Introduction to Embedded System Design

Professor Dhananjay V. Gadre
Electronics and Communication Engineering
Netaji Subhas University of Technology
Badri Subudhi
Electrical Engineering Department
Indian Institute of Technology, Jammu

Lecture 35
Coding Ninja

Hello and welcome to a new session. I am Dhananjay Gadre and we are here because I am offering a course on Introduction to Embedded System Design. In this lecture today we are going to get involved with programming aspects and we hope to convert you into a coding ninja. That is, we hope that at the end of this lecture you would be able to program like a ninja, like an expert.

Till now we have covered so many features about MSP430 microcontroller. We have done experiments related to digital input and output. We have done ADC. We have done timer experiments. We have introduced ourselves to the concept of interrupts. We have looked at serial communication.

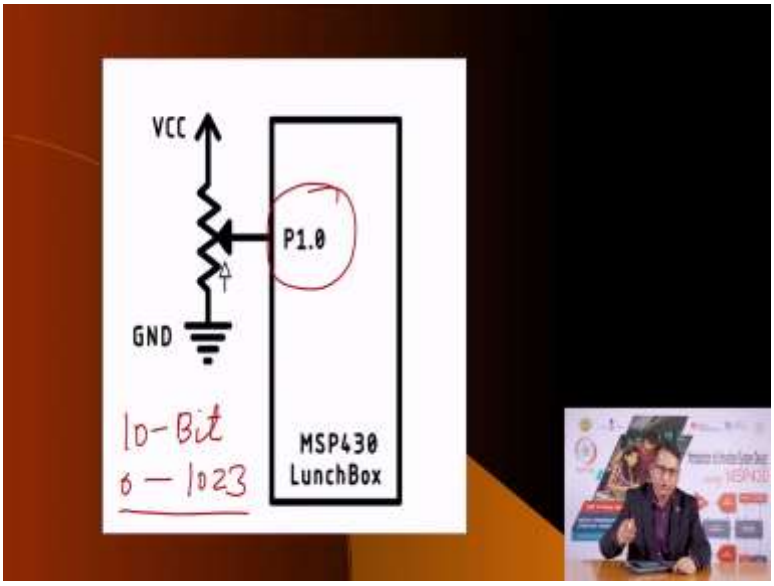
Now, is the time to actually integrate all these features that we are now aware of and to be able to program like a pro, alright. And so, the topic of this lecture is how to code like a ninja. What we have probably not covered enough and we want to address that here is the mechanisms of related to mathematical manipulation of information, of data.

Till now our manipulation was simple. We were adding numbers, we were doing bit manipulations and, you know, simple stuff. But what if we wanted to do more mathematically involved calculations? Like we wanted to calculate the logarithm of the numbers or exponential of the numbers and so on. Is a microcontroller capable of doing that? Yes, it is.

Ofcourse, if you were to program the microcontroller in assembly language, it would be a big challenge but because we are programming our embedded systems, our microcontrollers in C doing all these operations is quite a breeze. As in there is no, not much challenge in terms of writing program. Ofcourse, it takes a lot of time when the program is calculating complex mathematical operations.

It takes time and so we must be aware of that. And so, the beginning of the exercise is how to read information from the outside world, how to process it with more complex mathematical operations and how to see the result of those operations.

(Refer Slide Time: 02:57)



And so, in the first example, what we are going to do is, here is the setup. Maybe it is shown here. We are going to read the analog value of this potentiometer which is 10 kilo ohm potentiometer connected to P1.0 and it is going to read it. And as we know, we have already dealt with the ADC. The ADC on MSP430 is a 10-bit ADC. And therefore, the number will range from 0 to 1023.

Oftentimes, you do not need such a large variation in the number. You would like to represent this range with the smaller values of numbers. And one way to do that is to take a logarithm of the of this number. And so, in this exercise what we are going to do is we are going to read the ADC as frequently as we can. And once we read number, we would like to calculate the logarithm of this number.

Now because the number is between 0 to 1000 or so the numbers will range from 0 to 3. Eventually, in the second part of this program, a second variation of this program we are going to display that number on our local display. In which case it is Charlieplexed LED display. We have already covered Charlieplexing in the previous exercise. So, you are aware of what is Charlieplexing.

This is a form of multiplexing which allows you to control a large number of, relatively large number of LEDs using relatively smaller number of pins. And so, we are going to rig up on the breadboard we are going to rig up a Charlieplexed display with 6 LCDs. And for 6 LEDs you only need 3 pins. So, using 3 pins and 6 LEDs we will display the numbers calculated from this exercise, taking the logarithm.

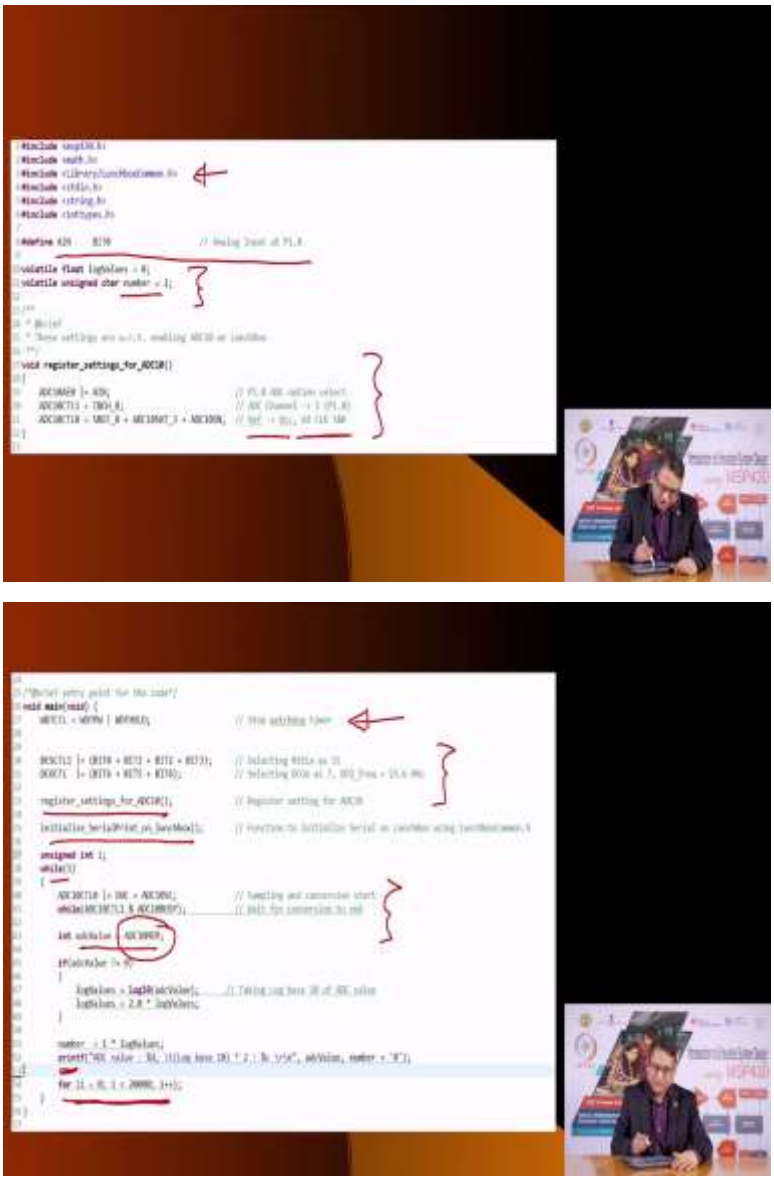
But because the numbers are going to be in the range of 0 to 3 and we have 6 LEDs, we will multiply it by 2 and then display it on the Charlieplexed LED display. In the first part of this exercise, we are not going to do anything like that. We are going to calculate the log and then using the serial print mechanism.

