CPE 323
MODULE 11
Analog-to-Digital Conversion

Aleksandar Milenković

Email: milenka@uah.edu

Web: http://www.ece.uah.edu/~milenka

Overview

This module introduces main concepts of analog-to-digital (AD) and digital-to-analog (DA) conversion. You will learn hardware aspects as well as software aspects of the analog-to-digital and digital-to-analog conversion and AD and DA converters. You will understand how to configure and utilize MSP430 ADC12 and DAC12 peripherals in your programs.

Objectives

- *Learners will understand hardware and software aspects of analog-to-digital and digital-to-analog converters*
- *Learners will understand how to configure and interact with MSP430 ADC peripheral*
- *Learners will understand how to configure and interact with MSP430 DAC peripheral*

Contents

LaCASA

1 AD Conversion: An Introduction

Embedded computer systems typically a part of other systems or devices. We often refer to four main tasks of any embedded computer system (or any computer system in general) as: (1) sensing the external physical world through sensors; (2) processing information; (3) storing information; and (4) communicating information and acting on the environment. Sensors or transducers are used to convert physical quantities (e.g., force, atmospheric pressure, sound, light, temperature, and others) into electrical signals (e.g., voltage or current) that we can measure. The electrical signals are often either noisy, weak, or both noisy and weak, so signal conditioning circuits are responsible to remove undesired harmonics of continual electrical signals (filtering) and amplify them (amplification) so they can be properly measured. Once the electrical signals are ready and in a desired range, a critical step is to convert them into corresponding digital values that can be further processed, stored, and/or communicated using digital computers. The process of converting analog electrical signals into binary numbers that correspond to the magnitude of the input signals is known as analog-to-digital conversion and is carried out using dedicated peripherals called analog-to-digital converters (ADCs).

One outcome of data processing is that we may need to act on the environment. For example, think about the air conditioning system at your house – a sensor continually measures the temperature (sensing), and if the temperature rises above a certain threshold (processing), a controller sends a signal to your AC unit to start pumping in cool air (acting on the environment). Once the temperature is lowered, the controller sends a signal to the AC unit to stop its operation. Acting on the environment sometimes requires that we generate analog electrical signals of certain amplitude and frequency. To generate such signals we conduct digital-to-analog conversion and for that we rely on peripheral devices called digital-to-analog converters (DACs).

In this module you will learn hardware aspects as well as software aspects of the analog-to-digital and digital-to-analog converters. You will understand how to configure and utilize MSP430 ADC and DAC peripherals in your programs. The very name MSP that stands for *Mixed Signal Processor* underscores the fact that MSP430 family of microcontrollers integrates a processor, non-volatile and volatile memories, and peripherals that deal with both analog and digital electrical signals.

Things to remember 1-1. Analog-to-digital conversion.

Analog-to-digital conversion is a process of converting analog continuous input signals (typically voltage or current) into discrete digital numbers that represent the magnitude of the input signal.

Things to remember 1-2. Digital-to-analog conversion.

Digital-to-analog conversion is a process of converting digital numbers into a continuous analog output signal.

2 AD Conversion: An Example

Let us consider an input analog signal, a_0, as shown in Figure 1. The ramp like signal is periodic with the period of $T_{a0} = 0.5\ ms$. The frequency of the input signal is $F_{a0} = 1/T_{a0} = 2{,}000\ Hz$. The signal rises from 0 to 2.5 V in 0.3 ms and falls back to 0 V in 0.2 ms. Please note that the signal is bounded between 0 and 2.5 V. Analog-to-digital conversion assumes that we want to convert this continual analog input signal into a sequence of binary numbers, where each binary number corresponds to the magnitude of the input signal at a given moment. Two important questions related to the process of analog-to-digital conversion are as follows: (a) how many bits do we want in the binary representation (resolution of AD conversion); and (b) how many discrete samples do we want to get per each signal period (sampling frequency). By increasing the number of bits, we increase the resolution and accuracy of the AD conversion. By increasing the sampling frequency, we can recreate input signal more faithfully.

Let us assume the following parameters of an AD conversion: the bit length of the binary representation is 8 bits and the number of samples per single period of the input signal is 5. This is equivalent to having 10,000 samples per second (2,000 Hz * 5 samples/period = 10,000 samples per second, or $F_s = 10{,}000$ sps). The sampling period is the the time distance between two consecutive samples and in this example is: $\Delta t_s = 1/F_s = 0.1\ ms$.

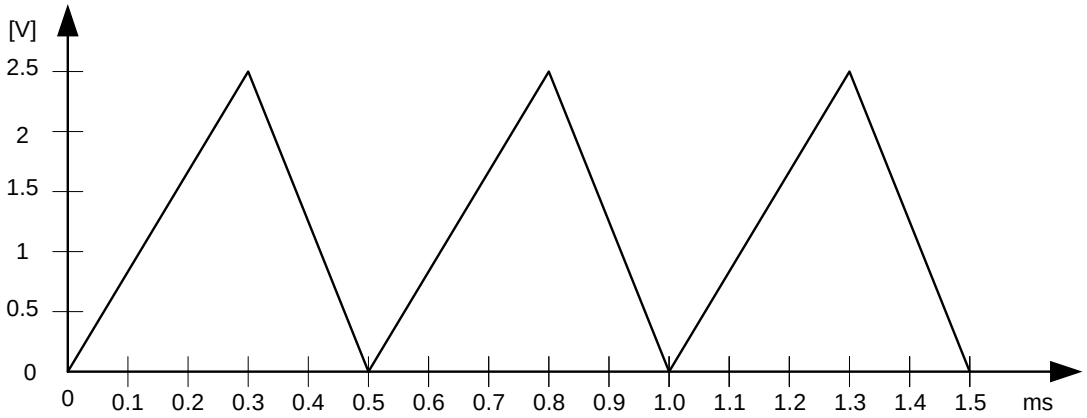


Figure 1. An example input analog signal.

Figure 2 shows a block-box representation of analog-to-digital conversion. An analog input voltage signal a_0 is periodically sampled by an AD converter. The sampling is controlled by a control signal named *Sample/Hold*. The reference voltages of the AD converter ideally match the input signal bounds, i.e., $V_{R-} \leq a_0 \leq V_{R+}$. The AD converter produces an *N*-bit binary representation that corresponds to the magnitude of the input signal at the moment it has been sampled. This binary output is then read by the processor core using one of the I/O interfacing approaches (polling, ISRs, DMA transfer).

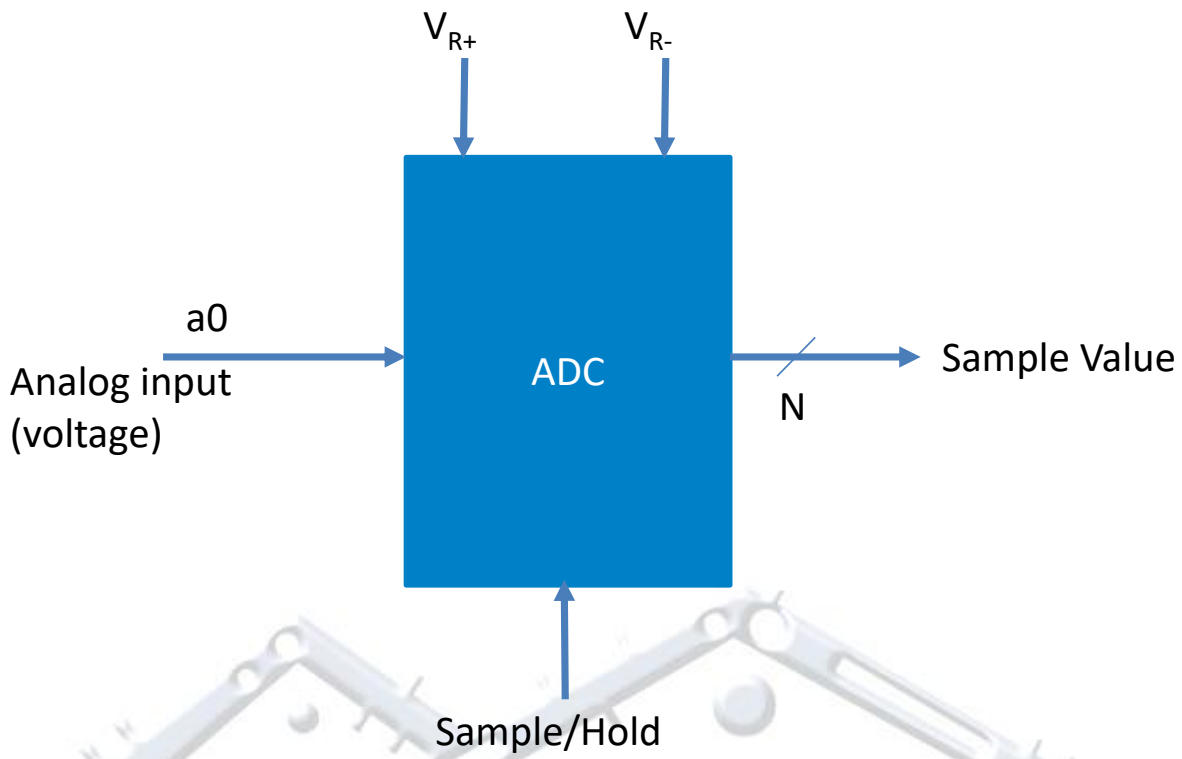


Figure 2. Analog-to-digital conversion: a top view.

Without lack of generality, we assume that samples are taken at times: $t_i = i \cdot \Delta t_s$, for *i*=0, 1, 2, 3, and so on. In a general case the function of an AD converter can be described by the following transfer function:

$$\text{Sample Value} = nint\left((2^N - 1) \cdot \frac{a_0 - V_{R-}}{V_{R+} - V_{R-}}\right)$$

where N is the resolution of the AD converter (the number bits in the output binary representation), a_0 is the value of the analog input at a given time, and V_{R+} and V_{R-} are the reference voltages of the AD converter. The $nint$ stands for the nearest integer. In our example we assume the following parameters: $N = 8, V_{R+} = 2.5\ V$, and $V_{R-} = 0\ V$. The transfer function is thus as follows:

$$\text{Sample Value} = nint\left(255 \cdot \frac{a_0}{2.5\ V}\right)$$

Consequently, if $a_0 = 0\ V, Sample\ Value = 0\ or\ 0x00$; if $a_0 = 2.5\ V$, $Sample\ Value = 255\ or\ 0xFF$.

