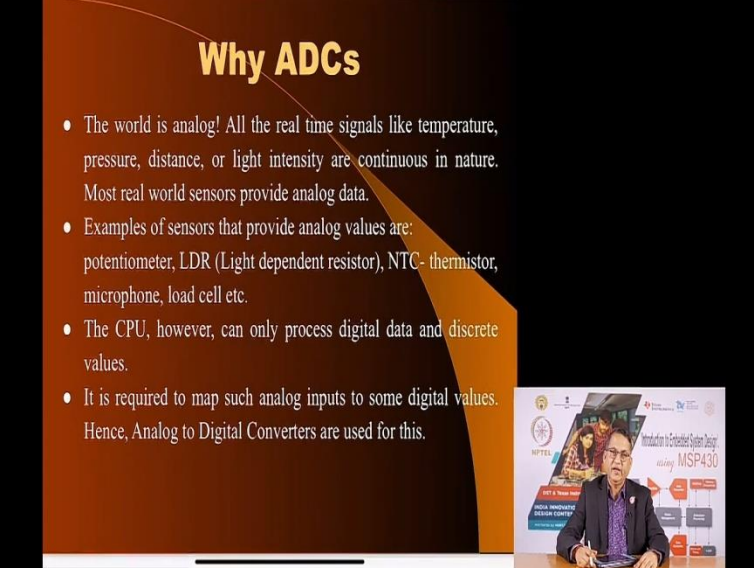


Introduction to Embedded System Design
Professor Dhananjay V. Gadre (NSUT)
&
Badri Subudhi (IIT Jammu)
Indian Institute of Technology, Delhi
Lecture 31
Analog to Digital Converter in the MSP430

Hello and welcome to a new session for this online course on Introduction to Embedded System Design. I am your instructor Dhananjay Gadre and in this session we are going to talk about another important aspect of embedded system that is the ability to read external analog voltages. And to be able to do that, we need to have the support of a peripheral which is often available on most modern microcontrollers that is the analog to digital converters.

Now why do we need ADCs? ADC means analog to digital converter. Why do we need them?

(Refer Slide Time: 1:01)



Why ADCs

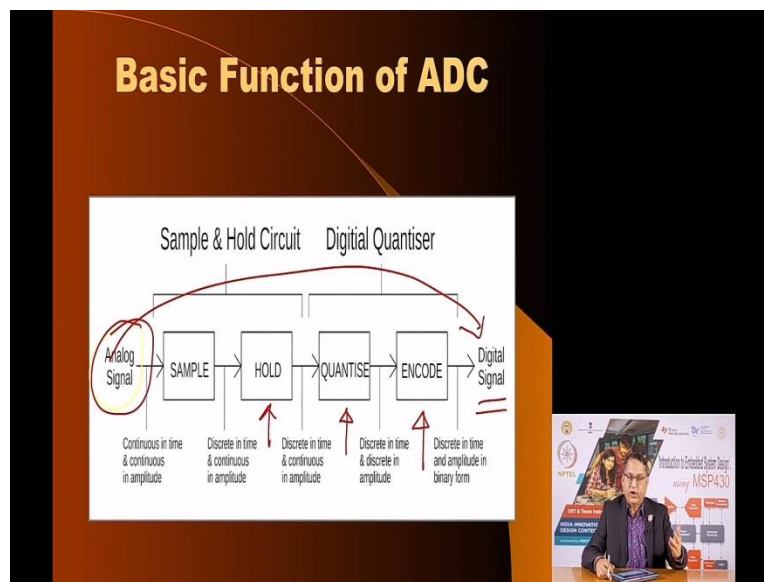
- The world is analog! All the real time signals like temperature, pressure, distance, or light intensity are continuous in nature. Most real world sensors provide analog data.
- Examples of sensors that provide analog values are: potentiometer, LDR (Light dependent resistor), NTC- thermistor, microphone, load cell etc.
- The CPU, however, can only process digital data and discrete values.
- It is required to map such analog inputs to some digital values. Hence, Analog to Digital Converters are used for this.

The slide also features a small inset video of Professor Dhananjay V. Gadre, who is seated at a desk and speaking. Behind him is a presentation board with the course title 'Introduction to Embedded System Design using MSP430' and logos for NSUT and IIT Jammu.

We need them because the world is analog which means the values that surround us, the features that are around us, they are all changing their parameters continuously. They can take any arbitrary value. Information like temperature or pressure or distance or light intensity as we have seen, they are all continuous in nature. And so we need to convert that continuous voltage continuous parameter into discrete numbers discrete values.

Examples of sensors which provide continuous values are for using a Potentiometer or Light dependent resistor or negative temperature coefficient thermistor, load cell, microphone there are so many. So problem is that the CPU cannot read analog values, it can only accept digital numbers 0s and 1s. And so we need a mechanism to convert these analog inputs in digital value and that is where an ADC comes into use. So that is the justification for having an analog to digital convertor mechanism connected to the micro-controller.

(Refer Slide Time: 2:27)




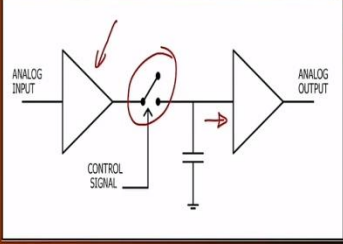
This is how ADC works. You have the analog signal in the form of the output of a particular sensor or a voltage. You need to sample it that is you decide at what point of time you want to look at this value. Now the input is asynchronous, meaning it can change whenever it wants to but while the conversion process from the analog voltage into a digital number is in process, you do not want the input voltage to change. So you include a circuit which can hold the value of the analog voltage to a fixed value to that value when you start the conversion and hold it till the conversion is over, such a circuit is called a sample and hold circuit.

Then, once you have started the conversion and you convert it, you quantise it into a discrete number and you encode it meaning assign a digital code to this quantised number. And once that is done the analog voltage which started from here is available as a digital signal ready for the micro-controller to be processed or stored or whatever else you would want to do.