You may recall that in one of the earlier lectures when we were talking of the gate and CCS and we talked about various aspects related to embedded C programming, we introduced to you a black-box approach of including a certain lines of code which basically initialize the UART to communicate with the outside world at the rate of 9600 bits per second.

And you could use that mechanism to print information as long as your microcontroller is connected to a computer, and the computer is running some terminal emulation program. The information which is being sent from the microcontroller could be monitored on the PC. And so, this is very useful when you are developing your program. You want to see what numbers are being calculated.

And since your microcontroller does not have elaborate display features, you could use the display of your laptop or desktop computer to monitor those numbers. So, in the first exercise, we are going to read the value of the potentiometer, take a log of that, multiply it by 2 and then print these numbers onto the serial monitor.

(Refer Slide Time: 06:17)



Here is the program. Now, we are saying that code like a ninja. Ofcourse, it would also mean that the program becomes longer. And so, probably the programs that we cover in this lecture here are by far the longest programs in terms of the size also. And so, here what we are doing is we are defining that we have a potentiometer connected to the bit 0 that is P1.0. We have declared two volatile variables.

One which will maintain the log of the value and the other which is a number which is been initialized to 1. Then we have function where we initialize the ADC that convert the value,

connect the ADC to channel 0 that is P1.0. And then use the VCC as a reference and using clock frequency 64 clock pulses per conversion for the sample and hold.

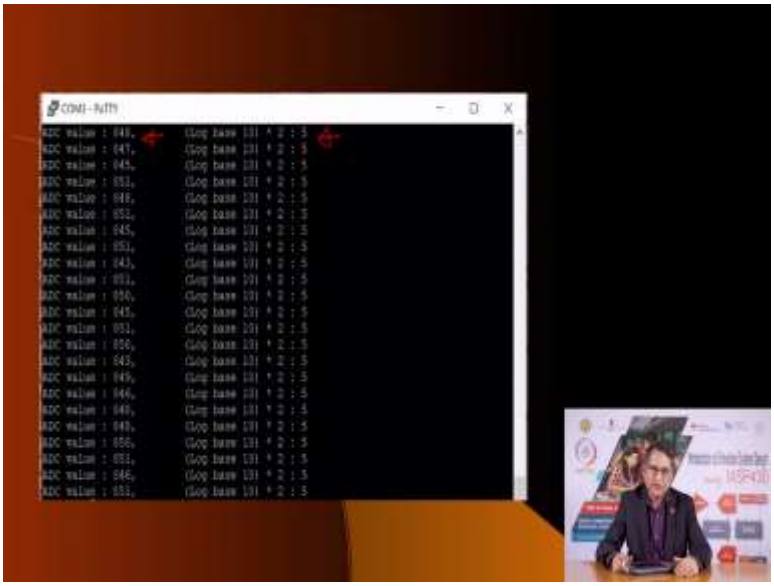
Then this is the main part of the program. We have just stopped the watchdog timer. We have also; although we have not done this often in our previous programming exercises, we have increased the clock frequency internally generated DCO to operate at the highest possible clock frequency. Here in this case about 16 megahertz. And why?

Because we are doing mathematical calculations. Mathematical calculations using the built-in functions take a lot of time; log and exponential and so on. And so, if we did not increase the clock frequency it would slow down the entire processing time. It would take more time to process that information and so we wanted to reduce that and so we increased the clock frequency.

Then we are you calling this subroutine as we saw to set the various values for the ADC. Then we initialize the serial printer and you know this function is defined in our lunchbox common dot h file, as you may recall from a previous exercise. And then we enter this infinite loop where while 1 what we do is we start the conversion, read the value from this memory location.

Once the conversion is over, the ADC writes the value in into this memory location ADC10MEM. We have transferred it into the ADC value. And now based on the ADC values, we are processing that information. And then just transferring that to the printf which actually invokes the serial print. And then this is simply a delay value.

(Refer Slide Time: 08:59)



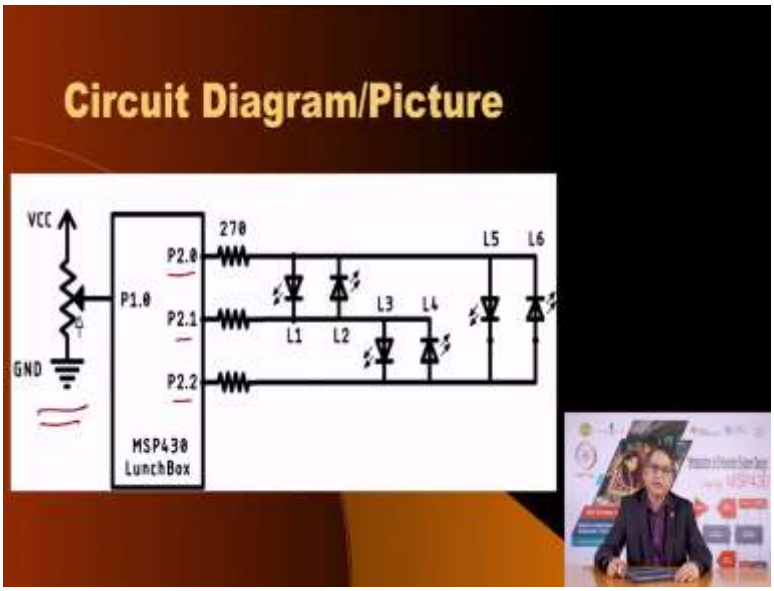
Now when you run this program, you would see that based on this it prints the ADC value, also the log and also the converted value. And you see, as long as whatever the number you can verify that using a calculator what should be the log 10 of this number into 2? Is it matching these numbers here?

Ofcourse, you are not been able to take a larger variety of these numbers. But please verify when you compile this program, rebuild and download after making appropriate connections. In this case, there are not many connections to be made. The lunchbox itself is quite sufficient. All you have to do is connect a potentiometer.

One part of the potentiometer to Vcc, the other extreme to ground and the centre point to the P1.0 pin. And when you run this program, you can verify that the ADC value will be printed and also the converted value.