Example 2-1. Find binary representation of samples in one period for signal a_0 assuming $N = 8$, $V_{R+} = 2.5\,V$, and $V_{R-} = 0V$. The initial sample is taken at $t_0 = 0$ s.

Sample 0 at time $t_0 = 0\,ms$: $a_0 = 0\,V \Rightarrow sample\ value = nint\left(255 \cdot \frac{0}{2.5\,V}\right) = 0\ (0x00)$

Sample 1 at time $t_1 = \Delta t_s = 0.1\,ms$: $a_0 = \frac{2.5}{3} = 0.8333 \Rightarrow$

$sample\ value = nint\left(255 \cdot \frac{0.8333\,V}{2.5\,V}\right) = 85\ (0x55)$

Sample 2 at time $t_2 = 2 \cdot \Delta t_s = 0.2\,ms$: $a_0 = \frac{2 \cdot 2.5}{3} = 1.6666 \Rightarrow$

$sample\ value = nint\left(255 \cdot \frac{1.666\,V}{2.5\,V}\right) = 170\ (0xAA)$

Sample 3 at time $t_3 = 3 \cdot \Delta t_s = 0.3\,ms$: $a_0 = \frac{3 \cdot 2.5}{3} = 2.5 \Rightarrow$

$sample\ value = nint\left(255 \cdot \frac{2.5\,V}{2.5\,V}\right) = 255\ (0xFF)$

Sample 4 at time $t_4 = 4 \cdot \Delta t_s = 0.4\,ms$: $a_0 = \frac{1 \cdot 2.5}{2} = 1.25 \Rightarrow$

$sample\ value = nint\left(255 \cdot \frac{1.25\,V}{2.5\,V}\right) = 127\ (0x7F)$.

Note: to find values of the input signal use simple proportions. For the input signal in i*0.1 ms + k*0.5 ms (i=0, 1, 2, 3, k=0, 1, 2, 3, …) the proportion is i*2.5/3. For the input signal at i*0.1 ms + k*0.5 ms (i=4, k=0, 1, 2, …), the proportion is 2.5/2.

Figure 3 illustrates the input signal overlapped with the samples taken at $t_i = i \cdot \Delta t_s$. If we were to create an output analog signal using a DA converter based on samples acquired by the AD conversion, the signal would look like the one shown in red. The ramp shape of the input signal is quite distorted and this is captured by the signal-to-noise ratio metric. One way to improve accuracy of the signal is to take more samples per one signal period. Repeat the exercise from Example 2-1 if $F_s = 20{,}000$ sps. Also, repeat the exercise assuming *N*=16 bits. In general, we are always limited with the number of bits in the binary representation as well as with how many samples we can acquire in any period of time, so our "digitized" interpretation of the real physical world is never perfect.

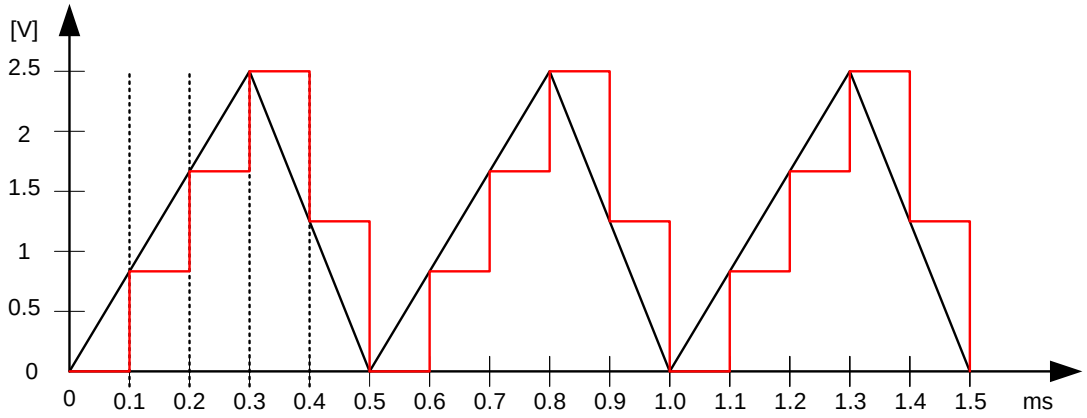


Figure 3. An example input analog signal sampled every 0.1 ms.

3 Analog-to-Digital Conversion Flow

Figure 4 illustrates an analog-to-digital conversion flow. We are embedded into physical world and want to quantify and measure it. Sensors or transducers are used to convert physical quantities into electrical signals such as voltage or current that we can further act on. Examples of sensors are many. Just pick-up your smartphone and try to list all sensors it has. We have a 3D accelerometer that measures force, a 3D gyroscope that measures angular velocity, a 3D magnetic field sensor that measures strength of Earth's magnetic field. These 3 sensors together are used in navigation and orientation and are sometimes referred to as inertial sensors (9 degrees of freedom term comes for these 3 sensors, each having x, y, and z components). Next you have a microphone, a camera or rather multiple cameras, and a proximity sensor. We have seen a tremendous progress in sensor technology with rise of so-called MEMS devices – Micro-Electro-Mechanical Systems. Taking advantages of semiconductor innovations and miniaturization we can now have a single chip that integrates multiple sensors, replacing multiple older mechanical sensors that are both cumbersome and heavy. As an example, next time you visit the U.S. Rocket and Space Museum across the street, please pay attention to the gyroscope used in Saturn-V rocket.

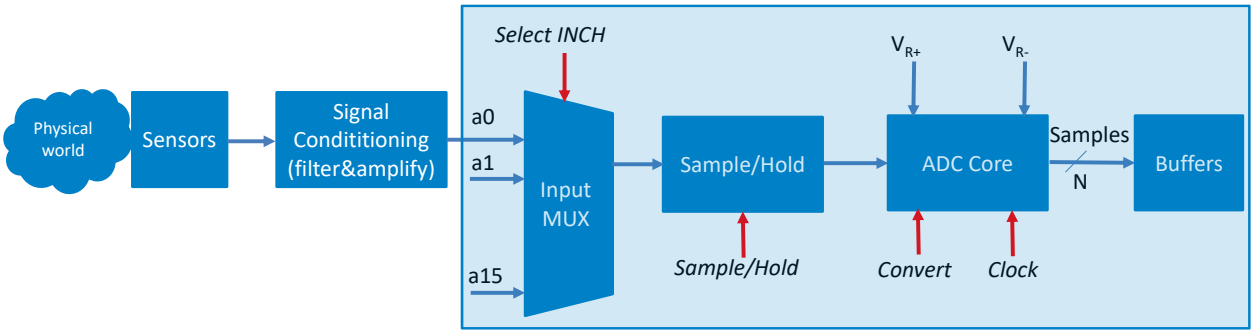


Figure 4. Analog-to-digital conversion flow.

Electrical signals coming from sensors often need to be filtered to remove undesired harmonics and/or to be amplified so we can measure them. For example, if you are interested in getting an ECG (electrocardiogram) signal from humans, its amplitude is ~1 mV. So even if you hook yourself up on an oscilloscope, you are not going to see your ECG signal because it is buried in electrical noise. Before proceeding we have to remove undesired harmonics through filtering and amplify the signal perhaps 1,000 times to bring it to 1 V. For this we can use an external specialized chip designed for ECG filtering and amplification (often referred to as bio amplifiers becasue they can work with other types of physiological electrical signals: EEG – electroencephalogram that represents macroscopic electrical activity of the brain underneath; and EMG – electromyogram that represents skeletal muscle activity).

It is cost-prohibitive and unnecessary to have an AD converter for each analog signal you might be interested in a system. Rather, multiple analog signals can be brought to a single AD converter. An analog multiplexer can select one analog signal at a time using Select INCH (select input channel) control bits. This way, we can handle multiple analog inputs, by converting one input signal at a time.

Once the input channel is selected, the selected analog input signal is brought to the next block called Sample&Hold. The input analog signals are continuous, meaning they are not fixed in time. Converting a signal that constantly changes is not an option. So the purpose of the Sample&Hold block is to quickly capture the true value of the analog input (Sample mode) and then present that signal to the ADC core (Hold mode), where the conversion takes place. A good abstraction for the Sample&Hold module is shown in Figure 5. An analog input goes through a switch, a resistor with resistance R, and a capacitor with capacitance C. During sampling, the switch is closed. The capacitor is charged so that the output voltage from the capacitor (V_{out}) matches as closely as possible the voltage at the input channel (V_{out}). Holding the switch closed longer than needed will capture continual changes of the input signal and will limit our ability to get as many samples as possible. Opening the switch to early, will result in discrepancy between the true value of the input signal and the voltage presented to the ADC core. Once sampling is done, the switch is opened, thus disconnecting the capacitor from the input channel. The further changes at the input channel are thus not going to affect the charge on the capacitor that presents the fixed voltage to the AD core.

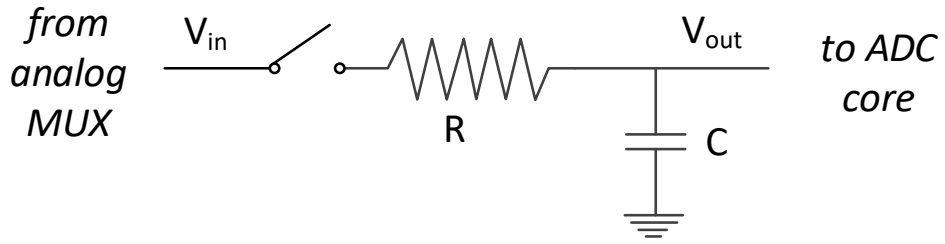


Figure 5. Sample&Hold Module.

Things to remember 3-1. Sample and hold

The sample&hold block is responsible to capture the true value of the input signal (sampling phase) and decouple the analog input from the AD core logic during conversion (holding phase).