(Refer Slide Time: 3:48)

Sample And Hold Circuit

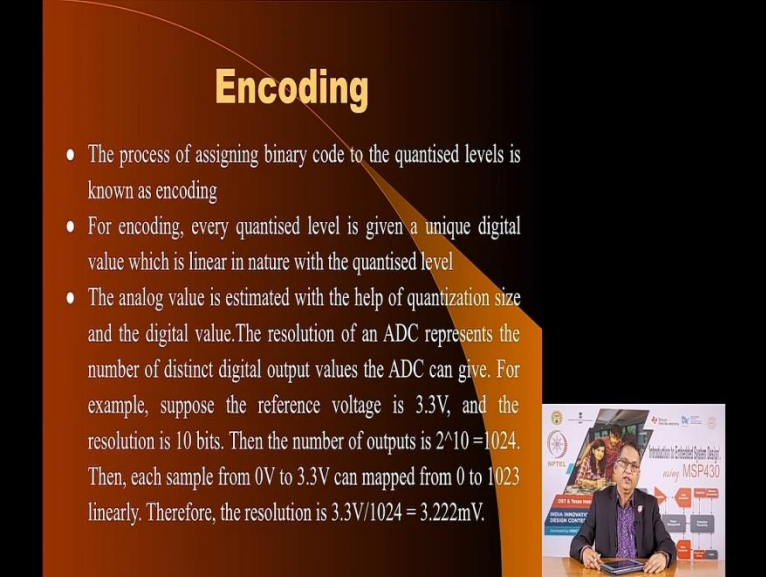
- Sample and hold circuit is required to sample analog signal (voltage) at particular intervals depending upon timing of the control signal. The sampling rate has to be higher than Nyquist Rate. The sampled value is held for a specified minimum period of time. This is done to remove the variations in signal which can corrupt the conversion process.



Here is a Sample and Hold Circuit. You have an analog input; you can pass it through an operational amplifier which provides very high input impedance using a switch, when the switch is closed it will charge its capacitor but when you open the switch the capacitor will hold the value to which it was charged. And as long as the input impedance of the subsequent amplifier is very high the capacitor is not going to discharge. Therefore, it will hold the voltage to a fixed state fixed value for the ADC to complete its function.

Quantisation, this is the next important step. After the sample and hold process which we have seen, the signal becomes discrete in time, meaning it has been held to a value but it is still continuous in amplitude meaning the value could have been any value in the range that is acceptable. You need to have a reference voltage against which this analog voltage will be compared and will be divided in small discrete steps and from there on it will be converted into a number.

(Refer Slide Time: 5:06)

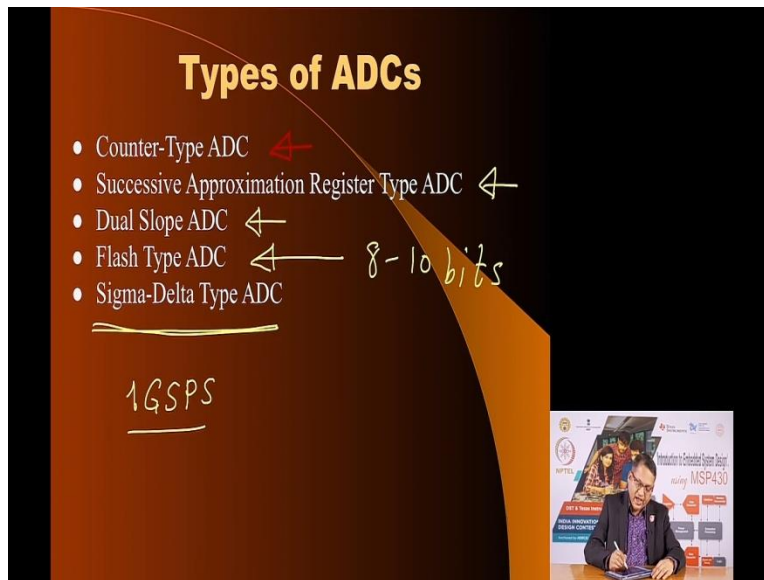


Encoding

- The process of assigning binary code to the quantised levels is known as encoding
- For encoding, every quantised level is given a unique digital value which is linear in nature with the quantised level
- The analog value is estimated with the help of quantization size and the digital value. The resolution of an ADC represents the number of distinct digital output values the ADC can give. For example, suppose the reference voltage is 3.3V, and the resolution is 10 bits. Then the number of outputs is $2^{10}=1024$. Then, each sample from 0V to 3.3V can be mapped from 0 to 1023 linearly. Therefore, the resolution is $3.3V/1024 = 3.222mV$.

This is where the encoding comes into picture, the process of assigning a binary code. Do you want it to be linearly converted or do you want it to have a non linear function or do you want to have a 2's complement. All this you can include in the encoding function of the ADC.

(Refer Slide Time: 5:23)



There are many types of ADCs, depends on the kind of functions that you want to perform that is the kind of input that you have, the kind of speed that you want with the AC because ADC is not a very fast process. It takes some time for the sample and hold and rest of the circuit to operate. And therefore, each type of ADC has a specific application. The simplest of the ADC is called a Counter ADC, Counter-Type of ADC which we use as a counter and a comparator to convert the analog voltage into a number.

The most common ADC is called Successive Approximation Type of ADC and the ADC that we have on MSP430 is actually a Successive Approximation Type of ADC SAR. Then we have a Dual Slope it is also is a counter type except now you use two slopes, one where you are charging the voltage to the input value and then from the input value you discharge to a non reference value.