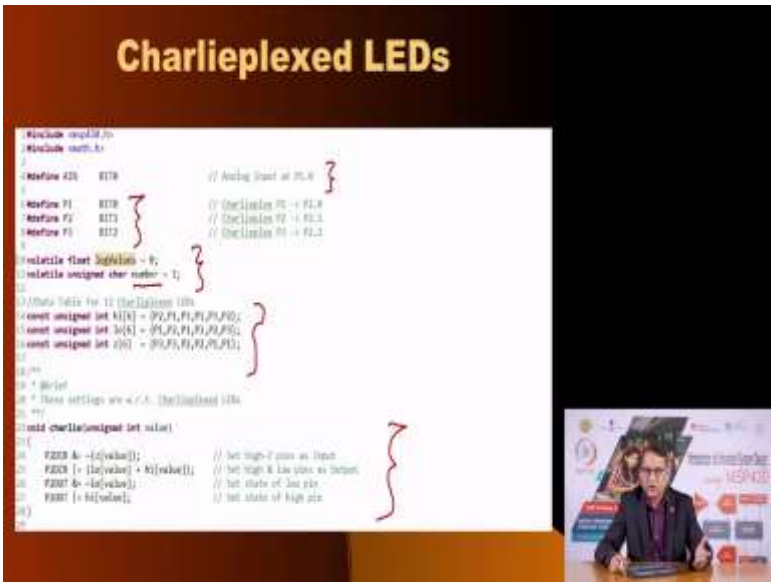
In the second part of this exercise where the conversion and all is not changed. But now instead of displaying the result on the serial monitor we would display the numbers on a Charlieplexed display. And for that we use certain pins as we see here. There is the Charlieplexed display. Here is the whole block diagram in a way of the exercise, the circuit that will use for the exercise.

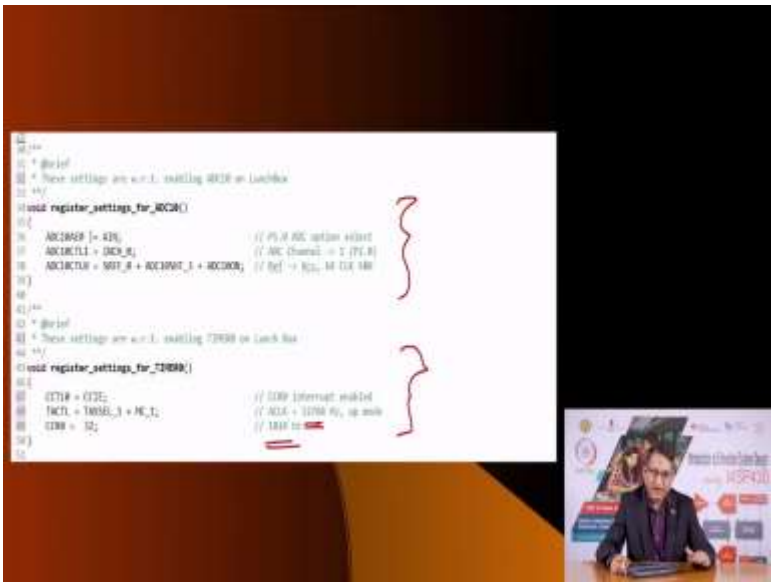
(Refer Slide Time: 10:15)



We are going to connect a potentiometer to P1.0 as earlier. But to P2.0, 2.1 and 2.2 we are going to connect 3 LEDs; 3 pins we are going to connect 6 LEDs in the Charlieplexed display fashion. And these displays, these LEDs will be handled in a interrupt subroutine. And we will see how that happens.

(Refer Slide Time: 10:38)





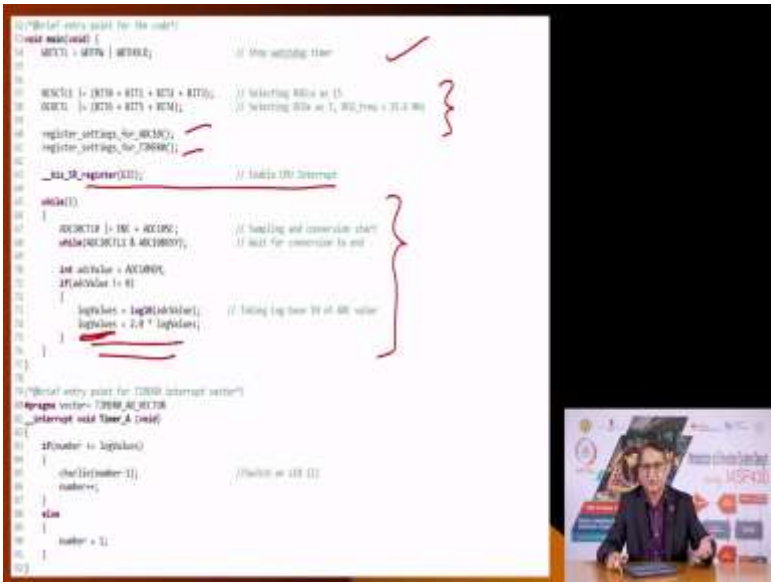
We have already seen how interrupt programs work and so I am not going to go through that. The important point is that these 3 pins are defined here for the display. Here is the pin definition for the potentiometer. Here are 2 variables: one which contains the log value and the other which is a number which is going to go from 1 to 6 and so on.

Here are 3 arrays into which the port values are stored to respond to correspond to the Charlieplexed display that when a particular LED is to be turned on, the pins that connect to that LED have to appropriately turn the LED on. That means the anode has to be high; the cathode has to be low, and the other pin has to be set in the high impedance mode.

And the way to do that is to program that pin which connects to that LED, the third pin to operate in as an input pin with no pull up or pull down resistor. It would make it high impedance. And so basically these 3 arrays deal with that. Here is a subroutine called Charlie which based on the number to be displayed, it appropriately sets the direction of the ports and converts the third pin whichever that third pin might be for that number, into high impedance.

Here is the register setting for the ADC. It is similar to what we did previously. Here is the register setting for timer. Now, this is an additional subroutine into this program because we are using the timer to generate an interrupt at roughly 1 kilohertz using the internal 32 kilohertz clock. By dividing it appropriately, we are generating a timer interrupt every 1000 times a second that is at a rate of 1 millisecond. So, the Charlieplexed LEDs will be refreshed 1000 times a second.

(Refer Slide Time: 12:34)



Here is the main code where we have disabled the watchdog timer. We have changed the frequency to about 16 megahertz and we have called the subroutines to program the ADC and the timer, and we also enable the interrupt. Now, why we have enabled the interrupt? Is that when the timer expired, it will generate an interrupt and go into the interrupt subroutine where it will refresh the LEDs and come back.