The AD core implements the transfer function as described above (and repeated here):

$$\text{Sample Value} = nint\left((2^N - 1) \cdot \frac{a_0 - V_{R-}}{V_{R+} - V_{R-}}\right)$$

Figure 6 illustrates an ideal transfer function for a 4-bit AD converter. The entire input range ($V_{R+} - V_{R-}$) is divided into 16 ranges, 0 to 0.5V_{LSB}, 0.5V_{LSB} to 1.5V_{LSB}, 1.5V_{LSB} to 2.5V_{LSB}, . . . 14.5V_{LSB} to 16V_{LSB} that correspond to binary outputs 0000, 0001, … 1111, respectively.

Finally, once the AD core produces an N-bit sample, it is stored into a buffer register and appropriate flags are set to alert the processor core to read them before the next sample can perhaps overwrite the existing one.

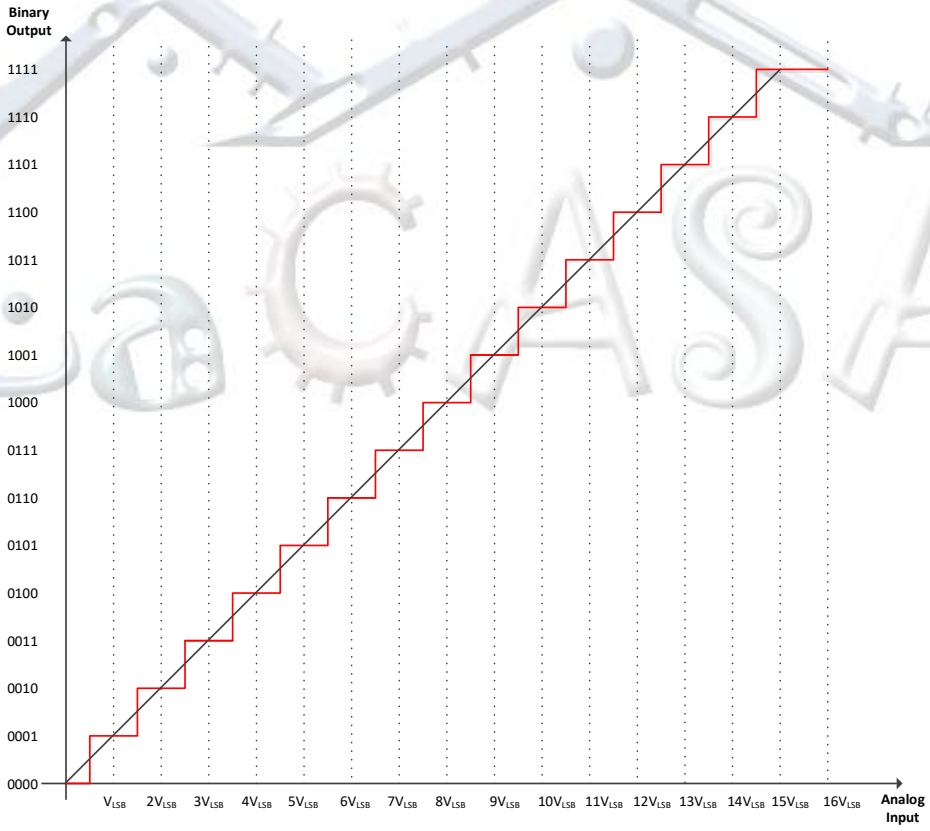


Figure 6. Ideal AD Transfer Function (N=4). The input voltage is divided into 16 (2^4 levels) with 16 digital representations on the output. Please note that the input range 0 to 0.5V_{LSB} corresponds to binary 0000, 0.5V_{LSB} to 1.5V_{LSB} to 0001, and so on. The input range of 14.5V_{LSB} to 16 V_{LSB} corresponds to 1111. Please note asymmetricity of input ranges for 0000 and 1111.

Things to remember 3-2. AD Transfer Function

The transfer function of an AD converter that converts an analog input voltage a_0 into a binary representation is described by the following equation:

$$\text{Sample Value} = nint\left((2^N - 1) \cdot \frac{a_0 - V_{R-}}{V_{R+} - V_{R-}}\right)$$

where *N* is the number of bits in the binary representation, and V_{R+} and V_{R-} are reference voltages ($V_{R-} \leq a_0 \leq V_{R+}$). *nint* is the nearest integer function.

When dealing with AD converters, you will sometimes find terms such as resolution, accuracy, aperture time, conversion time, and the maximum sampling frequency.

AD Resolution. The resolution of an AD converter defines how finely the value measured can be represented and it is defined as follows:

$$\mathrm{V}_{LSB} = \frac{\mathrm{V}_{R+} - \mathrm{V}_{R-}}{2^N}$$

The quantization error is thus always present and can be quantified as $\pm 0.5\mathrm{V}_{LSB}$.

AD Accuracy. The AD accuracy refers to how much the value under measurement deviates from its true value due to AD converter inaccuracies. The AD converters are not ideal, they can have issues with offset, gain, they may not be perfectly linear or may have some missing codes in the binary output. In addition, there is also some finite amount of noise within the converter.

Aperture Time. The aperture time is the time the Sample&Hold signal is looking at the input signal. This time should be long enough so that the captured voltage at the output of the sample&hold module is as close as possible to the voltage at the input channel. This error caused by finite sampling time (aperture time) should be less than $0.5\mathrm{V}_{LSB}$. If not, we would not fully utilize the AD converter's resolution.

Conversion Time. The conversion in the AD core itself takes a finite amount of time. Depending on the implementation of the AD core, this time can be directly proportional to the resolution of the AD converter (e.g., takes N+1 clock cycles), or can be rather fast requiring a single clock cycle. The design space usually includes trade-offs – faster implementations require more sophisticated circuitry and will cost more.

Maximum Sampling Frequency. The maximum number of samples one can get from an AD converter is bounded by the the sum of the aperture time and conversion time. Thus, the maximum sampling frequency is defined as follows:

$$F_{s.max} \leq \frac{1}{Aperture\ Time + Conversion\ Time}$$

Please keep in mind that this is an upper limit on how many samples your AD converter can give you in a unit of time. If multiple analog input channels are converted in a sequence, the maximum number of samples per each channel will be lower. For example, if the maximum number of

samples per second (sps) is 200,000 and you are sweeping across 8 different analog inputs, the maximum sampling frequency on each channel will be 25,000 sps. Often you will see that maximum sampling frequency of MSP430 ADC12 converter is 200,000 sps. Where does this number come from? The ADC12 can work at maximum 5,000,000 Hz clock, its conversion time is 13 clock cycles. Assuming aperture time of 12 clock cycles, the maximum number of samples we can get from the ADC12 is 5,000,000/(12+13) = 200,000 sps.

`Things to remember 3-3. ADC properties`

When dealing with ADCs, we often discuss their properties in terms of ADC resolution (higher is better), ADC accuracy (higher is better), ADC aperture time (sampling time), ADC conversion time, and maximum sampling frequency (higher is better). Make sure you can define these properties.

4 Analog-to-Digital Converter Types

There are a number of analog-to-digital converter implementations differing in their properties, speed, and cost. In this section we will describe two types: succession approximation that is relatively simple to implement but slow and flash A/D that requires a lot of resources but is fast.

4.1 Successive Approximation ADC

The successive approximation is one of the most widely used ADC implementations. Figure 7 shows its block diagram. It consists of a digital-to-analog converter, a comparator, and a successive approximation register. The conversion is performed in multiple steps. In the first step, we determine the most significant bit (MSB, bit position N-1), in the second step we determine the bit at position N-2, and so on. The last step is to determine the LSB. In the first step the binary value 100…0 is written into the successive approximation register (SAR). The DAC produces the voltage that corresponds to $2^{N-1}V_{REF}$, or one half of the total range. The comparator compares the analog input and gives a logic 1 at the output if $v_{in} \geq \frac{1}{2}V_{REF}$ or a logic 0 otherwise. If the comparator output is set to a logic 1 that means that the input signal is in the upper half of the full scale range, and the MSB of the register should be 1. If the comparator output is a logic 0 that means the input voltage is in the lower half of the full scale range, and the MSB of of the SAR should be 0. Thus the MSB bit is determined at the end of the step 1. In the next step, we probe bit at position N-2, and so on until we determine all bits. The total number of step corresponds to the number of bits, making this implementation cost-effective but relatively slow.

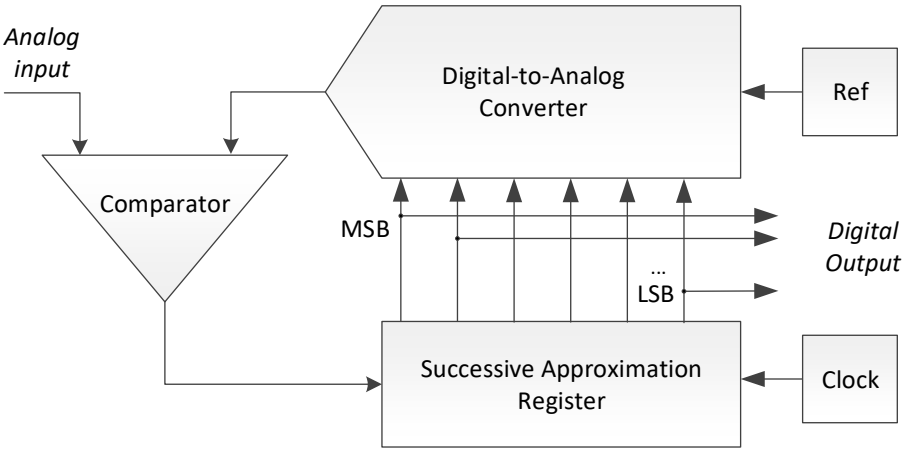


Figure 7. Successive approximation ADC.

Example 4-1. Walk through the SAR AD conversion step-by-step assuming a 4-bit AD converter (N=4), $V_{R+} = 2.5\,V$, and $V_{R-} = 0V$. The input voltage is 1.8 V.