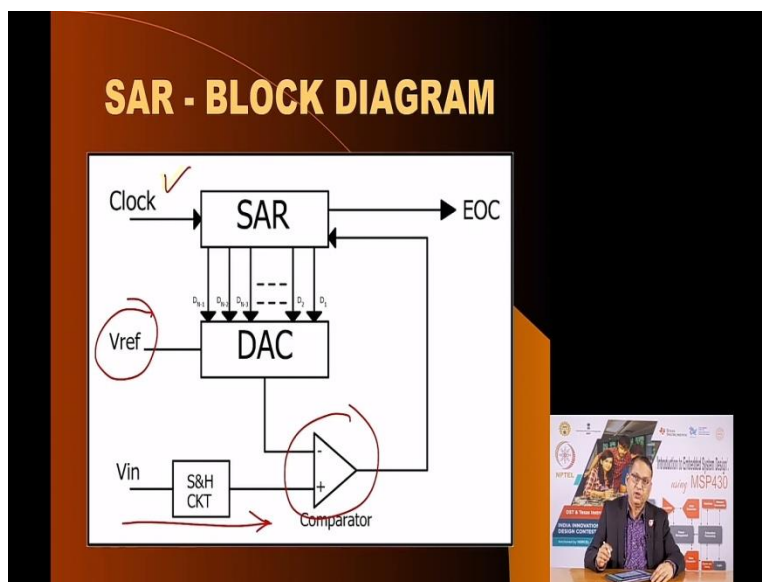
Then you have very fast ADCs of the type called Flash ADCs. In fact, the most common use of ADC is that you would notice that in your engineering life would be in the use of a digital storage oscilloscope. A DSO is a digital device but the voltage is that it is able to sample are analog in nature. And therefore at the front end after you have your amplifier you have a ADC and the ADC you have in these DSO are f the type called Flash ADC. The reason why they are called Flash is because they are very very fast.

But because they are very fast the kind of fastness comes from the architecture of these ADCs, they consume a lot of power. And usually these ADCs offer a resolution from 8 to 10 bits but they offer blistering speeds. For example if you look at the front panel of the DSO it would say, for example the one we had in our lab says 1 GSPS. This stands for one giga samples per second, this is the kind of ADC that we have on these modern digital storage oscilloscopes.

And then the next type of ADCs that we have is called Sigma-Delta type of ADC. They are used in voice encoding. So these are the various types. But since our micro-controller is a general purpose microcontroller, is not specifically useful for high speed conversions. The ADC that it has is a SAR type ADC. So we are going to look at ADC in this context.

This is a ADC where you do binary search, compare analog voltage that is applied to the ADC with binary with a voltage generated through a binary number. And when the number matches then you stop the conversion. You can go through this slide to go through the entire conversion process.

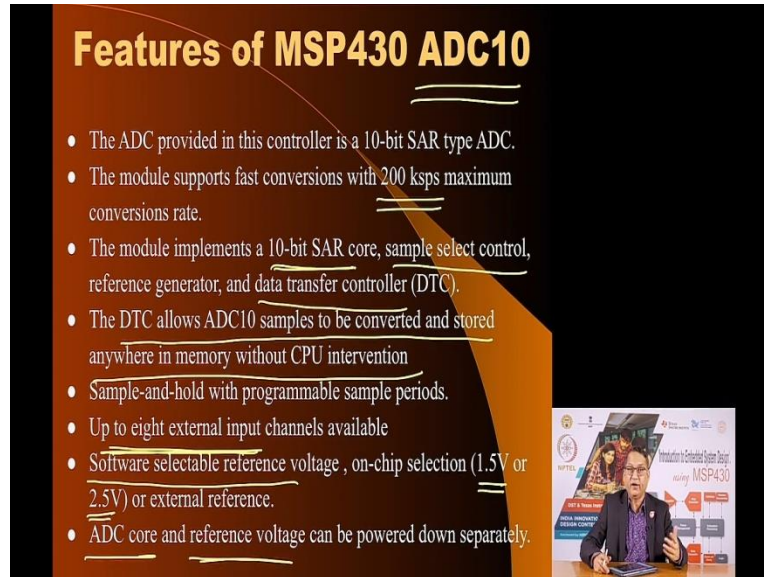
(Refer Slide Time: 8:34)



This is the block diagram of the SAR type of ADC. SAR type of AD requires a clock, so you need to decide what type of clock source are you going to use in your MSP430 ADC. We have option of a lot of clock sources. You also need a reference voltage against which the values will

be compared in this comparator. Here is the input voltage held to a value using the sample and hold circuit and when that value matches then you stop the conversion.

(Refer Slide Time: 9:10)



Now these are the features of ADC on the MSP430, we call it ADC10. The reason is that resolution of ADC is a 10-bit resolution, which means from the entire range of input voltages that it can accept, the range is divided in 10-bits and therefore the number of levels that you have from the minimum to maximum is 2 raised to power 10 which is 1024 twenty four discrete values.

The ADC on the MSP430 offers quite a fast conversion speed. These conversion speeds are up to 200 kilo samples per second, of course these are very small compared to the ADCs that are available in digital storage oscilloscopes as I mentioned, they offer speeds up to one giga samples or more giga samples per seconds.

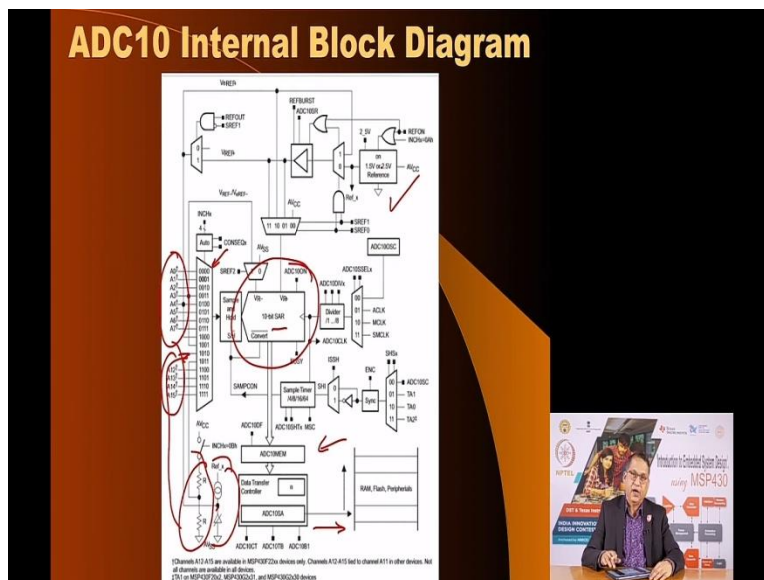
The module implements a 10-bit SAR and it has a mechanism to select the input channel because in this case you do not have multiple ADCs on your micro-controller. You have only one ADC but you have an analog multiplexer which allows you to select a particular source of voltage through the analog multiplexer and use the common ADC feature to convert that voltage and then you can change the multiplexer and convert another voltage.