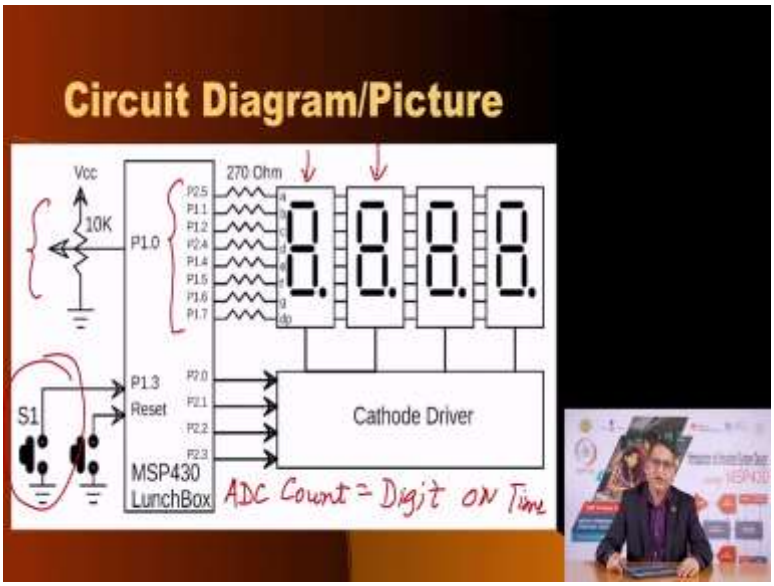
And then we enter infinite loop where we are reading the ADC. Then we copy the value of the ADC from the memory location and then perform the log calculation and then store the number result into these log values.

And from there, what it does is the interrupt goes into the interrupt subroutine; the program goes into the interrupt subroutine whenever the timer expires and the information is converted to from log it is converted into appropriate numbers and displayed on the Charlieplexed display.

So, I strongly recommend to you that go through this program, understand what is happening and see how the internal functions the available functions on the code composer studio for MSP430 allow you to manipulate data mathematically using complex functions. Now, that was the objective of this exercise.

Now we are coming to the part two of this program; part two of this lecture where we are going to deal with more elaborate scenario, where we have created a mechanism where multiple activities are taking place and how does a microcontroller look at multiple activities. We are going to illustrate this requirement or illustrate this need in 2 methods. Of course, the first method is to illustrate the shortcoming of that method and that we are only doing it to convey to you that this is not the method of handling information in that way and that the better method is to use the interrupts.

(Refer Slide Time: 14:42)



So, what is the objective of our exercise? The objective of our exercises is that we have created a 4-digit display. Now, how do you connect four 7 segment digits to a microcontroller with limited number of pins is that you employ what is called as multiplexing. And so in multiplexing instead of each digit having its own independent seven or eight pins, we are sharing the pins.

And so instead of if you are to connect four 7 segment displays, you would require 32 input output pins. And we do not have 32 output pins on our microcontroller. And so instead of connecting one 7 segments to its own private 8 pins, we have shared the pins using this technique called multiplexing, where the segment's A, B, C, D, E, F, G, and decimal point, that is eight segments are connected to pins of the lunchbox in this fashion. You see, because of some of the pins being allocated for certain functions, there is no sort of a sequence in those pins, but that is alright our program can handle that.

And then we are turning one digit on, displaying the information there. Then we are turning the turning this display off, then displaying the second digit and so on so forth. And when we read the last digit after that we come back to the first digit. And if we did this exercise fast enough, then fast enough to beat the shortcoming of the human eye that is the persistence of vision. If we change these displays or we multiplexed these displays faster than the persistence of vision limit of the human eye, you would notice, you would see that all these displays are ON at the same time.

While we are multiplexing the display, what do we want to do? We want to count the number of times the switch connected to P1.3 is pressed. So essentially the objective of this exercise is count the number of times that switch S1 connected to P1.3 has been pressed. Every time the switch is pressed and released; it should increment an internal counter. And because we have a 4-digit display, the counter we have programmed to go from 0000 to 9999. If you press more than that then it will reset to 0000 again, and that process will continue.

Now the objective of this exercise is not just to count and display but also to illustrate this wonderful concept called multiplexing, and how the rate of multiplexing should be high enough, so that the human eye in the normal circumstances is able to see all the digits at the same time. But in reality, is that what is happening? No.

In reality, what is happening is that only one digit is on at any given time. But it is changed so fast that the human eye is unable to follow that change. And so, what we are doing with this exercise is not only counting the times that the switch is pressed and displaying the count on the digit but we are also controlling the multiplexing rate. And for that we are inputting that information through a potentiometer.

And so, obviously, as a designer, if you are going to implement this system, you have to have an idea as to how are you going to change the multiplexing rate using an external input. And so, we have connected a potentiometer to the ADC input. And when the way we have connected when the centre tap of the potentiometer is connected to the ground pin, it will the ADC will read a value of 0. When the centre tap is connected to the other extreme, it will read the maximum count which in this case will be 1023. So, using a count which varies from 0 to roughly 1000, we want to use this information to vary the rate of multiplexing. And so, one way could be that forget the 0 part. That is when the value read by the ADC is 0.

Let us say that the value read by the ADC is 1 to 1023. We could map these values directly in terms of the time for which any given displays on at any given time. So, for example, if the setting of the ADC is such that the ADC values 1, one way to deal with that would be that 1 means the count means the time or let me rewrite it. Let that the ADC count is equal to digit; any given digit ON time.

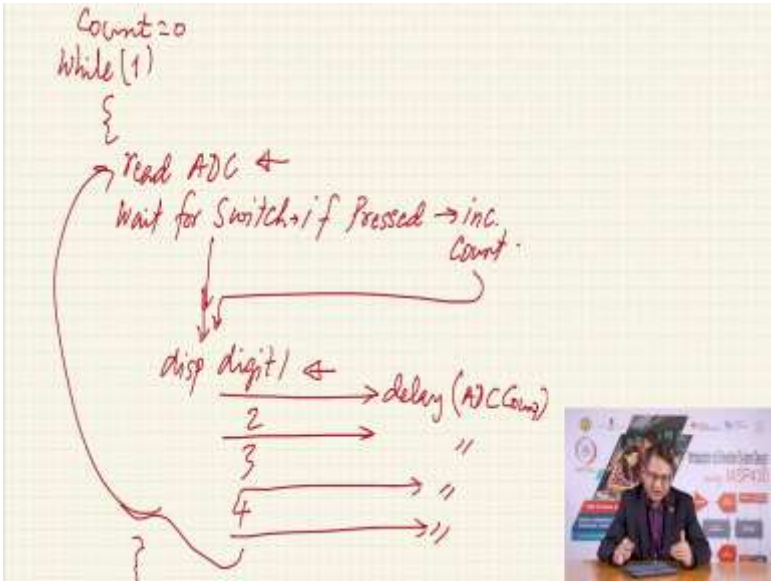
So, if the ADC count is 1 that would mean that each digit will be on for 1 millisecond. And so, one millisecond, one millisecond, one millisecond, one millisecond to the last digit again it will come back. And of course, this is a very high rate of change and my eye will be unable to follow that change. So, at this extreme of the ADC value, the display will be very crisp. You will be able to see all the 4 digits at the same time. If you press the switch at any given point of time (switch S1), it will count and update the count and you will see that on the display.