Step 1: SAR=1000b; $V_{DAC} = 8 \cdot V_{LSB} = 8 \cdot \frac{2.5}{16} = 1.25\,V;\ V_{in} \geq V_{DAC} => bit\ 3\ is\ 1$

Step 2: SAR=1100b; $V_{DAC} = 12 \cdot V_{LSB} = 12 \cdot \frac{2.5}{16} = 1.875\,V;\ V_{in} \leq V_{DAC} => bit\ 2\ is\ 0$

Step 3: SAR=1010b; $V_{DAC} = 10 \cdot V_{LSB} = 10 \cdot \frac{2.5}{16} = 1.5625\,V;\ V_{in} \geq V_{DAC} => bit\ 1\ is\ 1$

Step 4: SAR=1011b; $V_{DAC} = 11 \cdot V_{LSB} = 11 \cdot \frac{2.5}{16} = 1.71875\,V;\ V_{in} \geq V_{DAC} => bit\ 0\ is\ 1$

So, the final digital value is 1011b (11). As a way of verification we can use the transfer function of the ideal 4-bit AD converter to confirm correctness of the final binary output:

$$\text{Sample Value} = nint\left(15 \cdot \frac{1.8\,V}{2.5\,V}\right) = nint(10.8) = 11$$

4.2 Parallel or Flash ADC

Figure 8 shows a block diagram of a parallel or flash ADC. It implements the ADC voltage transfer function. The resistors in series divide the full voltage range producing $0.5V_{LSB}$, $1.5V_{LSB}$, $2.5V_{LSB}$, . . . to $(2^N\text{-}1.5)V_{LSB}$ threshold voltages (the resistors in series create a voltage divider). These threshold voltages are brought to 2^N-1 analog comparators. They are compared in parallel to the analog input. The outputs of comparators are brought into a 2^N-to-N bit encoder that gives the digital output.

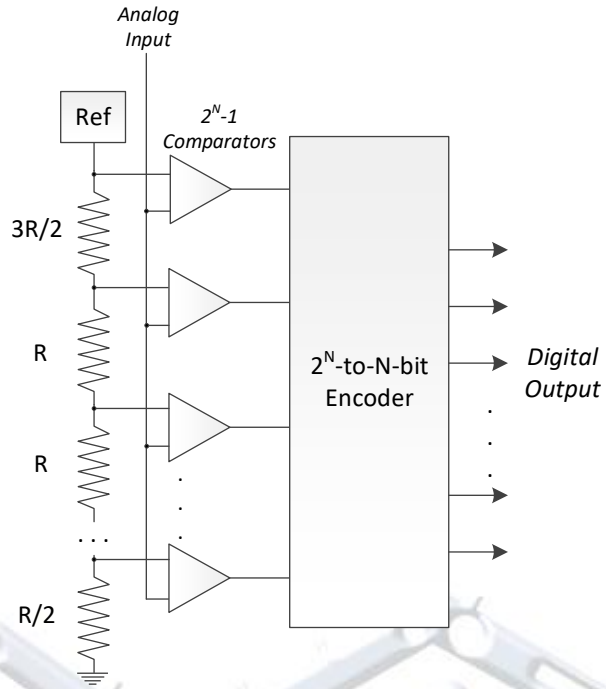


Figure 8. Parallel or flash ADC.

Example 4-2. Illustrate inner workings of the flash ADC. Assume a 4-bit flash ADC (N=4), $V_{R+} = 2.5\,V$, and $V_{R-} = 0V$. The input voltage is 1.8 V.

There are 15 comparators comparing the input voltage to the threshold voltages as follows: $0.5V_{LSB}$, $1.5V_{LSB}$, $2.5V_{LSB}$, . . . to $14.5V_{LSB}$. The output of bottom 11 comparators is set to logic 1 because input voltage is larger than $0.5V_{LSB}$, $1.5V_{LSB}$, $2.5V_{LSB}$, . . . to $10.5V_{LSB}$, respectively. The output of top 4 comparators is at logic 0 because the input is less than $11.5V_{LSB}$, $12.5V_{LSB}$, $13.5V_{LSB}$, and $14.5V_{LSB}$, respectively. The outputs from comparators from 0 to 15 are thus: 1111_1111_1110_000b. The output of the encoder will give you 1011b (11).

5 MSP430's ADC12_A Controller

Figure 9 shows a block diagram of an MSP430F5529 device. It includes an ADC12_A peripheral – a 12-bit ADC that can sample up to 200 Ksps (kilo samples per second) and has 16 input channels. It also includes a comparator COMP_B and REF module that generates reference voltages for ADCs and DACs. Figure 10 shows a block diagram of MSP430F4618. It includes an 12-channel ADC12, a two-channel DAC12, an analog comparator Comparator_A, and three op amps.

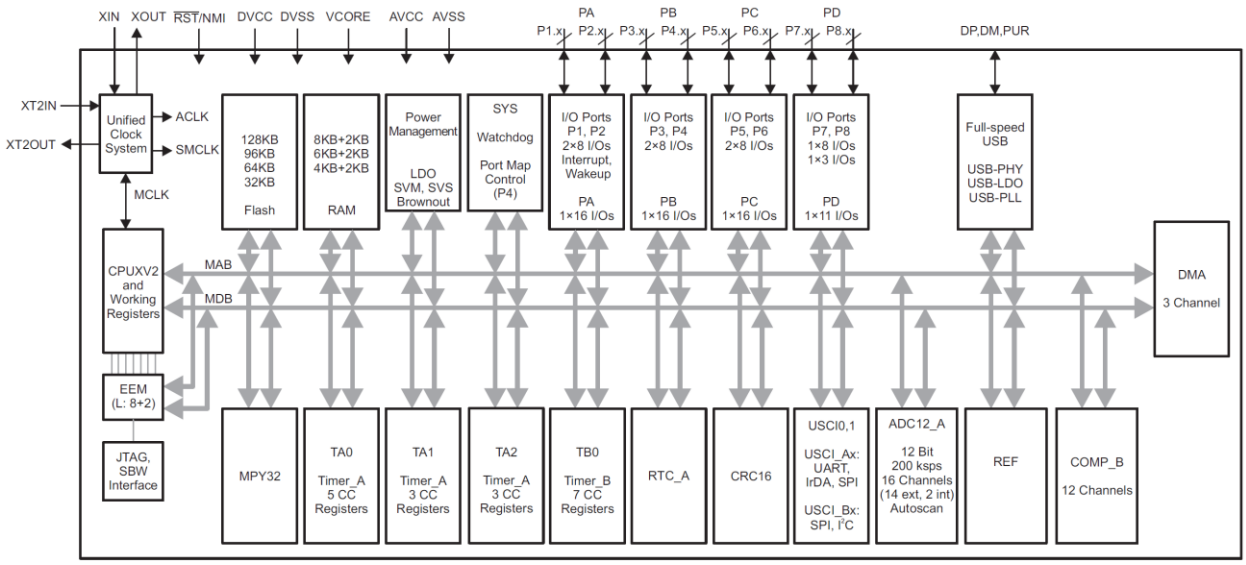


Figure 9. Block diagram of MSP430F5529.