The reference, the converted value is handled by a data transfer controller called a DTC. The DTC allows the samples to be converted and stored anywhere in the memory without the CPU intervention. So your program simply instructs the ADC which channel you want to convert and where do you want to store the results.

The DTC will take care of it. You can do up to eight external input channels you can convert voltages on eight external inputs, of course not simultaneously but in a sequential or serial fashion. You can through software select the reference voltage, the MSP430 micro-controller offers on-chip reference voltage which you can select to be of 1.5 volts or 2.5 volts or if you want you have a special requirement you can provide external voltage reference also through a available pin.

And the ADC core that is the peripheral which controls the ADC and the reference voltage can be powered down meaning you can enter a low power mode so that these peripheral devices do not consume power.

(Refer Slide Time: 11:46)



Here is the block diagram of the ADC. This is the core part of the ADC here, this is the actual convertor. Here is the analog multiplexer that allows you to select so many input channels. Here is the part which allows you to select the reference voltage. You can have an internal reference voltage of 1.5 or 2.5 volts. You can have internal, interestingly MSP430 has an internal temperature sensor and the output of that temperature sensor can also be routed through the

multiplexer into the ADC so that you can determine the temperature of the micro-controller, the temperature of the chip of the micro-controller.

And you can also have, read the voltage through the divider here. Now, once the ADC converts the value it is sent to this ADC10 memory from where it can decide to send it to appropriate memory location so that the processor can implement a new conversion. So this is the basic block diagram of the ADC that we have on MSP430.

Refer Slide Time: 13:05)

ADC Operation

- The ADC core converts an analog input to its 10-bit digital representation and stores the result in the ADC10MEM register
- The reference voltages can be selected by the user
- The digital output is full scale when the input is higher or equal than the positive reference voltage and zero when the input signal is equal to or lower than negative reference voltage
- The conversion formula for the 10-bit ADC result when using straight binary format is:

$$N = 1023 * (V_{in} - V_{r-}) / (V_{r+} - V_{r-})$$

- The ADC10 source clock is selected using the ADC10SSELx bits and can be divided from 1 to 8 using the ADC10DIVx bits. Possible ADC10CLK sources are SMCLK, MCLK, ACLK, and internal oscillator ADC10OSC.

The slide features a dark background with orange and white text. A small inset image in the bottom right corner shows a man in a suit sitting at a desk with a laptop, with a presentation slide titled 'Introduction to Embedded System Design using MSP430' visible behind him.

The ADC core converts analog voltages using a 10-bit representation. The reference voltage as we have discussed can be selected by the user. The digital output full scale is when the input is higher or equal to the positive reference voltage and a 0 when it is 0 or equal to or less than the negative reference voltage. In this case it is 0.

The conversion formula for the binary format of the number is this that is the N bits that you will get, the value of the 10-bits that you will get will be represented by this information. Here is the input voltage and here is the reference voltage with which it is being compared. The clock that is the rate at which the comparison will be made in this excessive approximation register module of the ADC, the clock source can be selected from a variety of sources. And this is available through these registers through these bits.

And the sources could be SM clock, M clock or A clock and you could also have an extra internal dedicated oscillator called ADC10 oscillator.

(Refer Slide Time: 14:30)

Modes of operation

The ADC10 module has four operating modes which can be selected accordingly: ✓


1. Single channel single-conversion- In this mode, only a single channel is converted once. ✓
2. Sequence-of-channels- In this mode, a sequence of channels is converted once. ✓
3. Repeat single channel- In this mode, a single channel is converted repeatedly. ✓
4. Repeat sequence-of-channels- In this mode, a sequence of channels is converted repeatedly. ✓

The ADC on MSP430 offers multiple ways of conversion. One of them is that you can do a single channel conversion or you can do sequence of channels meaning you can do channel 1,2,3...1,2,3 like that. You could repeat a single channel that is selected channel you can have continuous conversion or a sequence of channels you can do repeat of conversions on those channels.

(Refer Slide Time: 15:02)

ADC10 Registers

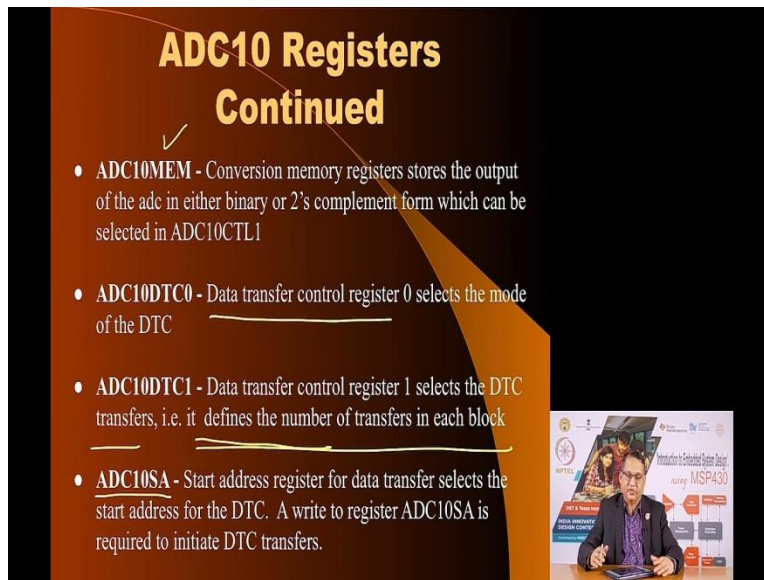
- **ADC10CTL0** - Control register 0 of ADC10 selects the reference voltage, sample time, etc. It also enables the ADC Conversions and Interrupts.
- **ADC10CTL1** - Control register 1 of ADC10 selects the input channel, output data format, clock source and divider,
- **ADC10AE0** - Analog enable control register 0 enables the corresponding pin for analog input A0 - A7
- **ADC10AE1** - Analog enable control register 1 enables the corresponding pin for analog input A12-A15 (for the devices which have them)



