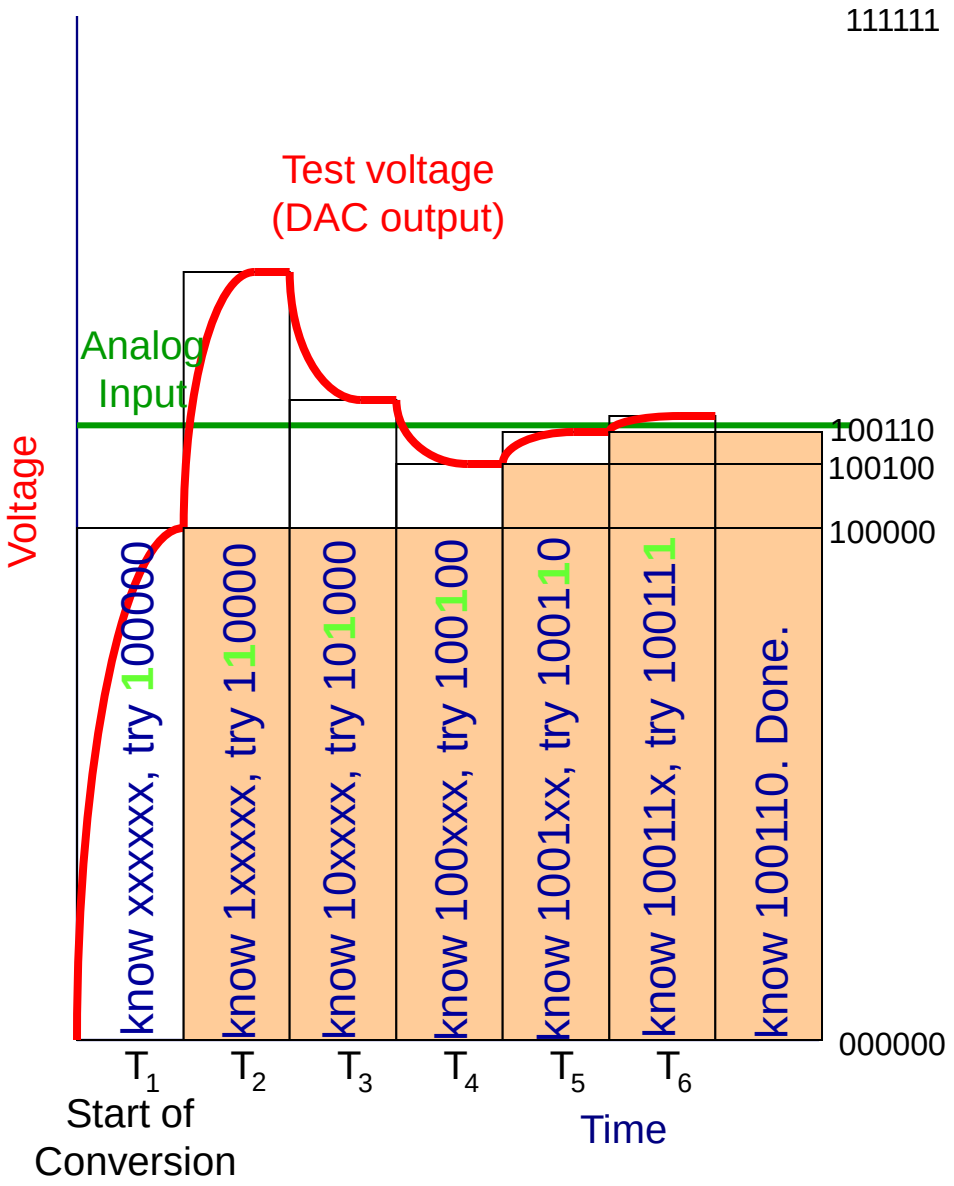


$$\frac{1}{C}\int_0^{T_1} V_{in}\,dt = -\frac{1}{C}\int_0^{T_2} V_{ref}\,dt$$

$$V_{in} = V_{ref}\,\frac{T_2}{T_1}$$

ADC - Successive Approximation Conversion

- Successively approximate input voltage by using a binary search and a DAC
- SA Register holds current approximation of result
- Set all DAC input bits to 0
- Start with DAC's most significant bit
- Repeat
  - Set next input bit for DAC to 1
  - Wait for DAC and comparator to stabilize
  - If the DAC output (test voltage) is **smaller** than the input then set the current bit to 1, else clear the current bit to 0