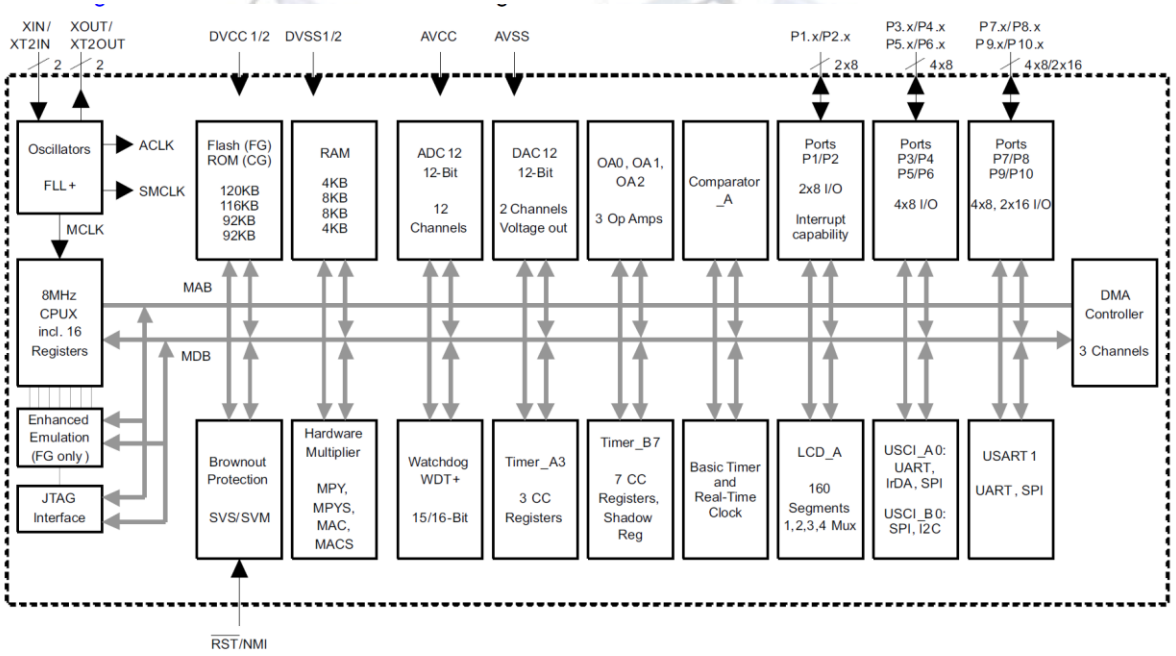


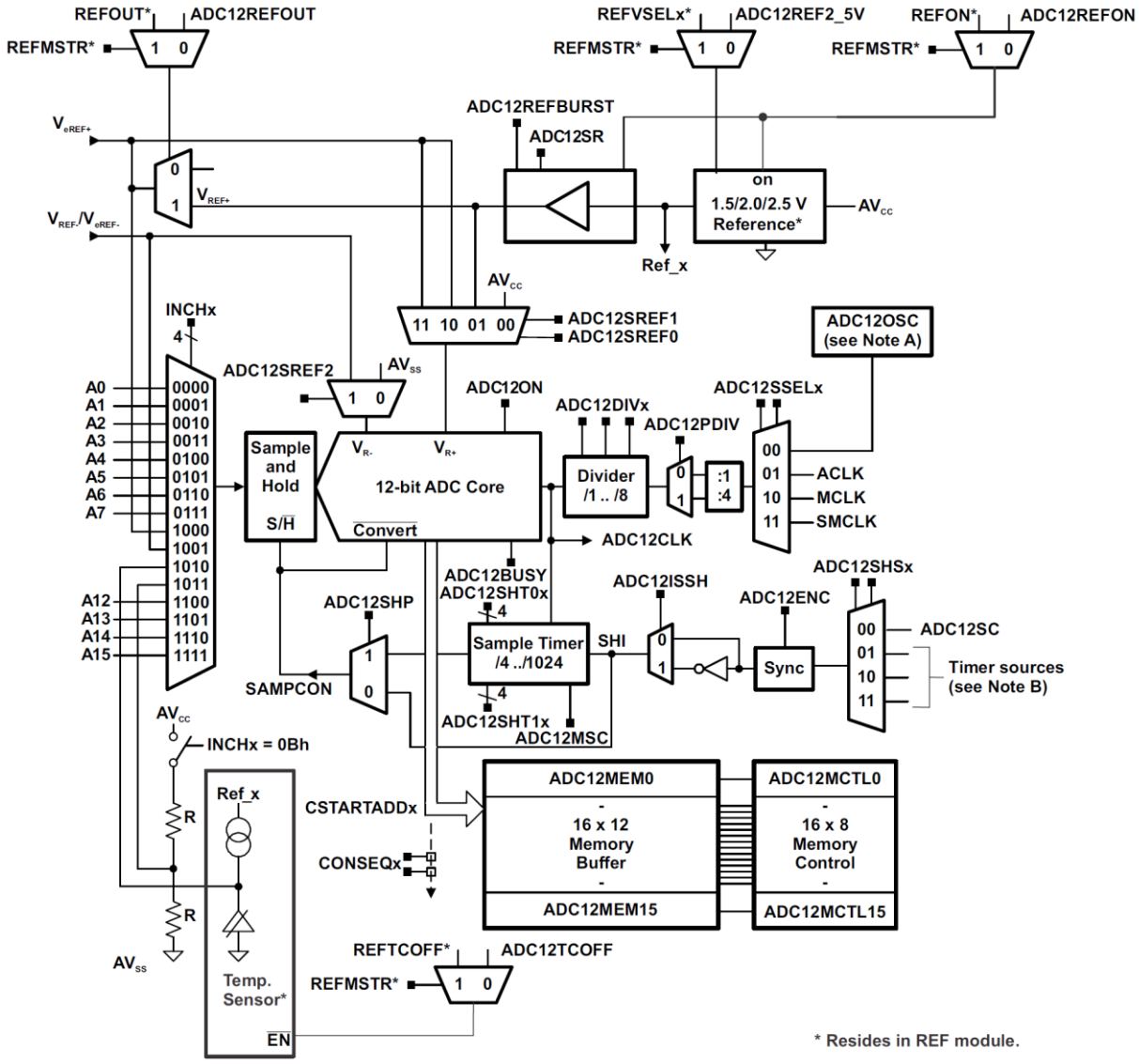
Figure 10. Block diagram of MSP430F4618.

The MSP430 ecosystem includes several other types of ADCs, e.g., ADC10, SD24_B, CTSD16, and others, but they are out of scope in this module.

5.1 ADC12_A Organization

Figure 11 shows a block diagram of the ADC12_A module (including the REF module). The ADC12_A supports 12-bit AD conversion with 16 input channels (12 external and up to 4 internal). It relies on a 12-bit SAR core, supports programmable sampling periods controlled by software

or timers, supports software-selectable internal and external reference voltages, and multiple modes of operations. It includes a 16-word conversion buffer (ADC12MEM0 – ADC12MEM15) and a 16-word control buffer (ADC12MCTL0 – ADC12MCTL15).



A ADC12OSC refers to the MODCLK from the UCS. See the UCS chapter for more information.

B See the device-specific data sheet for timer sources available.

Figure 11. Block diagram of ADC12_A.

ADC12_A Core. At the heart of the ADC12 is a 12-bit switched capacitor SAR core. It is guaranteed monotonic with no missing codes. The ADC12 core is configured by two control registers, ADC12CTL0 and ADC12CTL1. The core is enabled with the ADC12ON bit. The ADC12 can be turned off when not in use to save power. With few exceptions the ADC12 control bits can only be modified when ENC = 0. ENC must be set to 1 before any conversion can take place. The BUSY

flag is set while sampling and conversion is in progress. The result is written to ADC12MEMx memory buffers.

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 (0x0FFF) 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:

$$N_{ADC} = nint\left(4095 \cdot \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}}\right)$$

Conversion Clock. The SAR block uses ADC12CLK signal that feeds both the sample-and-hold and the SAR core blocks. The ADC12_A source clock is selected using the predivider controlled by the ADC12PDIV bit and the divider using the ADC12SSELx bits. The input clock can be divided from 1 to 32 using both the ADC12DIVx bits and the ADC12PDIV bit. Possible ADC12CLK sources are SMCLK, MCLK, ACLK, and the ADC12OSC. The ADC12OSC refers to the MODCLK 5 MHz oscillator from the UCS which can vary with individual devices, supply voltage, and temperature. The user must ensure that the clock chosen for ADC12CLK remains active until the end of a conversion. If the clock is removed during a conversion, the operation will not complete and any result will be invalid. If selected, the ADC12OSC is automatically enabled when needed and disabled when conversions have finished.

ADC12_A Inputs and 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 ADC, and the intermediate node is connected to analog ground (AVSS) so that the stray capacitance is grounded to eliminate crosstalk.

The ADC12_A uses the charge redistribution method. When the inputs are internally switched, the switching action may cause transients on the input signal. These transients decay and settle before causing errant conversion.

Analog Port Selection. The ADC12_A inputs are multiplexed with digital port pins. When analog signals are applied to digital gates, parasitic current can flow from V_{CC} 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 PxSEL.y bits provide the ability to disable the port pin input and output buffers.

Voltage Reference Generator. The ADC12_A modules have a separate reference module (REF) that supplies three selectable voltage levels, 1.5 V, 2.0 V, and 2.5 V to the ADC12_A. Any of these voltages may be used internally and externally on pin V_{REF+}. The internal AVCC can also be used as the reference. On devices with the REF module, the voltage reference settings can be controlled either by the REF module or by the ADC12_A module. This is to allow for backward compatibility with older families. This is handled by the REFMSTR bit in the REF module. If REFMSTR = 1 (default), the REF module registers control the reference settings. If REFMSTR = 0,

the ADC12_A reference setting define the reference voltage of the ADC12_A module (2.5 or 1.5 V). External references may be supplied for V_{R+} and V_{R-} through pins V_{REF+}/V_{eREF+} and V_{REF-}/V_{eREF-}, respectively. External storage capacitors are required only if ADC12REFOUT = 1 (REFOUT = 1 when using REF module) and the reference voltage is made available at the pins.

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 SHSx bits and includes the following:

- ADC12SC bit
- Up to three timer outputs (see the device-specific data sheet for available timer sources)

The ADC12_A supports 8-bit, 10-bit, and 12-bit resolution modes selectable by the ADC12RES bits. The analog-to-digital conversion requires 9, 11, and 13 ADC12CLK cycles, respectively. The polarity of the SHI signal source can be inverted with the ADC12ISSH 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. Two different sample-timing methods are defined by control bit ADC12SHP, extended sample mode and pulse mode. See the device-specific data sheet for available timers for SHI sources.

The extended sample mode is selected when ADC12SHP = 0. The SHI signal directly controls SAMPCON and defines the length of the sample period t_{sample} (Figure 12). When SAMPCON is high, sampling is active. The high-to-low SAMPCON transition starts the conversion after synchronization with ADC12CLK.

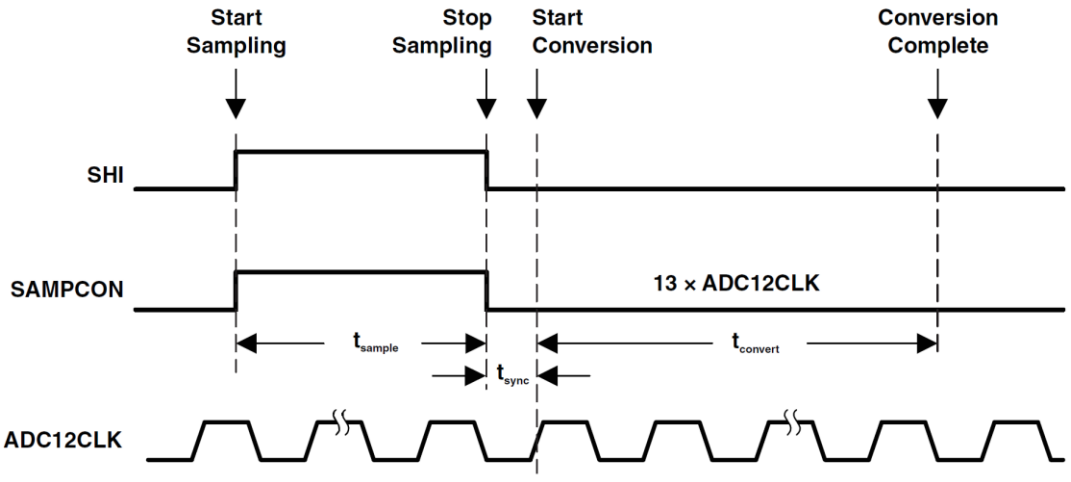


Figure 12. Extended Sample Mode.

The pulse sample mode is selected when ADC12SHP = 1. The SHI signal is used to trigger the sampling timer. The ADC12SHT0x and ADC12SHT1x bits in ADC12CTL0 control the interval of the sampling timer that defines the SAMPCON sample period t_{sample}. The sampling timer keeps SAMPCON high after synchronization with AD12CLK for a programmed interval tsample. The total sampling time is t_{sample} plus t_{sync} (see Figure 13). The ADC12SHTx bits select the sampling time in

4× multiples of ADC12CLK. ADC12SHT0x selects the sampling time for ADC12MCTL0 to ADC12MCTL7. ADC12SHT1x selects the sampling time for ADC12MCTL8 to ADC12MCTL15.