These are the registers associated with the 10-bit ADC. You have the control register 0 which selects the reference voltage, the sample time and it also allows you to generate interrupts once the conversion is over. So that your micro-controller can keep on doing something whenever a conversion on the selected channel is complete, the micro-controller the ADC module will tell the micro-controller that the conversion is complete and please look at the numbers and do whatever you can with it.

You have another control register 1 which selects the input channel, the format of the input data, the clock source and the divider. Then you have the analog ADC 10 analog enable 10 control register 0 which is used to select analog inputs A0 to A7. And control register 1 which allows you to select inputs A12 to A15. And this is only available for devices which have those additional pins.

(Refer Slide Time: 16:08)



ADC10 Registers Continued

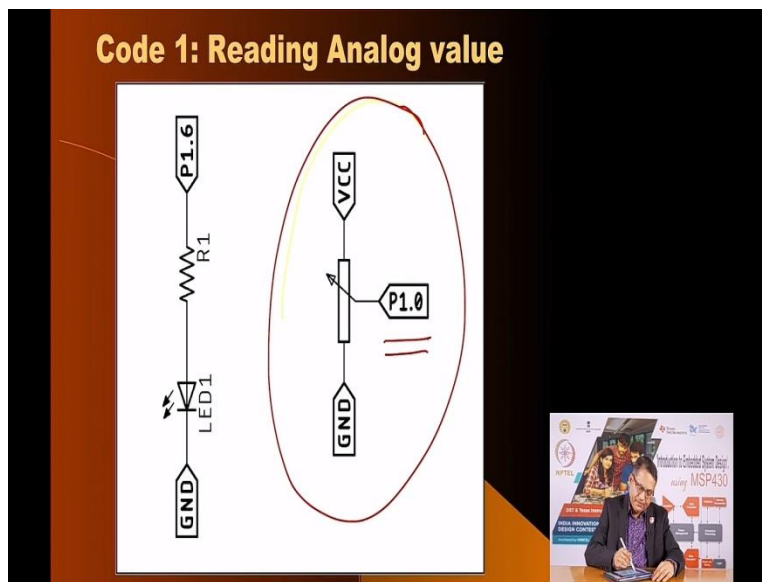
- **ADC10MEM** - Conversion memory registers stores the output of the adc in either binary or 2's complement form which can be selected in ADC10CTL1
- **ADC10DTC0** - Data transfer control register 0 selects the mode of the DTC
- **ADC10DTC1** - Data transfer control register 1 selects the DTC transfers, i.e. it defines the number of transfers in each block
- **ADC10SA** - Start address register for data transfer selects the start address for the DTC. A write to register ADC10SA is required to initiate DTC transfers.

Then you have ADC10MEM register which allows you to store the result in binary or 2s compliment format as selected in the control register. This ADC10DTC0 is date transfer control register 0 which selects the mood of the conversion control, the transfer control.

ADC10DTC1 selects the DTC transfer whether it defines the number of transfers in each block. And then the start register once you have start register address, once you have started the conversion and you want to convert a whole set of numbers where do you want to store them because if you overwrite a memory location then there is no point in having a series of conversions.

So you have you can define the start address and it will convert the first value and store it at the start address and the next address and so on. These are the registers and the memory locations. I strongly recommend that you go through these slides so that you understand that this is nothing but a repeat of what I have just gone through. Now what we are going to do is we are going to look at the few exercises so that you can get familiar with the whole process of initializing the ADC, selecting an appropriate channel, selecting the clock for the conversion and once the result is available what do you want to do with that result, you want to store it in the memory location or do you want to send it to some output port or change the duty cycle of a timer for that matter.


(Refer Slide Time: 18:03)



So we are going to look at some examples. In the first example the code example we are going to read the analog voltage connected using this mechanism where we have connected a potentiometer. And the central tap of the potentiometer we have connected to the P1.0, we will ensure that the P1.0 becomes an ADC input. And the result of the ADC will be used to send to LED connected to P1.6, remember P1.6.

(Refer Slide Time: 18:28)

```
1#include <msp430.h>
2
3#define GREEN BIT6
4#define AIN BIT0
5
6/**
7 *@brief This function maps input range to the required range
8 *@param long Input value to be mapped
9 *@param long Input value range, minimum value
10 *@param long Input value range, maximum value
11 *@param long Output value range, minimum value
12 *@param long Output value range, maximum value
13 *@return Mapped output value
14 **/
15 long map(long x, long in_min, long in_max, long out_min, long out_max) {
16     return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
17 }
18
19 /**
20 * @brief
21 * These settings are wrt enabling ADC10 on LUNCHBOX
22 **/
23 void register_settings_for_ADC10()
24 {
```



So this is what the definitions are, you have decided that you are going to connect a green LED to bit 6. You are going to have ADC on bit 0. Then we have a map function, we will come to this later.