ADC Performance Metrics

- Linearity measures how well the transition voltages lie on a straight line.
- Differential linearity measure the equality of the step size.
- Conversion time:between start of conversion and generation of result
- Conversion *rate* = inverse of conversion *time*

Sampling Problems

- Nyquist criterion
  - $F_{sample} >= 2 * F_{max\ frequency\ component}$
  - Frequency components above ½ F_{sample} are aliased, distort measured signal
- Nyquist and the real world
  - This theorem assumes we have a perfect filter with "brick wall" roll-off
  - Real world filters have more gentle roll-off
  - Inexpensive filters are even worse (e.g. first order filter is 20 dB/decade, aka 6 dB/octave)
  - So we have to choose a sampling frequency high enough that our filter attenuates aliasing components adequately

Quantization

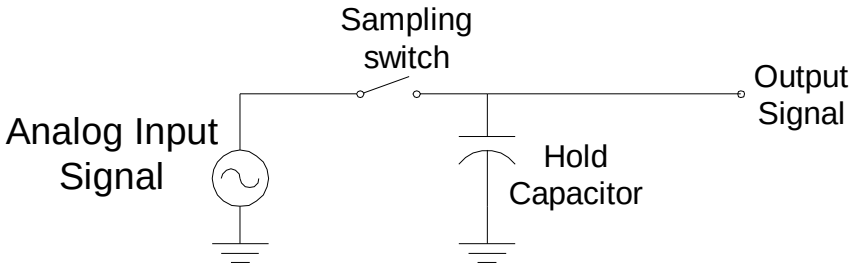
- Quantization: converting an analog value (infinite resolution or range) to a digital value of N bits(finite resolution, 2^N levels can be represented)
- Quantization error
  - Due to limited resolution of digital representation
  - <= $1/(2*2^N)$
  - Acoustic impact can be minimized by dithering (adding noise to input signal)
- 16 bits…. too much for a generic microcontroller application?
  - Consider a 0-5V analog signal to be quantized
  - The LSB represents a change of 76 microvolts
  - Unless you're very careful with your circuit design, you can expect noise of of at least tens of millivolts to be added in
  - 10 mV noise = 131 quantization levels. So $\log_2 131 = 7.03$ bits of 16 are useless!

Inputs

- Multiplexing
  - Typically share a single ADC among multiple inputs
  - Need to select an input, allow time to settle before sampling
- Signal Conditioning
  - Amplify and filter input signal
  - Protect against out-of-range inputs with clamping diodes