Now, how do you handle such? So, there are 3 activities going on if you see. One activity relates to reading the ADC value and using the count result of the ADC to change the rate of refresh. Meaning if the ADC value is 1, the time for each display is one millisecond. As this ADC value increases, let us say the time is 10 milliseconds. That means each digit is now ON for 10 milliseconds.

So, the total time cycle time is 10 plus 10 plus 10 plus 10, which is 40 milliseconds. This would give you a refresh rate of about 25 hertz. Now if you keep on increasing it even more, so you go from 1 count of ADC of 1 to count of let us say 1000. And we since we are saying that the count is directly translated in terms of milliseconds, a count of 1000 from the ADC would mean that the time for each digit is 1000 milliseconds. That is 1 second.

And so now you would be able to see each digit being turned ON for 1 second. Next second, the second digit, third digit, four digit and so on. And so, you would not see all the 4 digits ON at the same time; you would actually see each digit, only one digit ON at a time. However, even so, it would the microcontroller would still be able to count or should be able to count the switch being pressed and released and the changed count to be reflected on the display. Now, how do we deal with such a requirement?

(Refer Slide Time: 21:38)



And so, the first part of the program is to deal with this in a sequential fashion. How? So we write infinite loop like this. While 1 and then in that loop we say read ADC; read the appropriate channel of ADC. Then read wait for switch to be pressed. Now, what if the switch is not pressed? If the switch is not pressed of course, you still have to display whatever the last value of the count.

And therefore, you can say all right, nobody has pressed the switch. Let me if the switch is, if pressed, if pressed, wait for it to be released and then increment count, increment count. If it is not pressed you come here directly. If it is pressed then you come after it is incremented. And now what do you do? Break that count into 4 digits. Display digit 1. Then display digit 2, 3, 4 and then here your loop terminates which means you are going to now go back, read ADC and so on.

Now see what is going to happen? The time for which this display each digit is ON will be determined by the ADC value. And so, suppose the ADC value resulted in 1 millisecond. Of course, I need to add here delay appropriate count say delay based on the value of the ADC count. And so, you are going to repeat this here, here and also here. And then you will go back into this loop, go to read the ADC again.

So, the ADC count is 1 millisecond. After you read it, you are going to wait for the switch to be pressed. Suppose the switch is not pressed. Fine. Let us say the count before that and so which

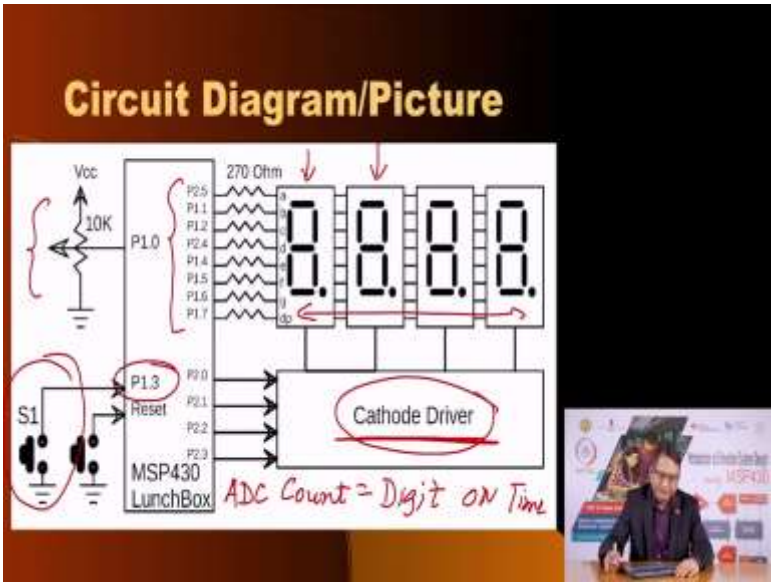
means before the count and say count is equal to 0. So, you are going to display 0000. And once you have displayed all the digits, you come back. Now let us say I do not touch, so the ADC is still 1.

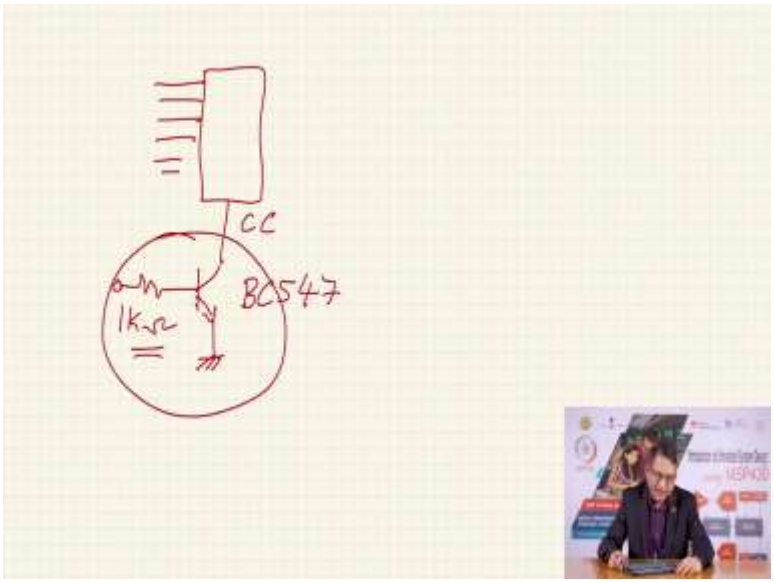
Now I go for the switch and now I press the switch. Now if I press the switch, I am going to wait for the switch to be released. And suppose I keep the switch press for a very long time. What you would see when you compile and download this program is that for the time that I keep the switch pressed, the display is going to vanish, all the digits will be off. Of course, when you release, the count would be updated and you would see them on the 4 digits. But it certainly shows that yes, it is, it appears that it is possible to do multiple things in an infinite loop. Each activity takes a certain amount of time, then you go to the second activity, third and so on. And if you did it fast enough, it is possible to keep doing all the activities so that everyone is happy.

But here is a situation where somebody. some part of that repeated act repeated activity takes hogs all the time of the microcontroller, then it is going to deprive the other activities and, in this case, the other activities are to turn the 4 digits ON one after the other. The processor, the switch press activity is going to deprive the display multiplexing and therefore, you will see the display is black-off.

And so of course, this is just to illustrate that this is not a very good idea of programming. And so, yet I would like you to understand this program where essentially it is doing what I mentioned here. It is reading the value of the ADC. Based on that it is after that is going to read the switch, wait for the switch to be pressed. If it is not pressed, is going to go to the displaying the values. I want you to understand this code, compile it and download it into your MSP430 kit. And of course, the lunchbox would need to be connected to these 4 digits. So, you need a breadboard and so on.