(Refer Slide Time: 18:44)

```
long map(long x, long in_min, long in_max, long out_min, long out_max) {
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

/**
 * @brief
 * These settings are wrt enabling ADC10 on Lunchbox
 */
void register_settings_for_ADC10()
{
    ADC10AEB |= AIN; // P1.0 ADC option select ✓
    ADC10CTL1 = INCH_0; // ADC Channel -> 1 (P1.0)
    ADC10CTL0 = SREF_0 + ADC10SHT_3 + ADC10ON; // Ref -> Vcc, 64 CLK 5BH
}

/**
 * @brief
 * These settings are wrt enabling TIMER0 on Lunchbox
 */
void register_settings_for_TIMER0()
{
    P1DIR |= GREEN; // Green LED -> Output
    P1SEL |= GREEN; // Green LED -> Select Timer Output
    CCR0 = 255; // Set Timer0 PWM Period
    CCTL1 = OUTMOD_7; // Set TAB.1 Waveform Mode
    CCR1 = 1; // Set TAB.1 PWM duty cycle
    TACTL = TASSEL_2 + MC_1; // Timer Clock -> SMCLK, Mode -> Up Count
}
```

Here are the register settings for ADC 0. What are we going to do, we are going to use P1.0 for ADC input and we are going to use Vcc as the reference and we are going to use 64 clock cycles for sample and hold. This allows the values to be stabilized. Then for the timer 0 we are going to connect the timer output to P1.6 and we have selected P1.6 as the output and we have selected to be timer output.

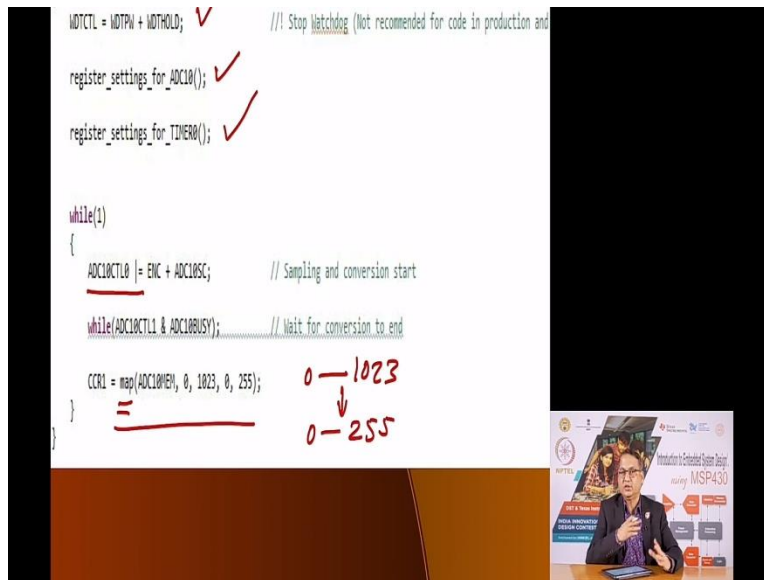
And we have CCR0, you may recall that in our previous discussions we discussed that CCR0 decides the resolution and the time period of the PWM signal. So we have decided CCR0 value to be 255 which means we have chosen 8-bit ADC, 8-bit PWM. The we are choosing the output mode to be output 7 which means PWM output and the initial value of CCR1 is 1. And the clock source of the timer is going to be SM clock and it is going to operate the timer in the up count mode.

(Refer Slide Time: 20:10)

```
WDTCTL = WDTPW + WDTHOLD; ✓ // Stop Watchdog (Not recommended for code in production and
register_settings_for_ADC10()); ✓
register_settings_for_TIMER0(); ✓

while(1)
{
  ADC10CTL0 |= ENC + ADC10SC; // Sampling and conversion start
  while(ADC10CTL1 & ADC10BUSY); // Wait for conversion to end
  CCR1 = map(ADC10MEM, 0, 1023, 0, 255);
}

// Handwritten notes:
// 0 - 1023
// ↓
// 0 - 255
```

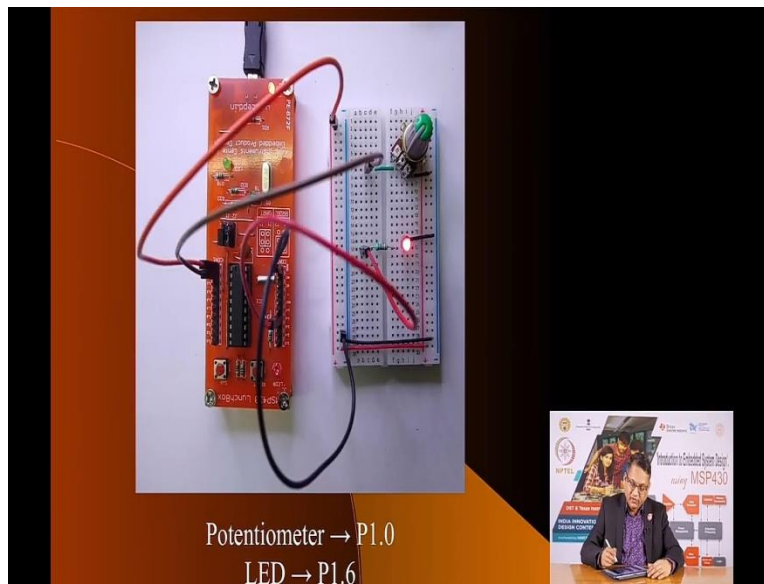


Here is the main code, as usual we have stopped the watchdog timer and we have called this function for register setting for the ADC and register setting for the timer 0. And then we simply enter infinite loop in which we read the value of the ADC and once the ADC tells that it has completed the conversion we simply call this map function.

Now the ADC result is a 10-bit value but our PWM requirement is 8-bits. So we have to map 10-bits to 8-bits. Instead of doing the bits yourself, we call this map functions. So basically it does 0 to 1023 values the output of the 8-bit, 10-bit ADC be mapped to 0255. And the best way to think of it is imagine that out of those 10-bits the least two bits are thrown off that is how you can map very easily a 10-bit number into a 8-bit output. And so what you should see when you compile and output load this into your lunchbox.

Let me correct myself here LED is connected to P1.6 which as you recall in our previous exercise I not an onboard LED, onboard LED is on a different pin.