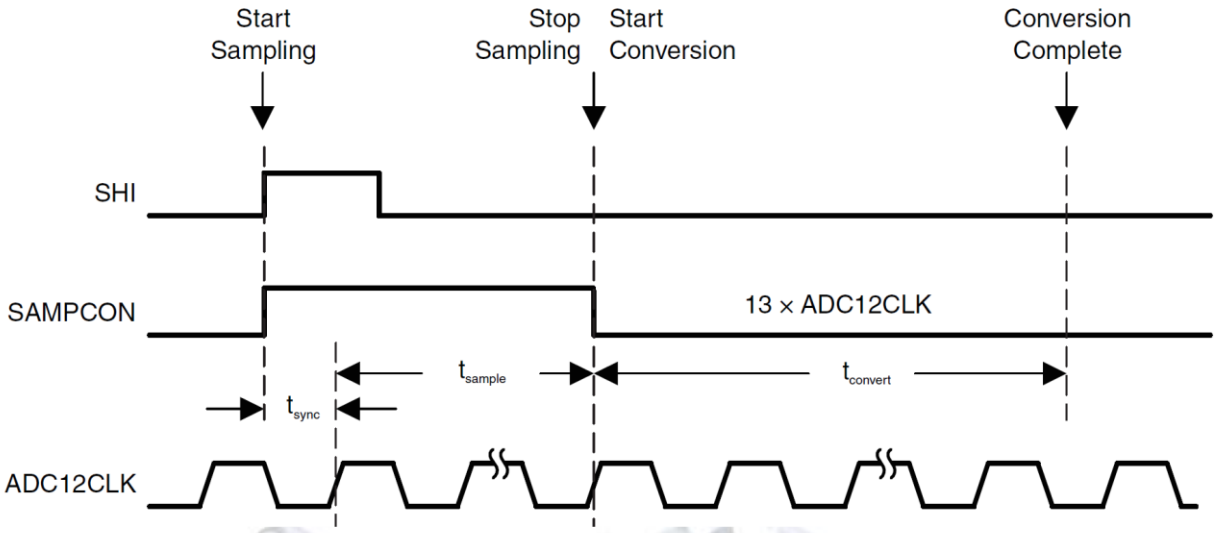
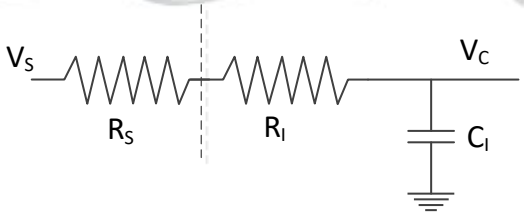


Figure 13. Pulse Sample Mode.

Sample Timing Considerations. When SAMPCON = 0, all inputs are high impedance. When SAMPCON = 1, the selected Ax input can be modeled as an RC low-pass filter during the sampling period as shown in Figure 14. An input resistance R_I ~ 1.8 KΩ, C_I ~ 25 pF. The capacitor voltage V_C must be charged to within one half of V_{LSB} of the source voltage V_S for an accurate N-bit conversion. The following equation can be used to calculate the minimum sampling time:

$$t_{sample} \geq (R_S + R_I) \cdot C_I \cdot \ln(2^{N+1}) + 800ns$$



V_I – input voltage at pin Ax
V_S – external source voltage
R_S – external source resistance
R_I – internal MUX input resistance
C_I – input capacitance
V_C – capacitance-charging voltage

Figure 14. Analog input equivalent circuit.

Conversion Memory. Conversions are specified and the results are stored as follows. There are 16 ADC12MEMx conversion memory registers to store conversion results. Each ADC12MEMx is configured with an associated ADC12MCTLx control register. The SREFx bits in the control register define the voltage reference and the INCHx bits select the input channel. The EOS bit defines the end of sequence when a sequential conversion mode is used. A sequence rolls over from ADC12MEM15 to ADC12MEM0 when the EOS bit in ADC12MCTL15 is not set. The CSTARTADDx

bits define the first ADC12MCTLx used for any conversion. If the conversion mode is single-channel or repeat-single-channel the CSTARTADDx points to the single ADC12MCTLx to be used.

If the conversion mode selected is either sequence-of-channels or repeat-sequence-of-channels, CSTARTADDx points to the first ADC12MCTLx location to be used in a sequence. A pointer, not visible to software, is incremented automatically to the next ADC12MCTLx in a sequence when each conversion completes. The sequence continues until an EOS bit in ADC12MCTLx is processed - this is the last control byte processed. When conversion results are written to a selected ADC12MEMx, the corresponding flag in the ADC12IFGx register is set.

Modes of operation. The ADC12_A has 4 conversion modes specified by the CONSEQx control bits as shown in Table 1. Consult the corresponding user guide for more information on individual modes of operation.

Table 1. Conversion Modes.

CONSEQx	Mode	Operation	Description
00	Single channel single-conversion	A single channel is converted once	A single channel is sampled and converted once. a. The ADC result is written to the ADC12MEMx defined by the CSTARTADDx bits. b. When ADC12SC triggers a conversion, successive conversions can be triggered by the ADC12SC bit. c. When any other trigger source is used, ENC must be toggled between each conversion.
01	Sequence-of-channels	A sequence of channels is converted once	A sequence of channels is sampled and converted once. a.-c. (from above) d. The sequence stops after the measurement of the channel with a set EOS bit.
10	Repeat single-channel	A single channels is converted repeatedly	A single channel is sampled and converted continuously. The ADC results are written to the ADC12MEMx defined by the CSTARTADDx bits. It is

			necessary to read the result after the completed conversion because only one ADC12MEMx memory is used and is overwritten by the next conversion.
11	Repeat sequence of channels	A sequence of channels is converted repeatedly	A sequence of channels is sampled and converted repeatedly. The ADC results are written to the conversion memories starting with the ADC12MEMx defined by the CSTARTADDx bits. The sequence ends after the measurement of the channel with a set EOS bit and the next trigger signal re-starts the sequence.

5.2 ADC12_A Control Registers

The ADC12_A peripheral is a 16-bit peripheral device with three control registers ADC12CTL0 (Figure 15), ADC12CTL1 (Figure 16), and ADC12CTL2 (Figure 17), 16 data registers ADC12MEM0-ADC12MEM15 (Figure 18), and 16 channel control registers ADC12MCTL0 – ADC12MCTL15 (Figure 19). In addition, it has ADC12IE (Figure 20), ADC12IFG (Figure 21), and ADC12IVT (Figure 22) registers.

15	14	13	12	11	10	9	8
ADC12SHT1x				ADC12SHT0x			
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12MSC	ADC12REF2_5V	ADC12REFON	ADC12ON	ADC12OVIE	ADC12TOVIE	ADC12ENC	ADC12SC
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ADC12ENC = 0

Bit	Field	Type	Reset	Description
15-12	ADC12SHT1x	RW	0h	ADC12_A sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM8 to ADC12MEM15.
11-8	ADC12SHT0x	RW	0h	ADC12_A sample-and-hold time. These bits define the number of ADC12CLK cycles in the sampling period for registers ADC12MEM0 to ADC12MEM7. 0000b = 4 ADC12CLK cycles 0001b = 8 ADC12CLK cycles 0010b = 16 ADC12CLK cycles 0011b = 32 ADC12CLK cycles 0100b = 64 ADC12CLK cycles 0101b = 96 ADC12CLK cycles 0110b = 128 ADC12CLK cycles 0111b = 192 ADC12CLK cycles 1000b = 256 ADC12CLK cycles 1001b = 384 ADC12CLK cycles 1010b = 512 ADC12CLK cycles 1011b = 768 ADC12CLK cycles 1100b = 1024 ADC12CLK cycles 1101b = 1024 ADC12CLK cycles 1110b = 1024 ADC12CLK cycles 1111b = 1024 ADC12CLK cycles
7	ADC12MSC	RW	0h	ADC12_A multiple sample and conversion. Valid only for sequence or repeated modes. 0b = The sampling timer requires a rising edge of the SHI signal to trigger each sample-and-convert. 1b = The first rising edge of the SHI signal triggers the sampling timer, but further sample-and-conversions are performed automatically as soon as the prior conversion is completed.
6	ADC12REF2_5V	RW	0h	ADC12_A reference generator voltage. ADC12REFON must also be set. In devices with the REF module, this bit is only valid if the REFMSTR bit of the REF module is set to 0. In the F54xx devices (non-A), the REF module is not available. 0b = 1.5 V 1b = 2.5 V
5	ADC12REFON	RW	0h	ADC12_A reference generator on. In devices with the REF module, this bit is only valid if the REFMSTR bit of the REF module is set to 0. In the F54xx devices (non-A), the REF module is not available. 0b = Reference off 1b = Reference on
4	ADC12ON	RW	0h	ADC12_A on 0b = ADC12_A off 1b = ADC12_A on

Bit	Field	Type	Reset	Description
3	ADC12OVIE	RW	0h	ADC12MEMx overflow-interrupt enable. The GIE bit must also be set to enable the interrupt. 0b = Overflow interrupt disabled 1b = Overflow interrupt enabled
2	ADC12TOVIE	RW	0h	ADC12_A conversion-time-overflow interrupt enable. The GIE bit must also be set to enable the interrupt. 0b = Conversion time overflow interrupt disabled 1b = Conversion time overflow interrupt enabled
1	ADC12ENC	RW	0h	ADC12_A enable conversion 0b = ADC12_A disabled 1b = ADC12_A enabled
0	ADC12SC	RW	0h	ADC12_A start conversion. Software-controlled sample-and-conversion start. ADC12SC and ADC12ENC may be set together with one instruction. ADC12SC is reset automatically. 0b = No sample-and-conversion-start 1b = Start sample-and-conversion

Figure 15. Control Register ADC12CTL0.