Sample and Hold Devices

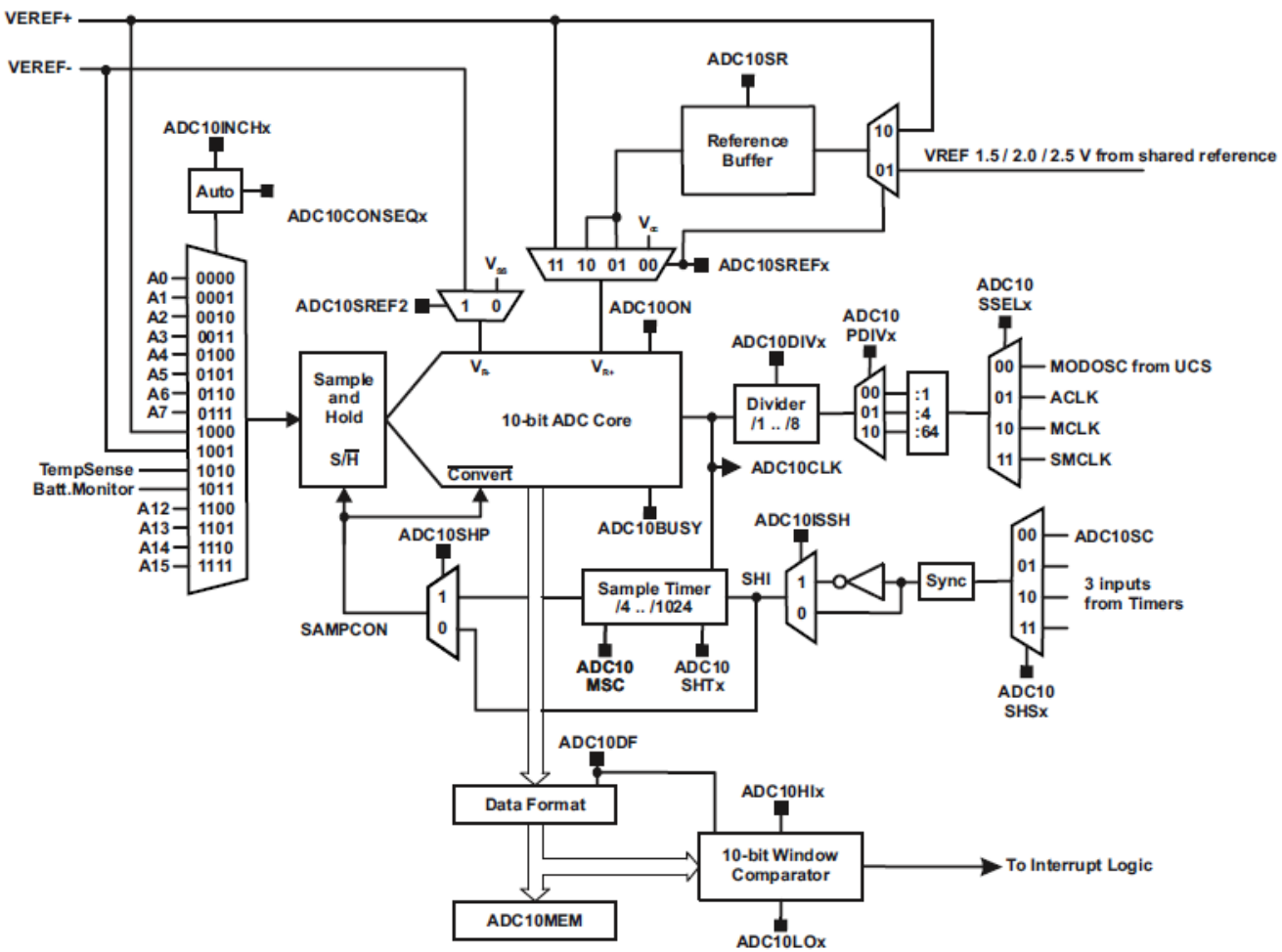
- Some A/D converters require the input analog signal to be held constant during conversion, (eg. successive approximation devices)
- In other cases, peak capture or sampling at a specific point in time necessitates a sampling device.
- This function is accomplished by a sample and hold device as shown to the right:
- These devices are incorporated into some A/D converters



MSP430 ADC Peripheral

- For details see **MSP430FR57xx Family User's Guide**
  - ADC10_B module Chapter 16
- 10 bit SAR converter
- 200KSPS
- 16 channels (12 external / 4 internal**)**
- Input voltage: 0 to V_{CC}
- Software-selectable on chip reference using the REF module or external reference V_{REF} pin
- Input Multiplexer: 16 input channels

MSP430 ADC10_B Block Diagram



A The MODOSC is part of the CS. See the CS chapter for more information.

B When using ADC10SHP = 0, no synchronisation of the trigger input is done.

ADC10_B Block Diagram

MSP430 ADC 10-Bit ADC Core

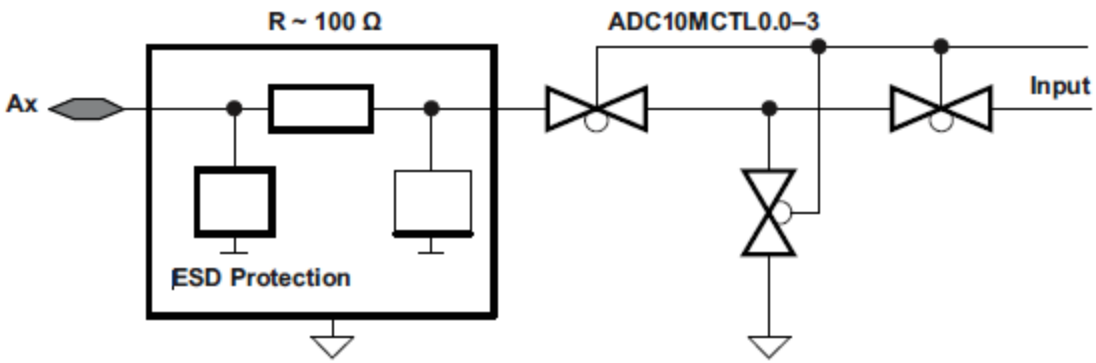
- The ADC core converts an analog input to its 10-bit digital representation and stores the result in the conversion register ADC10MEM0.
- The core uses two programmable/selectable voltage levels (V_{R+} and V_{R-}) to define the upper and lower limits of the conversion. The digital output (N_{ADC}) is full scale (0x03FF) when the input signal is equal to or higher than V_{R+}, and zero when the input signal is equal to or lower than V_{R-}.
- The input channel and the reference voltage levels (V_{R+} and V_{R-}) are defined in the conversion-control memory. The conversion formula for the ADC result N_{ADC} is:

$$N_{ADC} = \left\lfloor \frac{V_{in} - V_{-ref}}{V_{+ref} - V_{-ref}} 2^N + 1/2 \right\rfloor$$