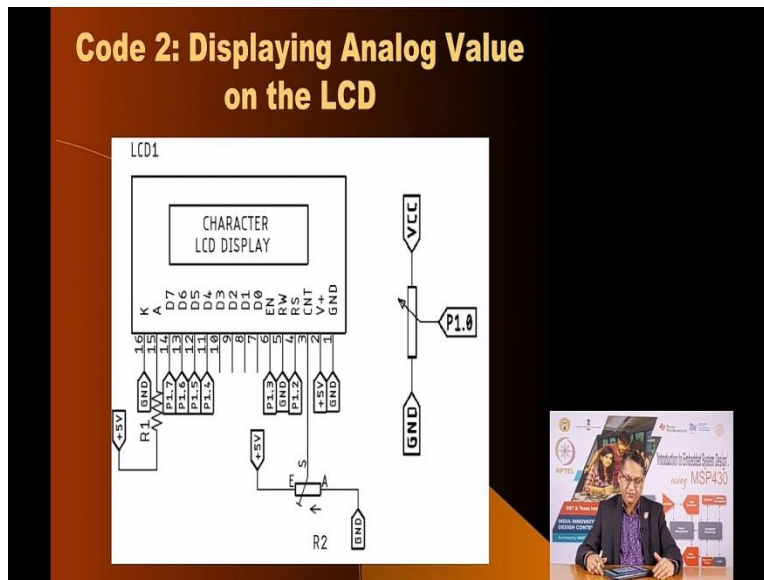
(Refer Slide Time: 21:44)



Here an onboard LED is on a pin which does not have a PWM function. We have connected the LED to P1.6 which I connected to here so you need a bread board to connect a LED. Here is another bread board connection which allows you to connect a potentiometer to an appropriate pin as discussed earlier to the ADC input. Now when you move the potentiometer from minimum to maximum, you should see that the LED intensity goes from off to fully lit condition.

And you will see the intensity of the LED can be changed by changing the potentiometer setting. So this is the objective of the code. Is strongly recommend that you download this code, build it and send this code into the memory of lunch box and make this connection on the bread board to see the impact.

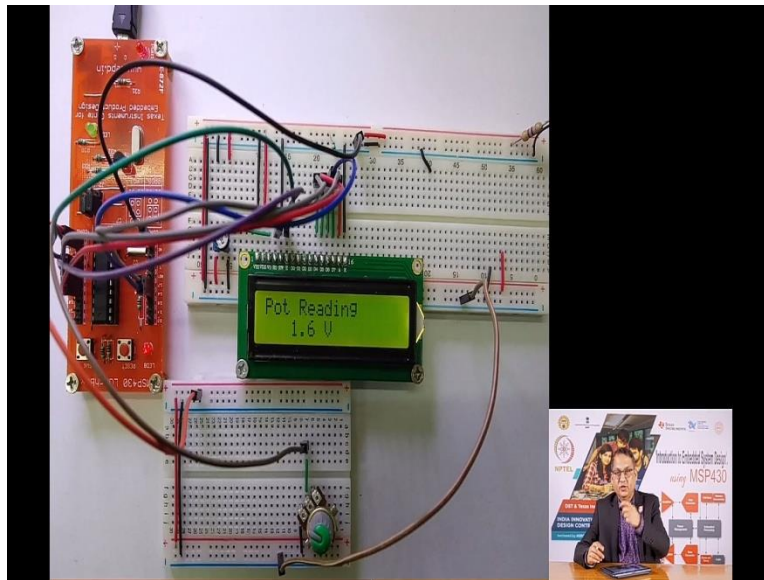
(Refer Slide Time: 22:35)



Now we have another variation of the same code not much is changing here except instead of modulating the PWM duty cycle we are just printing the value as read by the ADC and printing it on our LCD. We already covered LCD, so you know how LCD can be connected to the MSP430 lunchbox. In this case the ADC is still connected to P1.0; these are the same connections that we considered last time when we were talking about the LCD. So those connections remain the same.

Here is the code, I am not going to go through that code this is nothing but a mix of code for the previous exercise of reading the ADC value of the connected to the potentiometer and changing the duty cycle instead of changing the duty cycle now we are outputting a number onto the LCD.

(Refer Slide Time: 23:35)



Here is the display that you see, it has been converted into a voltage instead of the binary number it has been converted to a voltage. So you should see the voltage going from 0 to 3.3 volts because we are choosing the reference to be 3.3 volts which is the power supply voltage.

(Refer Slide Time: 23:48)

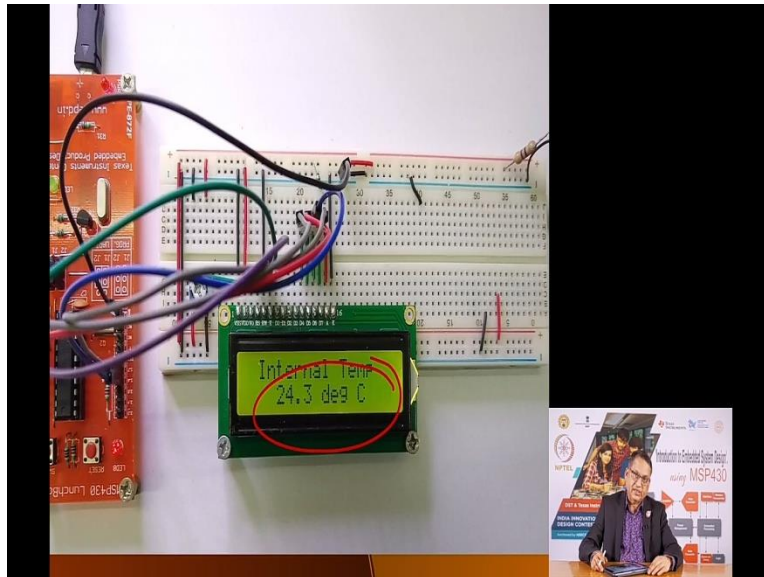
Example Code 3: Reading Internal Temperature sensor value

- $Temp = (Vtemp - 0.986) / 0.00355$
- $Vtemp = ADC10MEM * 1.5 / 1023$
- $Temp = (ADC10MEM - 673) * 423 / 1023$

Now in the block diagram of the ADC you saw that the ADC also has an input from the temperature sensor that is on-chip temperature sensor and we can modulate the ADC control registers so that we can select that channel which is connected to the on-chip temperature sensor. So in this third code example what we are going to do is read the value, convert it in the formula

that is given here recommended in the application note and user manual of MSP430 and the number that you get out of that ADC can be directly mapped to the actual temperature. These are the formulas that you have.