15	14	13	12	11	10	9	8
ADC12CSTARTADDx				ADC12SHSx		ADC12SHP	ADC12ISSH
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12DIVx			ADC12SSELx		ADC12CONSEQx		ADC12BUSY
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)

Can be modified only when ADC12ENC = 0

Bit	Field	Type	Reset	Description
15-12	ADC12CSTARTADDx	RW	0h	ADC12_A conversion start address. These bits select which ADC12_A conversion-memory register is used for a single conversion or for the first conversion in a sequence. The value of CSTARTADDx is 0 to 0Fh, corresponding to ADC12MEM0 to ADC12MEM15.
11-10	ADC12SHSx	RW	0h	ADC12_A sample-and-hold source select 00b = ADC12SC bit 01b = Timer source (see device-specific data sheet for exact timer and locations) 10b = Timer source (see device-specific data sheet for exact timer and locations) 11b = Timer source (see device-specific data sheet for exact timer and locations)
9	ADC12SHP	RW	0h	ADC12_A sample-and-hold pulse-mode select. This bit selects the source of the sampling signal (SAMPCON) to be either the output of the sampling timer or the sample-input signal directly. 0b = SAMPCON signal is sourced from the sample-input signal. 1b = SAMPCON signal is sourced from the sampling timer.
8	ADC12ISSH	RW	0h	ADC12_A invert signal sample-and-hold 0b = The sample-input signal is not inverted. 1b = The sample-input signal is inverted.
7-5	ADC12DIVx	RW	0h	ADC12_A clock divider 000b = Divide by 1 001b = Divide by 2 010b = Divide by 3 011b = Divide by 4 100b = Divide by 5 101b = Divide by 6 110b = Divide by 7 111b = Divide by 8
4-3	ADC12SSELx	RW	0h	ADC12_A clock source select 00b = ADC12OSC (MODCLK) 01b = ACLK 10b = MCLK 11b = SMCLK
2-1	ADC12CONSEQx	RW	0h	ADC12_A conversion sequence mode select 00b = Single-channel, single-conversion 01b = Sequence-of-channels 10b = Repeat-single-channel 11b = Repeat-sequence-of-channels
0	ADC12BUSY	R	0h	ADC12_A busy. This bit indicates an active sample or conversion operation. 0b = No operation is active. 1b = A sequence, sample, or conversion is active.

Figure 16. Control Register ADC12CTL1.

15	14	13	12	11	10	9	8
Reserved							ADC12PDIV
r-0	r-0	r-0	r-0	r-0	r-0	r-0	rw-0
7	6	5	4	3	2	1	0
ADC12TCOFF	Reserved	ADC12RES		ADC12DF	ADC12SR	ADC12REFOUT	ADC12REFBURST
rw-(0)	r-0	rw-(1)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Can be modified only when ADC12ENC = 0

Bit	Field	Type	Reset	Description
15-9	Reserved	R	0h	Reserved. Always reads as 0.
8	ADC12PDIV	RW	0h	ADC12_A predivider. This bit predivides the selected ADC12_A clock source. 0b = Predivide by 1 1b = Predivide by 4
7	ADC12TCOFF	RW	0h	ADC12_A temperature sensor off. If the bit is set, the temperature sensor turned off. This is used to save power. In devices with the REF module, this bit is only valid if the REFMSTR bit of the REF module is set to 0. In the F54xx devices (non-A), the REF module is not available. 0b = Temperature sensor on 1b = Temperature sensor off
6	Reserved	R	0h	Reserved. Always reads as 0.
5-4	ADC12RES	RW	2h	ADC12_A resolution. This bit defines the conversion result resolution. 00b = 8 bit (9 clock cycle conversion time) 01b = 10 bit (11 clock cycle conversion time) 10b = 12 bit (13 clock cycle conversion time) 11b = Reserved
3	ADC12DF	RW	0h	ADC12_A data read-back format. Data is always stored in the binary unsigned format. 0b = Binary unsigned. Theoretically, the analog input voltage -VREF results in 0000h, the analog input voltage +VREF results in 0FFFh. 1b = Signed binary (twos complement), left aligned. Theoretically, the analog input voltage -VREF results in 8000h, the analog input voltage +VREF results in 7FF0h.
2	ADC12SR	RW	0h	ADC12_A sampling rate. This bit selects the reference buffer drive capability for the maximum sampling rate. Setting ADC12SR reduces the current consumption of the reference buffer. 0b = Reference buffer supports up to approximately 200 ksps. 1b = Reference buffer supports up to approximately 50 ksps.
1	ADC12REFOUT	RW	0h	Reference output. In devices with the REF module, this bit is only valid if the REFMSTR bit of the REF module is set to 0. In the F54xx devices (non-A), the REF module is not available. 0b = Reference output off 1b = Reference output on
0	ADC12REFBURST	RW	0h	Reference burst 0b = Reference buffer on continuously 1b = Reference buffer on only during sample-and-conversion

Figure 17. Control Register ADC12CTL2.

15	14	13	12	11	10	9	8
Conversion Results							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
Conversion Results							
rw	rw	rw	rw	rw	rw	rw	rw

Bit	Field	Type	Reset	Description
15-0	Conversion Results	RW	undefined	Binary unsigned format: This data format is used if ADC12DF = 0. The 12-bit conversion results are right justified. Bit 11 is the MSB. Bits 15–12 are 0 in 12-bit mode, bits 15–10 are 0 in 10-bit mode, and bits 15–8 are 0 in 8-bit mode. Writing to the conversion memory registers corrupts the results. Twos-complement format: This data format is used if ADC12DF = 1. The 12-bit conversion results are left justified, twos-complement format. Bit 15 is the MSB. Bits 3–0 are 0 in 12-bit mode, bits 5–0 are 0 in 10-bit mode, and bits 7–0 are 0 in 8-bit mode. The data is stored in the right-justified format and is converted to the left-justified twos-complement format during read back.

Figure 18. Data Register ADC12MEMx.

7	6	5	4	3	2	1	0
ADC12EOS	ADC12SREFx			ADC12INCHx			
rw	rw	rw	rw	rw	rw	rw	rw

Can be modified only when ADC12ENC = 0

Bit	Field	Type	Reset	Description
7	ADC12EOS	RW	0h	End of sequence. Indicates the last conversion in a sequence. 0b = Not end of sequence 1b = End of sequence
6-4	ADC12SREFx	RW	0h	Select reference 000b = V_{R+} = AVCC and V_{R-} = AVSS 001b = V_{R+} = VREF+ and V_{R-} = AVSS 010b = V_{R+} = VeREF+ and V_{R-} = AVSS 011b = V_{R+} = VeREF+ and V_{R-} = AVSS 100b = V_{R+} = AVCC and V_{R-} = VREF-/VeREF- 101b = V_{R+} = VREF+ and V_{R-} = VREF-/VeREF- 110b = V_{R+} = VeREF+ and V_{R-} = VREF-/VeREF- 111b = V_{R+} = VeREF+ and V_{R-} = VREF-/VeREF-
3-0	ADC12INCHx	RW	0h	Input channel select 0000b = A0 0001b = A1 0010b = A2 0011b = A3 0100b = A4 0101b = A5 0110b = A6 0111b = A7 1000b = VeREF+ 1001b = VREF-/VeREF- 1010b = Temperature diode 1011b = (AVCC – AVSS) / 2 1100b = A12. On devices with the Battery Backup System, VBAT can be measured internally by the ADC. 1101b = A13 1110b = A14 1111b = A15

Figure 19. Control Register ADC12MCTLx.

15	14	13	12	11	10	9	8
ADC12IE15	ADC12IE14	ADC12IE13	ADC12IE12	ADC12IE11	ADC12IE10	ADC12IE9	ADC12IE8
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12IE7	ADC12IE6	ADC12IE5	ADC12IE4	ADC12IE3	ADC12IE2	ADC12IE1	ADC12IE0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Figure 20. ADC12IE Register (ADC12IEx = 0 – interrupt is disabled, 1 – interrupt is enabled when ADC12IFGx is set).

15	14	13	12	11	10	9	8
ADC12IFG15	ADC12IFG14	ADC12IFG13	ADC12IFG12	ADC12IFG11	ADC12IFG10	ADC12IFG9	ADC12IFG8
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12IFG7	ADC12IFG6	ADC12IFG5	ADC12IFG4	ADC12IFG3	ADC12IFG2	ADC12IFG1	ADC12IFG0
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Figure 21. ADC12IFG Register. ADC12IFGx bit is set when ADC12MEMx is loaded with a conversion result. This bit is reset if the ADC12MEMx is accessed, or it may be reset with software (0 - no interrupt is pending, 1 – interrupt is pending).

15	14	13	12	11	10	9	8
ADC12IVx							
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
ADC12IVx							
r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r-(0)	r0

Bit	Field	Type	Reset	Description
15-0	ADC12IVx	R	0h	ADC12_A interrupt vector value 00h = No interrupt pending 02h = Interrupt Source: ADC12MEMx overflow; Interrupt Flag: –; Interrupt Priority: Highest 04h = Interrupt Source: Conversion time overflow; Interrupt Flag: – 06h = Interrupt Source: ADC12MEM0 interrupt flag; Interrupt Flag: ADC12IFG0 08h = Interrupt Source: ADC12MEM1 interrupt flag; Interrupt Flag: ADC12IFG1 0Ah = Interrupt Source: ADC12MEM2 interrupt flag; Interrupt Flag: ADC12IFG2 0Ch = Interrupt Source: ADC12MEM3 interrupt flag; Interrupt Flag: ADC12IFG3 0Eh = Interrupt Source: ADC12MEM4 interrupt flag; Interrupt Flag: ADC12IFG4 10h = Interrupt Source: ADC12MEM5 interrupt flag; Interrupt Flag: ADC12IFG5 12h = Interrupt Source: ADC12MEM6 interrupt flag; Interrupt Flag: ADC12IFG6 14h = Interrupt Source: ADC12MEM7 interrupt flag; Interrupt Flag: ADC12IFG7 16h = Interrupt Source: ADC12MEM8 interrupt flag; Interrupt Flag: ADC12IFG8 18h = Interrupt Source: ADC12MEM9 interrupt flag; Interrupt Flag: ADC12IFG9 1Ah = Interrupt Source: ADC12MEM10 interrupt flag; Interrupt Flag: ADC12IFG10 1Ch = Interrupt Source: ADC12MEM11 interrupt flag; Interrupt Flag: ADC12IFG11 1Eh = Interrupt Source: ADC12MEM12 interrupt flag; Interrupt Flag: ADC12IFG12 20h = Interrupt Source: ADC12MEM13 interrupt flag; Interrupt Flag: ADC12IFG13 22h = Interrupt Source: ADC12MEM14 interrupt flag; Interrupt Flag: ADC12IFG14 24h = Interrupt Source: ADC12MEM15 interrupt flag; Interrupt Flag: ADC12IFG15; Interrupt Priority: Lowest

Figure 22. ADC12IVT Register. ADC12_A has 18 different interrupt sources.