(Refer Slide Time: 26:05)





You would also need; let me go back here to the circuit, the block diagram. So, you will need these 4 digits and in the cathode driver here, you are going to need to connect NPN transistors. And the circuit diagram for the those NPN transistors will be something like this. So, suppose this is your digit with all the ABCD. Here is the common part which we call the CC. You need to connect the NPN transistor like this to ground and the base of this you can connect to an appropriate, I would say roughly 1 kiloohm resistor.

And this would be coming from say P2.0, 2.1. So, you are going to need 4 NPN transistors and appropriate transistors could be BC547 and so on; general purpose NPN transistors are enough

and four 1 kiloohm resistors. And this you are going to connect one such combination for each digit. Since we have 4 digits, you are going to need 4 such setups. And so that will form the part of the block diagram here called the cathode driver.

Now let us keep the same circuit diagram. Let us not change anything on our breadboard and the way the connections are made from the lunch box and recall that the switch which is connected to P1.3 is on the lunch box. So, you do not need to connect any other switch on your breadboard. The only thing you need to connect on the breadboard are these 4 digits, these 8 resistors for the 7 segments, the 7 segments of the 7-segment display and the cathode driver in the form of 4 NPN transistors and 4 resistors in series connected to P2.0, 2.1, 2.2 and 2.3.

Once you rig this circuit up, breadboard it and then connect it connect the lunchbox to your computer. And this code, you download and execute. I want you to experiment with it. I want you to play with that press of the switch and see by changing the potentiometer the refreshing of the display changes from the one extreme where you see all the 4 digits at the same time to other extreme where you are seeing one digit at a time.

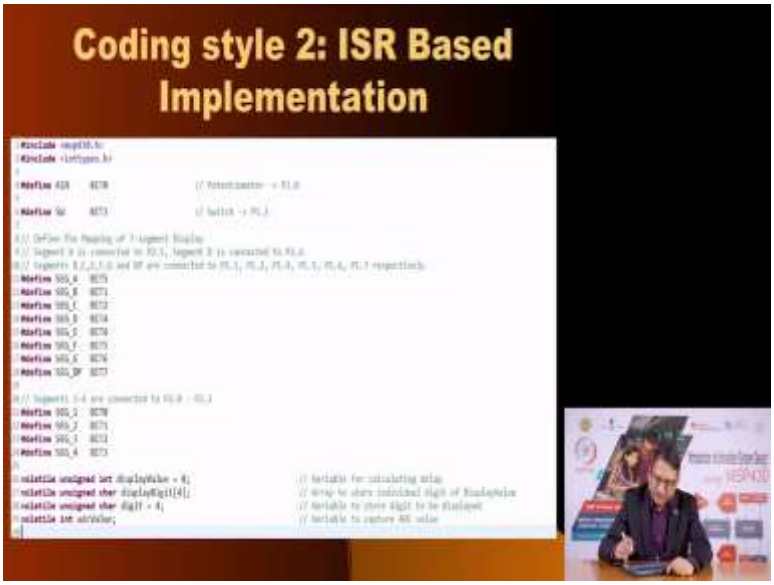
But even so, even if the display or you seem to, it appears that all the 4 digits are ON at the same time, the moment you press the switch, you see that there is the displays are going OFF. For such duration that you keep the switch pressed. And of course, therefore, this is not a professional method of doing multiple things at the same time. Multiple things being interfacing to a switch and connecting to 7-segment displays.

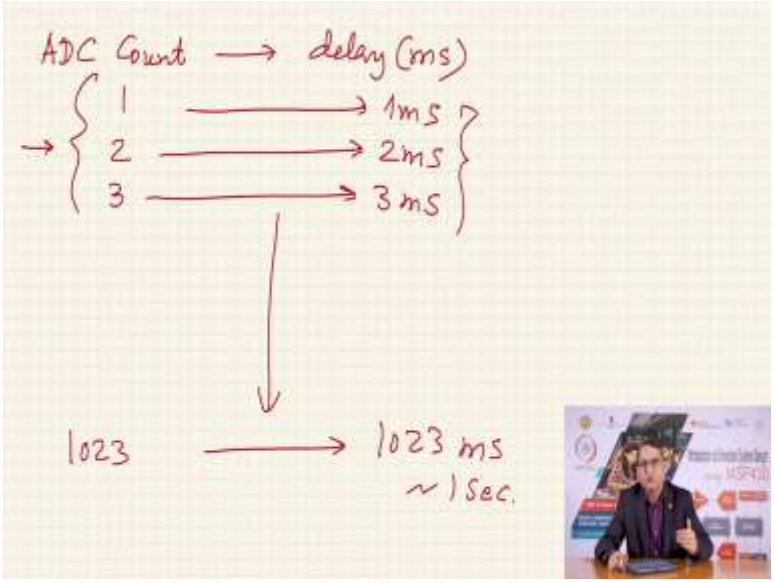
And so, what we are going to do is we are going to offer certain, some changes to this programming style. This one we call a sequential execution because you are doing all these things one after the other. Now of course, the microcontroller has only 1 CPU and therefore, it can only do one thing at a time. But imagine that we do all these things quickly without starving the other activities.

And we can do that if we incorporate the concept of interrupts in this system. If we used interrupts, then you would see suddenly the entire mechanism, the way the system works is far, far better. That there is no display blanking happening. You can parallelly change the rate of refresh using that potentiometer. And at the same time, be pressing the switch as frequently or infrequently that you wanted and you would see that the displays are reflecting the count based

on the number of presses that you perform on that switch. And therefore, that is the appropriate method of programming and I want to talk about it.

(Refer Slide Time: 29:57)



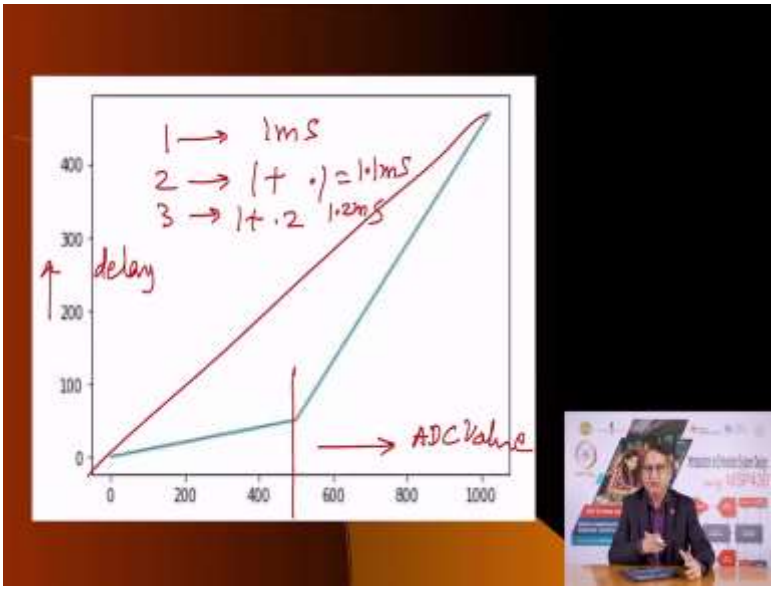


But before that, I want to also talk about that this whole mapping of ADC value directly to a delay value is not a very good idea. And let me explain what I mean by that. We said the ADC count is directly delay time in milliseconds. That means if the ADC count is 1, it means 1 millisecond. If the ADC count is 2, it means 2 milliseconds. If the ADC count is 3, it means 3 milliseconds.