- The ADC10_B core is configured by the control registers ADC10CTL0, ADC10CTL1 and ADC10CTL2.
- The core is enabled with the ADC10ON bit. The ADC10_B can be turned off when not in use to save power. With few exceptions, the ADC10_B control bits can only be modified when ADC10ENC = 0.
- ADC10ENC must be set to 1 before any conversion can take place.

MSP430 ADC10_B Analog Multiplexer

The 12 external and 4 internal analog signals are selected as the channel for conversion by the analog input multiplexer. The input multiplexer is a break-before-make type to reduce input-to-input noise injection resulting from channel switching. The input multiplexer is also a T-switch to minimize the coupling between channels. Channels that are not selected are isolated from the A/D and the intermediate node is connected to analog ground (AVSS), so that the stray capacitance is grounded to eliminate crosstalk.



MSP430 ADC Port Selection

- The ADC10_B inputs are multiplexed with digital port pins. When analog signals are applied to digital gates, parasitic current can flow from VCC to GND. This parasitic current occurs if the input voltage is near the transition level of the gate. Disabling the digital part of the port pin eliminates the parasitic current flow and, therefore, reduces overall current consumption. The PySELx bits provide the ability to disable the port pin input and output buffers.

```
//-------------------------------------------------------------------------------
//Configure Port 1
// Port 1 Pins
// V_DETECT_R           (0x01) // Port Pin 0
// V_DETECT_L           (0x02) // Port Pin 1
// IR_LED               (0x04) // Port Pin 2
// V_THUMB              (0x08) // Port Pin 3
void Init_Port1(void){
  P1SEL0 = 0x00;                // P1 set as I/0
  P1SEL1 = 0x00;                // P1 set as I/0
  P1SEL0 |= V_DETECT_R;         // V_DETECT_R selected
  P1SEL1 |= V_DETECT_R;         // V_DETECT_R selected
  P1SEL0 |= V_DETECT_L;         // V_DETECT_L selected
  P1SEL1 |= V_DETECT_L;         // V_DETECT_L selected
  P1SEL0 |= V_THUMB;            // V_THUMB selected
  P1SEL1 |= V_THUMB;            // V_THUMB selected
  P1DIR = 0x00;                 // Set P1 direction to input
}
//-------------------------------------------------------------------------------
```

MSP430 Voltage Reference Generator

- The ADC10_B module is designed to be used either with the on chip reference supplied by the REF module or an externally reference voltage supplied on external pins.
- The on chip reference is capable of supplying 1.5 V, 2.0 V and 2.5 V. The internal VCC can also be used as the voltage reference.
- External references may be supplied for V_{R+} and V_{R-} through pins V_{REF+} / Ve_{REF+} and V_{REF-} / Ve_{REF-} , respectively.
- The on chip reference is designed for low-power applications. The current consumption of each is specified separately in the device-specific data sheet. The ADC10_B also contains an internal buffer for reference voltages. This buffer is automatically enabled, when the internal reference gets selected for V_{REF+} but it is also optionally available for Ve_{REF+}. The on chip reference from the REF-module needs to be enabled by software. Its settling time is $\leq$30 μs.
- The reference buffer of the ADC10_B also has selectable speed versus power settings. When the maximum conversion rate is below 50ksps, setting ADC10SR = 1 reduces the current consumption of the buffer approximately 50%.