(Refer Slide Time: 24:46)



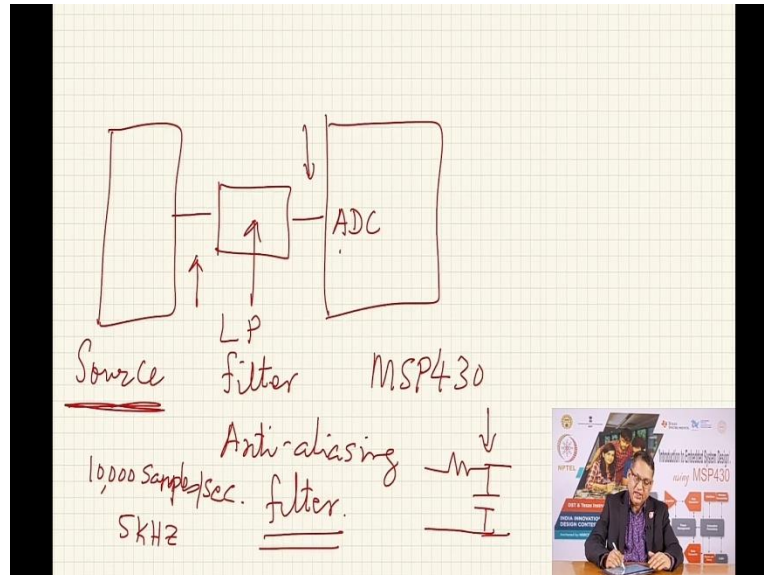
And once you understand this code, build and download it into the MSP430 lunchbox, what you would see is that the display would tell you the temperature at which the MSP430 is currently operating at and you can see the impact of it by trying to cool it or maybe you connect your you know the soldering iron gently onto the MSP430 IC and you should see this temperature reading changing. This is the current temperature when this experiment was being performed; this is the temperature of the IC at that point of time.

So I strongly recommend that you go through the codes, the three codes that have been discussed here. The first one was to read the value of the potentiometer and change the duty cycle of the LED connected on P 1.6. In second code example we took the value coming out from the potentiometer but instead of manipulating anything else we started displaying it on the LCD.

And in the third code example what we did was, we took the instead of reading the voltage from an external source we use the internal temperature source to feed the ADC, converted the values, applied these equations to convert these numbers into temperature and then printed the value onto the LCD display.

So this will give you a brief outline and idea about how to use the ADC and how to convert analog voltages connected to the ADC input of the MSP430 and use them in any which way.

(Refer Slide Time: 26:29)



Now one every important point I want to make here is that in any of the cases that you have here, let us draw a block diagram. So here is your source of analog voltage which could be a microphone or it could be a temperature sensor or it could be any other source. You should not connect, let us say this is our MSP430, and let us say this is the ADC input. It is not recommended that you convert; you connect the output of this source directly reason being depending upon the rate at which the source voltages may change.

Although you are saying that your sample and hold is going to hold the value it is still recommended that you incorporate a filter, a low pass filter in this and this low pass filter is often called Anti- aliasing filter, such that the conversion rate that you hope to have in your ADC the input source remains much lower in frequency compared to the conversion rate. And in our previous examples because the input voltages are not changing that much fast, we did not incorporate such a anti-aliasing filter.

But an anti-aliasing filter is nothing but a low pass filter which can make out of using the R and C. And the requirement is that the this filter will be less than much less than half the conversion rate.

So to take an example, suppose your ADC is going to convert at ten thousand samples per seconds. Now because of the nyquist rate, we require that the input to the ADC should never the frequency of the input connected to the ADC input should never be more than five thousand hertz, five thousand times or seconds, it should not change. And therefore this filter should, the low pass filter should have a bandwidth of five kilo hertz less than five kilo hertz.

So the input should never exceed that, so you should always include a low pass filter to serve as an anti aliasing filter. In our case in the input filters we saw, a potentiometer or an internal temperature sensor it is not going to change so fast because we are converting the ADC at a much higher rate. Therefore, we are not bothered about having an external anti aliasing filter but wherever you expect that the input frequency could go higher, you need to remove those elements from the spectrum so that the ADC can perform as per the nyquist rate.

So having an anti-aliasing filter is very important. And I hope that in any application where you expect the input source to have a component which is exceeding the nyquist rate, that you would cut down the source using a low pass filter to conform to the nyquist rate. So I hope that this lecture has introduced you to the great feature of 10-bit ADC that we have on MSP430.

It offers so many input channels; it also has an internal temperature sensor as an input to the ADC, an internal resistor divider which allows you to read the live voltage that is available on the MSP430 that powers the MSP430. You can read that through the ADC. I hope you will play with the ADC features and find how any various ways external signals you can convert using the ADC of MSP430.

In one of the next lectures we are going to not only look at some applications of the ADC by converting voltages being produced by thermistors but we are also going to use the ADC as the mechanism to give us random numbers or at least the seed for a random number. We are also going to consider how MSP430 can be programmed with additional hardware to produce analog voltages.

In this case we have seen how MSP430 can read analog voltages but we also need to generate analog voltages. We have already seen one mechanism which is using the PWM and using the low pass filter. But we will see additional mechanisms by which we can generate analog output voltages.

So I will see you very soon in a new lecture. Thank you. Bye bye.