And what is this delay going to do? It is going to turn a given digit ON for this amount of time. And so, from 3 if I go down to highest value of ADC, let say 1023 that means each digit will be on for 1023 milliseconds which is roughly 1 second. Now, if you see at the lower end of the ADC count, you are going to refresh at 1 millisecond. That means the total refresh rate will be 1 kilohertz corresponding to 1 millisecond divided by 4 (because there are 4 digits). So, you are doing 250 times a second for the entire display.

But the moment you increase the count to 2, the display rate changes drops drastically from 250 times a second to 125 times a second. And when you do 3 milliseconds, it is actually 3 into 4, 12 milliseconds. That means you are going roughly to about 80 times a second. And so, a very slight variation of the potentiometer will quickly go through the entire combinations of a 250 hertz, 125 hertz and 80 hertz and so. And you see there is a big jump from 250 hertz you are suddenly going to 125.

(Refer Slide Time: 32:18)



I would like to vary the refresh rate more gently by going from say 250 hertz to 220 hertz to 200 hertz and so on. So, one way would be to apply some non-linearity; some mapping of the ADC values in a non-linear fashion, so that I can have a different amount of delay. And one method could be as illustrated in this graph is that here is on this x-axis is the ADC value. And on the y-axis, you have the delay time.

Now, if this was a linear relationship, you would actually see something like this. And of course, the count here would be to 1023. But what we did was, we set, if the value of the ADC is less than 500, then the new value is the actual value of delay is the value of the ADC into 10 percent of that value.

So, a value of 1 will give you 1 millisecond. The value of 2 will be 1 plus difference that is now 1 into 10 percent. So, it is 0.1. So therefore, you will get a 1.1 millisecond delay. 3 will give you 1 plus 2 into point 10 percent of it; so, 0.2. So, you will now get a 1.2 millisecond delay and this we this calculation we perform till the count is less than 500. The moment it becomes more than 500, we suddenly increase the that percentage amount. Instead of being 10 percent, we increase it to 80 percent and now therefore, it is able to increase more rapidly.

Now, if you use this, you would see that when you use the potentiometer to change the refresh rate that you can actually see the refreshing of the displays changing gently from one extreme where all the 4 digits are ON at the same time, you are able to gently reduce the refreshing rate and you will actually start seeing individual digits.

You will first see flicker and then the flicker will increase and then you will actually see each digit being turned ON one after the other. This happens even when you are running this part of the code that is the sequential style of programming; you will see that happening there also. But it will be more prominent and more appreciable when you change the style of programming.

So, instead of sequential programming, you can do what is called an interrupt subroutine based implementation. Now which means we can look at all the events that are happening. And what are the events that we have? We have essentially 3 events: one is to read the ADC value and based on the ADC value decide the duration of time for which each digit is going to be turned ON. And for that we will invoke that function to convert from ADC value into delay value. The second is to be able to read the switch. Whenever it is pressed, wait for it to be pressed. Then wait for it to be released.

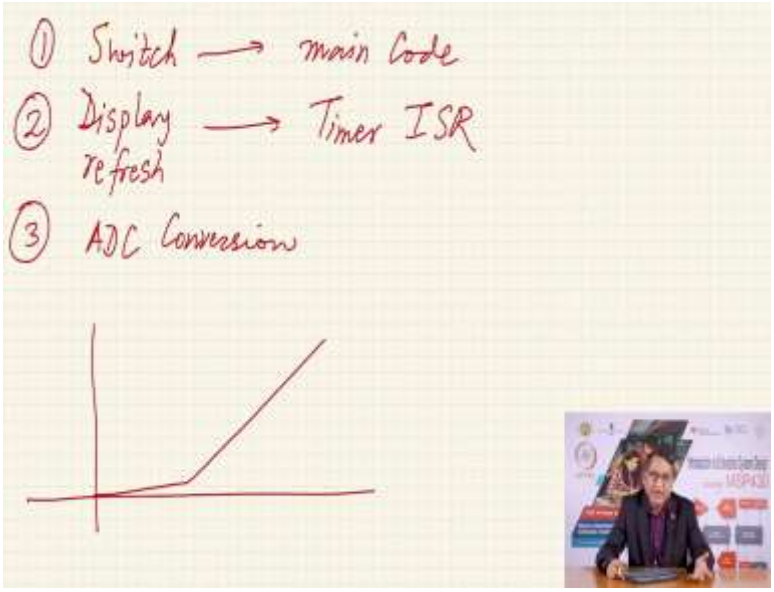
And of course, as you know, when you press a switch there is there is a potential of bounce. So, we have to debounce it and we can debounce it either using a hardware method that is using a capacitor, which I do not recommend at all or implement a delay function.

Now, the problem with implementing a delay function with a switch is that you cannot handle the activity of switch press as a subroutine because as we have discussed in the past, interrupt sub routines have to be kept very short. And you cannot include delay functions in interrupt subroutines because then it would delay the interrupts subroutine to last that much longer.

And then it would deprive the other parallel activities parallel in, you know, in a different way. In that the other contemporary activities would be deprived of the CPU attention. And so, we do not recommend at all invoking delay functions in interrupts subroutine.

So, what options do we have? We can deal with the debouncing of the switch as part of the main code. And we could relegate, we could offload the activity of refreshing of the display to an interrupt subroutine and the reading of the ADC also to a subroutine. This way, we would be able to implement a much smoother implementation, one in which even if you press the switch as long as you wanted, it would not affect the refresh rate. The refreshing will continue to happen as determined by the potentiometer.

(Refer Slide Time: 36:51)



And so, what we have done is we have in this program in this sequential style, let me go back. This is the, this is the interrupt style. What we have done is we have divided the multiple tasks into various ways that they will be handled. We have done switch. The switch as part of main code. The display refresh as part of timer interrupt, timer ISR. Of course, the rate of the interrupt generated by the timer is now not constant. The rate is now determined by a time which is dictated by the potentiometer value. And so, we have a third; so, this is activity number 1, this is activity number 2 and the activity number 3 is an ADC conversion.

An ADC conversion, once it is converted it implements that non-linearity that is it implements a function to find out what should be the actual delay value and it stores it. It uses this resultant value to program the timer. So, by changing the ADC, you are able to affect the interrupt rate generated by the timer. And therefore, you would see that the multiplexing of the displays is going from one digit ON which you can perceive to all the digits that appeared to be ON at the same time. Of course, they are not ON at the same time but they are being turned ON and OFF very fast for very short durations of time.