MSP430 Voltage Reference Generator

- The ADC10_B module is designed to be used either with the on chip reference supplied by the REF module or an externally reference voltage supplied on external pins.
- The on chip reference is capable of supplying 1.5 V, 2.0 V and 2.5 V. The internal VCC can also be used as the voltage reference.
- External references may be supplied for V_{R+} and V_{R-} through pins V_{REF+} / Ve_{REF+} and V_{REF-} / Ve_{REF-} , respectively.
- The on chip reference is designed for low-power applications. The current consumption of each is specified separately in the device-specific data sheet. The ADC10_B also contains an internal buffer for reference voltages. This buffer is automatically enabled, when the internal reference gets selected for V_{REF+} but it is also optionally available for Ve_{REF+}. The on chip reference from the REF-module needs to be enabled by software. Its settling time is $\leq$30 μs.
- The reference buffer of the ADC10_B also has selectable speed versus power settings. When the maximum conversion rate is below 50ksps, setting ADC10SR = 1 reduces the current consumption of the buffer approximately 50%.

MSP430 Voltage Reference Generator

- The ADC10_B module is designed to be used either with the on chip reference supplied by the REF module or an externally reference voltage supplied on external pins.
- The on chip reference is capable of supplying 1.5 V, 2.0 V and 2.5 V. The internal VCC can also be used as the voltage reference.
- External references may be supplied for V_{R+} and V_{R-} through pins V_{REF+} / Ve_{REF+} and V_{REF-} / Ve_{REF-} , respectively.
- The on chip reference is designed for low-power applications. The current consumption of each is specified separately in the device-specific data sheet. The ADC10_B also contains an internal buffer for reference voltages. This buffer is automatically enabled, when the internal reference gets selected for V_{REF+} but it is also optionally available for Ve_{REF+}. The on chip reference from the REF-module needs to be enabled by software. Its settling time is $\leq$30 μs.
- The reference buffer of the ADC10_B also has selectable speed versus power settings. When the maximum conversion rate is below 50ksps, setting ADC10SR = 1 reduces the current consumption of the buffer approximately 50%.

MSP430 Auto Power Down

- The ADC10_B module is designed for low-power applications. When the ADC10_B is not actively converting, the core is automatically disabled and automatically re-enabled when needed. The MODOSC is also automatically enabled when needed and disabled when not needed.

MSP430 ADC10_B Conversion Clock

- The ADC10CLK is used both as the conversion clock and to generate the sampling period when the pulse sampling mode is selected. The ADC10_B source clock is selected using the ADC10SSELx bits.

- Possible ADC10CLK sources are SMCLK, MCLK, ACLK, and the MODOSC. The input clock can be divided from 1–512 using both the ADC10DIVx bits and the ADC10PDIVx bits.

- MODOSC, generated internally in the UCS, is in the 5-MHz range, but varies with individual devices, supply voltage, and temperature.

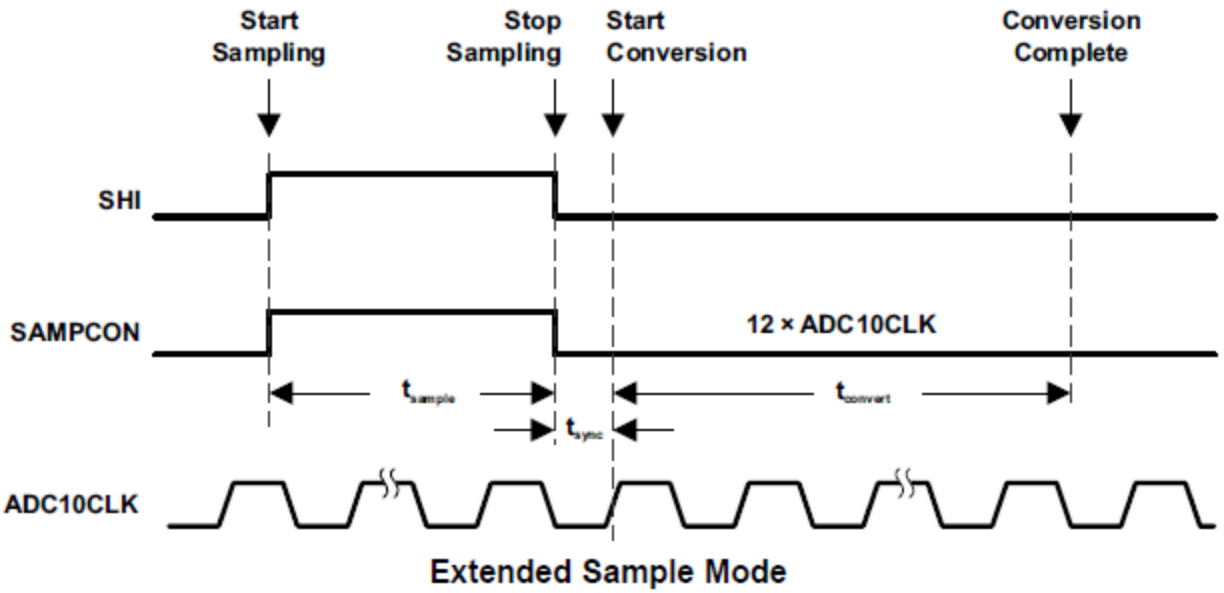
- The user must ensure that the clock chosen for ADC10CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation does not complete and any result is invalid.