6 Code Example

Code 1 shows a C program that measures temperature of the MSP430F5529 chip and reports it via serial asynchronous communication interface. The MSP430F5529 chip has an on-chip temperature sensor. Its transfer function is shown in Figure 23. The voltage from the temperature sensor connected to the input channel 10 (INCHx=1010). The temperature sensor is calibrated using the internal voltage references. Each reference voltage (1.5V/2.5V/2.5V) contains a measured value for two temperatures 30 °C ± 3 °C and 85 °C ± 3 °C and these are available in the TLV structure (device descriptor table residing in the flash memory containing calibration data provided by the manufacturer). The characteristic equation of the temperature sensors voltage in mV is as follows:

$$V_{SENSE} = TC_{SENSOR} \cdot Temp + V_{SENSOR}$$

where TC_{SENSOR} represents the temperature coefficient in mV/°C and V_{SENSOR} represents the y-intercept of the equation. Thus, the temperature in °C can be computed as follows:

$$Temp = (ADC(raw) - CAL\_ADC\_30) \cdot \frac{85 - 30}{CAL\_ADC\_85 - CAL\_ADC\_35} + 30$$

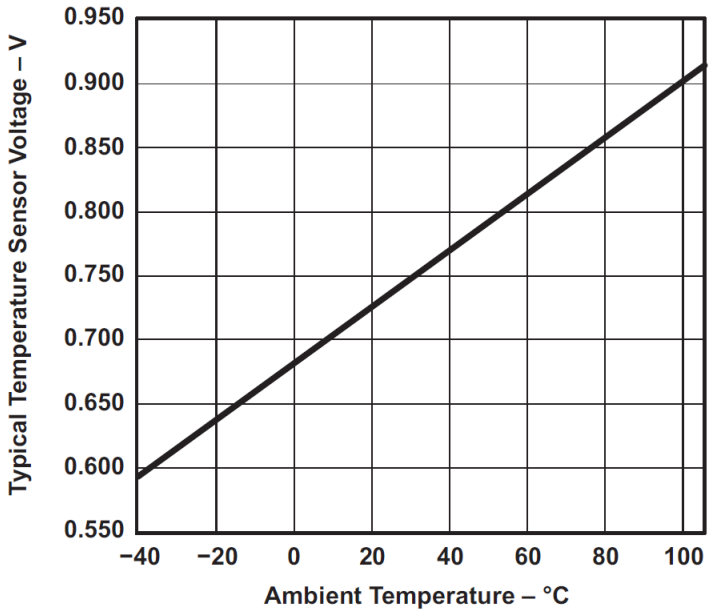


Figure 23. Typical Temperature Sensor Transfer Function.

```
/*-------------------------------------------------------------------------------
 * File:        Lab10_D1.c (CPE 325 Lab10 Demo code)
 *
 * Function:    Measuring the temperature (MPS430F5529)
 *
 * Description: This C program samples the on-chip temperature sensor and
 *              converts the sampled voltage from the sensor to temperature in
 *              degrees Celsius and Fahrenheit. The converted temperature is
 *              sent to HyperTerminal over the UART by using serial UART.
 *
 * Clocks:      ACLK = LFXT1 = 32768Hz, MCLK = SMCLK = DCO = default (~1MHz)
 *              An external watch crystal between XIN & XOUT is required for ACLK
 *
 * Instructions:Set the following parameters in HyperTerminal
 *                  Port :        COM1
 *                  Baud rate :   115200
 *                  Data bits:    8
 *                  Parity:       None
 *                  Stop bits:    1
 *                  Flow Control: None
 *
 *                       MSP430F5529
 *                   ------------------
 *                 /|\|              XIN|-
 *                  | |                 | 32kHz
 *                  --|RST          XOUT|-
```

```
 *                      |                 |
 *                      |     P3.3/UCA0TXD|------------>
 *                      |                 | 115200 - 8N1
 *                      |     P3.4/UCA0RXD|<------------
 *                      |                 |
 * Input:       Character Y or y or N or n
 *
 * Output:      Displays Temperature in Celsius and Fahrenheit in HyperTerminal
 * Author:      Aleksandar Milenkovic, milenkovic@computer.org
 *              Prawar Poudel
 *-------------------------------------------------------------------------------*/

#include  <msp430.h>
#include  <stdio.h>

#define CALADC12_15V_30C  *((unsigned int *)0x1A1A)   // Temperature Sensor
Calibration-30 C
                                                      //See device datasheet for TLV
table memory mapping
#define CALADC12_15V_85C  *((unsigned int *)0x1A1C)   // Temperature Sensor
Calibration-85 C

char ch;                  // Holds the received char from UART
unsigned char rx_flag;    // Status flag to indicate new char is received

char gm1[] = "Hello! I am an MSP430. Would you like to know my temperature? (Y|N)";
char gm2[] = "Bye, bye!";
char gm3[] = "Type in Y or N!";

long int temp;                    // Holds the output of ADC
long int IntDegF;                 // Temperature in degrees Fahrenheit
long int IntDegC;                 // Temperature in degrees Celsius

char NewTem[25];

void UART_setup(void) {

    P3SEL |= BIT3 + BIT4;   // Set USCI_A0 RXD/TXD to receive/transmit data
    UCA0CTL1 |= UCSWRST;    // Set software reset during initialization
    UCA0CTL0 = 0;           // USCI_A0 control register
    UCA0CTL1 |= UCSSEL_2;   // Clock source SMCLK

    UCA0BR0 = 0x09;         // 1048576 Hz  / 115200 lower byte
    UCA0BR1 = 0x00;         // upper byte
    UCA0MCTL = 0x02;        // Modulation (UCBRS0=0x01, UCOS16=0)

    UCA0CTL1 &= ~UCSWRST;   // Clear software reset to initialize USCI state machine
    UCA0IE |= UCRXIE;                       // Enable USCI_A0 RX interrupt
}

void UART_putCharacter(char c) {
    while (!(UCA0IFG&UCTXIFG));    // Wait for previous character to transmit
    UCA0TXBUF = c;                  // Put character into tx buffer
}
```

```
 81
 82  void sendMessage(char* msg, int len) {
 83      int i;
 84      for(i = 0; i < len; i++) {
 85          UART_putCharacter(msg[i]);
 86      }
 87      UART_putCharacter('\n');        // Newline
 88      UART_putCharacter('\r');        // Carriage return
 89  }
 90
 91  void ADC_setup(void) {
 92      REFCTL0 &= ~REFMSTR;                    // Reset REFMSTR to hand over control
 93  to
 94                                              // ADC12_A ref control registers
 95      ADC12CTL0 = ADC12SHT0_8 + ADC12REFON + ADC12ON;
 96                                              // Internal ref = 1.5V
 97      ADC12CTL1 = ADC12SHP;                   // enable sample timer
 98      ADC12MCTL0 = ADC12SREF_1 + ADC12INCH_10;  // ADC i/p ch A10 = temp sense i/p
 99      ADC12IE = 0x001;                        // ADC_IFG upon conv result-ADCMEMO
100      __delay_cycles(100);                     // delay to allow Ref to settle
101      ADC12CTL0 |= ADC12ENC;
102  }
103
104  void main(void) {
105      WDTCTL = WDTPW | WDTHOLD;         // Stop watchdog timer
106      UART_setup();                     // Setup USCI_A0 module in UART mode
107      ADC_setup();                      // Setup ADC12
108
109      rx_flag = 0;                      // RX default state "empty"
110      _EINT();                          // Enable global interrupts
111      while(1) {
112          sendMessage(gm1, sizeof(gm1));// Send a greetings message
113
114          while(!(rx_flag&0x01));       // Wait for input
115          rx_flag = 0;                  // Clear rx_flag
116          sendMessage(&ch, 1);          // Send received char
117
118          // Character input validation
119          if ((ch == 'y') || (ch == 'Y')) {
120
121              ADC12CTL0 &= ~ADC12SC;
122              ADC12CTL0 |= ADC12SC;                   // Sampling and conversion start
123
124              _BIS_SR(CPUOFF + GIE);      // LPM0 with interrupts enabled
125
126              //in the following equation,
127              // ..temp is digital value read
128              //..we are using double intercept equation to compute the
129              //.. .. temperature given by temp value
130              //.. .. using observations at 85 C and 30 C as reference
131              IntDegC = (float)(((long)temp - CALADC12_15V_30C) * (85 - 30)) /
132                      (CALADC12_15V_85C - CALADC12_15V_30C) + 30.0f;
133
134              IntDegF = IntDegC*(9/5.0) + 32.0;
```

```
135
136             // Printing the temperature on HyperTerminal/Putty
137             sprintf(NewTem, "T(F)=%ld\tT(C)=%ld\n", IntDegF, IntDegC);
138             sendMessage(NewTem, sizeof(NewTem));
139         }
140         else if ((ch == 'n') || (ch == 'N')) {
141             sendMessage(gm2, sizeof(gm2));
142             break;                      // Get out
143         }
144         else {
145             sendMessage(gm3, sizeof(gm3));
146         }
147     }                                   // End of while
148     while(1);                           // Stay here forever
149 }
150
151 #pragma vector = USCI_A0_VECTOR
152 __interrupt void USCIA0RX_ISR (void) {
153     ch = UCA0RXBUF;                 // Copy the received char
154     rx_flag = 0x01;                 // Signal to main
155     LPM0_EXIT;
156 }
157
158 #pragma vector = ADC12_VECTOR
159 __interrupt void ADC12ISR (void) {
160     temp = ADC12MEM0;               // Move results, IFG is cleared
161     _BIC_SR_IRQ(CPUOFF);            // Clear CPUOFF bit from 0(SR)
162 }
163
164
```

Code 1. Program measuring MSP430 chip temperature using on-chip temperature sensor. Try rubbing the chip package by your pointer finger and observe what happens.

7 Exercises