MSP430 ADC10_B Sample and Conversion Timing

- An analog-to-digital conversion is initiated with a rising edge of the sample input signal SHI. The source for SHI is selected with the ADC10SHSx bits.
- The polarity of the SHI signal source can be inverted with the ADC10ISSH bit. The SAMPCON signal controls the sample period and start of conversion.
- When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the analog-to-digital conversion, which requires 11 ADC10CLK cycles in 10-bit resolution mode. One additional ADC10CLK is used for the window comparator.
- Two different sample-timing methods are defined by control bit ADC10SHP
  - Extended Sample Mode
  - Pulse mode

MSP430 ADC10_B Extended Sample Mode

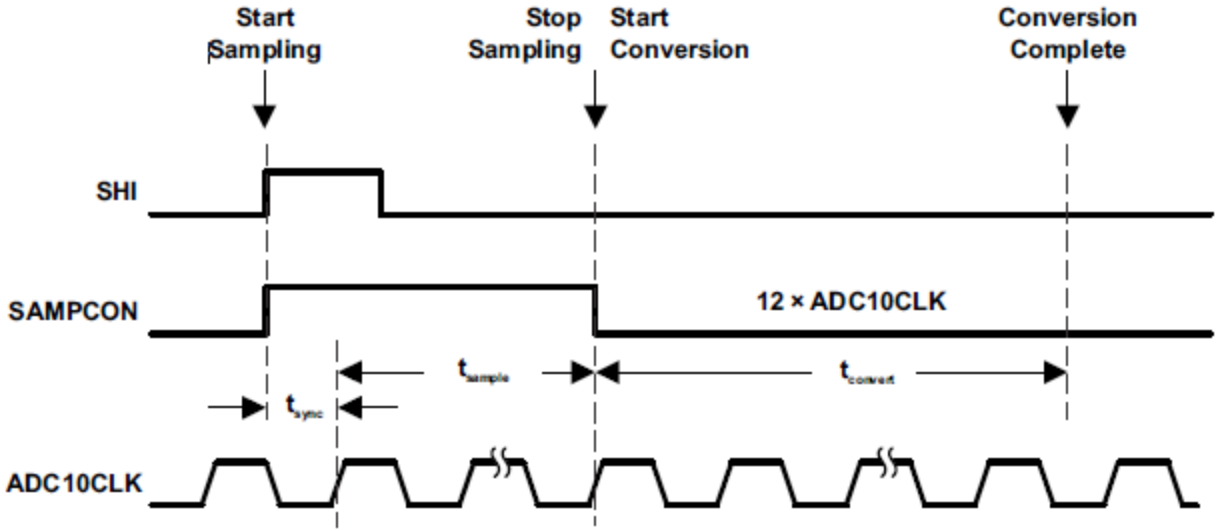
- The Extended Sample Mode is selected when ADC10SHP = 0. The SHI signal directly controls SAMPCON and defines the length of the sample period t_{sample}. When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the conversion after synchronization with ADC10CLK.



Extended Sample Mode

MSP430 ADC10_B Pulse Sample Mode

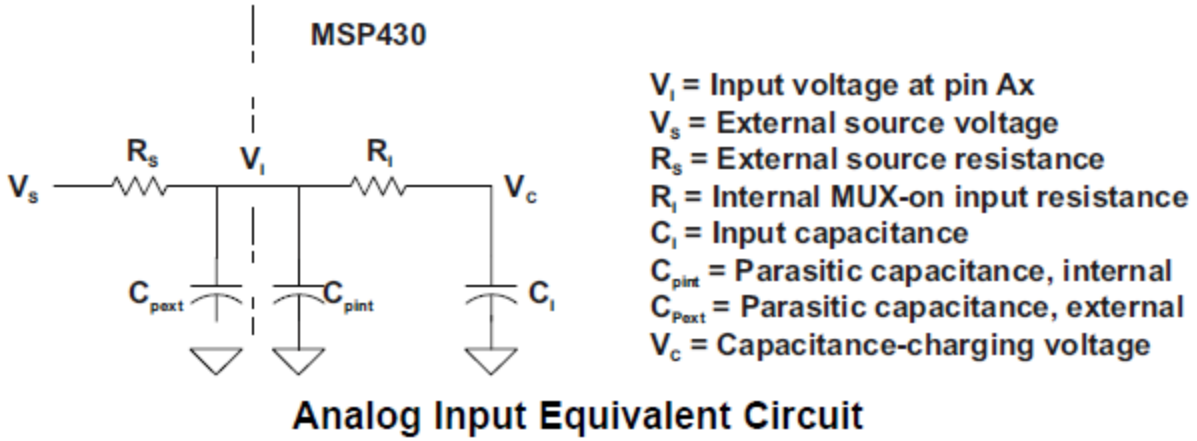
- The Pulse Sample Mode is selected when ADC10SHP = 1. The SHI signal is used to trigger the sampling timer. The ADC10SHTx bits in ADC10CTL0 control the interval of the sampling timer that defines the SAMPCON sample period t_{sample}. The sampling timer keeps SAMPCON high after synchronization with AD10CLK for a programmed interval t_{sample}. The total sampling time is t_{sample} plus t_{sync}.
- The ADC10SHTx bits select the sampling time in 4x multiples of ADC10CLK.



Pulse Sample Mode

MSP430 ADC10_B Sample Timing Considerations

- When SAMPCON = 0, all ADCx inputs are high impedance. When SAMPCON = 1, the selected ADCx input can be modeled as an RC low-pass filter during the sampling time t_{sample}. An internal MUX-on input resistance RI in series with capacitor CI is seen by the source. The capacitor CI voltage VC must be charged to within one-half LSB of the source voltage VS for an accurate 10-bit conversion.



Analog Input Equivalent Circuit

MSP430 ADC10_B Conversion Result / Modes

- The conversion result is accessible using the ADC10MEM0 register independently of the conversion mode selected by the user. When a conversion result is written to ADC10MEM0, the ADC10IFG0 is set.
- The ADC10_B has four operating modes selected by the CONSEQx bits as listed below.

Conversion Mode Summary

ADC10CONSEQx	Mode	Operation
00	Single-channel single-conversion	A single channel is converted once.
01	Sequence-of-channels	A sequence of channels is converted once.
10	Repeat-single-channel	A single channel is converted repeatedly.
11	Repeat-sequence-of-channels	A sequence of channels is converted repeatedly.

MSP430 ADC10_B ADC10IV, Interrupt Vector

- All ADC10_B Interrupt sources are prioritized and combined to source a single Interrupt vector.
- The highest-priority enabled ADC10_B Interrupt generates a number in the ADC10IV register. This number can be evaluated to enter the appropriate software routine. Disabled ADC10_B Interrupts do not affect the ADC10IV value.
- Read access of the ADC10IV register automatically resets the highest-pending Interrupt condition and flag.
- Only the ADC10IFG0 is not reset by this ADC10IV read access. ADC10IFG0 is automatically reset by reading the ADC10MEM0 register or may be reset with software.
- Write access of the ADC10IV register clears all pending interrupt conditions and flags.
- If another Interrupt is pending after servicing of an Interrupt, another Interrupt is generated.
  - For example, if the ADC10OV, ADC10HIIFG and ADC10IFG0 Interrupts are pending when the Interrupt service routine accesses the ADC10IV register, the highest priority interrupt (ADC10OV Interrupt condition) is reset automatically.
- After the RETI instruction of the Interrupt service routine is executed, the ADC10HIIFG generates another Interrupt.

Checklist for Using ADC

- Configure
  - I/O pin
    - Configure for ADC
  - ADC
    - ADC clock speed
    - resolution
    - mode
    - sample and hold
    - trigger mode
    - connect reference voltage
  - Interrupt
    - Create ISR
    - Write access of the ADC10IV register clears all pending interrupts
    - enable interrupt
- Use
  - Start conversion
  - Wait until done
    - polling
    - interrupt
  - Get data and use
    - read from correct ADC10MEMx result register

Demonstrations

- Single Sequence conversion mode
  - *Repeatedly convert input voltage and display value*
  - Configure ADC and interrupt
  - Main loop
    - Start conversion of channel 0
    - Store value during Interrupt in ADC_Thumb
    - Every ¼ second Convert last result to text and display on LCD

Single Sequence – Demonstration Code


```
//-------------------------------------------------------------------
// Configure ADC10_B
void Init_ADC(void){
  ADC10CTL0 = RESET_STATE;          // Clear ADC10CTL0
  ADC10CTL0 |= ADC10SHT_2;          // 16 ADC clocks
  ADC10CTL0 &= ~ADC10MSC;           // Single Sequence
  ADC10CTL0 |= ADC10ON;             // ADC ON - Core Enabled

  ADC10CTL1 = RESET_STATE;          // Clear ADC10CTL1
  ADC10CTL1 |= ADC10SHS_0;          // ADC10SC bit
  ADC10CTL1 |= ADC10SHP;            // SAMPCON signal sourced from sampling timer
  ADC10CTL1 &= ~ADC10ISSH;          // The sample-input signal is not inverted.
  ADC10CTL1 |= ADC10DIV_0;          // ADC10_B clock divider - Divide by 1.
  ADC10CTL1 |= ADC10SSEL_0;         // MODCLK
  ADC10CTL1 |= ADC10CONSEQ_0;       // Single-channel, single-conversion

  ADC10CTL2 = RESET_STATE;          // Clear ADC10CTL2
  ADC10CTL2 |= ADC10DIV_0;          // Pre-divide by 1
  ADC10CTL2 |= ADC10RES;            // 10-bit resolution
  ADC10CTL2 &= ~ADC10DF;            // Binary unsigned
  ADC10CTL2 &= ~ADC10SR;            // supports up to approximately 200 ksps

  ADC10MCTL0 = RESET_STATE;         // Clear ADC10MCTL0
  ADC10MCTL0 |= ADC10SREF_0;        // V(R+) = AVCC and V(R-) = AVSS
  ADC10MCTL0 |= ADC10INCH_3;        // Channel A3 Thumb Wheel
  ADC10IE |= ADC10IE0;              // Enable ADC conversion complete interrupt
}
//-------------------------------------------------------------------
```

Single Sequence – Demonstration Code

```
//-------------------------------------------------------------------------
// ADC10 interrupt service routine
// ADC_Right_Detector;                  // A00 ADC10INCH_0 - P1.0
// ADC_Left_Detector;                   // A01 ADC10INCH_1 - P1.1
// ADC_Thumb;                           // A03 ADC10INCH_3 - P1.3
// ADC_Temp;                            // A10 ADC10INCH_10 – Temperature REF module
// ADC_Bat;                             // A11 ADC10INCH_11 - Internal
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void){
  switch(__even_in_range(ADC10IV,12))  {
    case  0: break;                     // No interrupt
    case  2: break;                     // conversion result overflow
    case  4: break;                     // conversion time overflow
    case  6: break;                     // ADC10HI
    case  8: break;                     // ADC10LO
    case 10: break;                     // ADC10IN
    case 12:
        ADC_Thumb = ADC10MEM0;          // Channel A3
      break;
    default: break;
  }
}
//-------------------------------------------------------------------------
```

Single Sequence – Demonstration Code

- Call from within Main()

```
//-------------------------------------------------------------------
void ADC_Process(void){
  while (ADC10CTL1 & BUSY);         // Wait if ADC10 core is active
  ADC10CTL0 |= ADC10ENC + ADC10SC; // Sampling and conversion start
}
//-------------------------------------------------------------------
```

Using ADC Values

- The ADC gives an integer representing the input voltage relative to the reference voltages
- Several conversions may be needed
  - For many applications you will need to compute the approximate input voltage
    - V_{in} = …
  - For some sensor-based applications you will need to compute the physical parameter value based on that voltage (e.g. pressure) – this depends on the sensor's transfer function
  - You will likely need to do additional computations based on this physical parameter (e.g. compute depth based on pressure)
- Data type
  - It's likely that doing these conversions with integer math may lead to excessive loss of precision, so using floating point math may be required
    - This requires using the floating point library (math.h).
  - AFTER you have the application working, you can think about accelerating the program using fixed-point math (scaled integers).
- To output ASCII characters (to the LCD, for example). You will need to convert the Hexadecimal Integer number to ASCII.

Digital to Analog Conversion

- May need to generate an analog voltage or current as an output signal
  - Audio, motor speed control, LED brightness, etc.
- Digital to Analog Converter equation
  - n = input code
  - N = number of bits of resolution of converter
  - V_{ref} = reference voltage
  - V_{out} = output voltage
  - $V_{out} = V_{ref} * n/(2^N)$