And while this is happening, in the background because they are being handled in the interrupt subroutine, the main program can very happily read the switch, wait for it to be pressed, wait for it to be released. While it is being pressed, it will debounce it. While it is being released it is debounced and then only it will update the count. And this count will be as we have discussed,

this count is what is to be displayed on the digits. And so, this count must be a variable which is of the type volatile.

(Refer Slide Time: 39:10)

And so, I strongly recommend to you that based on this discussion based on this description of how the code is designed, I recommend to you to go through this code. The code has been nicely partitioned into several subroutines and 2 interrupt subroutines and the main code in which the switch is being looked at. And the count is incremented based on that.

And then the count is actually a memory location is being stored in the memory location, which is of the type volatile so that the interrupts subroutines, especially the interrupt subroutine of the timer interrupt will be able to pick the count.

And based on the value that has to be currently displayed, from that count it will be able to display exactly that count on that digit and it will happen smoothly. And so, I strongly recommend that based on this description that you go through this code, understand what we are doing and implemented and you can try your ideas.

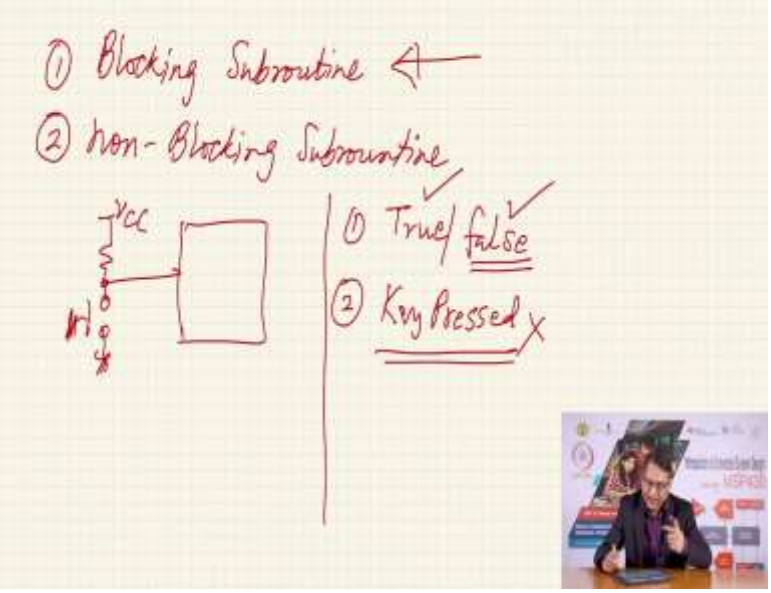
For example, would you like to implement the reading of the switch in the interrupt and doing the display refresh in the main code? You try doing that and you will clearly see that if you did not debounce the switch in the main subroutine in the interrupt code. And then if you applied a capacitor it would lead to an increase in component count. If you did not apply a capacitor for

hardware debouncing and instead included delay subroutine in the interrupt ISR for the switch, you would see how much delayed is the refresh rate.

So I recommend to you that you first implemented this code. As you have downloaded, build it and download it into your lunchbox and see the performance, and then play with it. Do you want to do the ADC conversion in the main loop and handle the display in interrupt and switch also in another interrupt? That is also possible. Or you want to do switch and ADC in interrupt and you want to handle display in the main program. All those permutation and combination exists.

And with experience you will realise that many of these things can be kept short and they can be relegated or they can be performed as a part of an interrupt subroutine and that would lead to a much better code implementation. That would lead to very crisp display where no count is being lost. No switch being pressed is being lost. The displays are managed nicely. And with this exercise we hope once you understand the code, you would become a better programmer.

(Refer Slide Time: 41:55)



There is one more aspect that I wanted to; I am not going to go through this code. Because by this time you have become quite good with programming. We have just increased the level of programming through this exercise. I want to discuss a little bit more here that when you write a subroutine; there are actually couple of 2 methods of writing a subroutine. One, that we call as blocking subroutine and the other that we call as non-blocking subroutine. Although we are not explicitly discussed this in our lectures till now. But this is a good time to talk about it.

Now, in this the programming exercises that we covered here, the reading of the switch was in the main program. But you could have as well offloaded it to a subroutine. And if you offload it to a subroutine, you could write the subroutine in these 2 fashions. You could write the reading of the switch as a blocking subroutine or you could write it as a non-blocking subroutine.

In the blocking subroutine what are you going to do? As the name suggests, it blocks; meaning when you call this subroutine it is going to wait till that intended action happens. If the action does not happen it is not going to go back. Which means, if this is my switch, alright? And this is pulled up with a pull-up register and this is going to a particular port pin and I call a subroutine to read the switch. Which means I want to return to the main program once the switch is pressed and released, to indicate that the switch was pressed and released.

The subroutine would have only one return value. In this case that the switched is pressed. If on the other hand, I do not press the switch. What is it going to do? It is never going to go back to

the main program. And so, in that way this subroutine is going to block the program in the main code. Of course, if anything was happening at the interrupt level, interrupt will not be able to will not be blocked. It will go, service the interrupt subroutine and come back into this blocking subroutine.

On the other hand, if I program the same subroutine in the non-blocking format. What do I do? I say call a subroutine to read the switch. Now because it is non-blocking, you are going for the switch to be pressed for some time. If the switch is not pressed in the default time that you defined, your subroutine is going to go back to the main program. But how would the main program know what is the reason for the return from the subroutine?

So in the case of non-blocking subroutine there will be 2 return values. One will be TRUE or FALSE and the other will be to tell you that the key was pressed, key pressed. Right? Now, in the case of non-blocking subroutine if the key was not pressed. The first variable will be set to FALSE; first return value would be set to FALSE. And, if the value if set to FALSE, you are not going to look at the second variable. And in this case actually make sense.

If instead of a single switch there were many switches. And many switches mean one of the switches is expected to be pressed. So, you would go back not only to say that the switch was pressed but also which switch was pressed. So, you would return back with the code of the switch that was pressed. And if there are say n switches, there will be n codes.

If this subroutine was written as a blocking subroutine, it will have only one return value which is the value of the switch that was pressed. That is the code of the switch that was pressed. And if subroutine is written in blocking format, it is going to block the rest of the execution of all subroutines and main program.

If it on the other hand, it is written in non-blocking, now the return values are different now. You have 2 return values. One which tells are you returning because some action happened or are you returning because there was a timeout. Which means for each of such subroutines, you have to also define a parameter called as a timeout period. Each timeout period will depend on the type of subroutine, the type of activity you want to do in this.

And I would like to bring to your attention things that are in embedded applications where the subroutines are written either in blocking mode or non-blocking subroutine format. Let us say, an

ATM. As we discussed earlier, ATM machine is a great example of an embedded application. You as a user when you go and insert your card into the ATM machine, the ATM machine responds by saying, “Please enter your pin”.

Now what if you did not enter your pin? What does; how does the ATM machine responds? If the subroutine for reading the switch matrix which has number from 0 to 9 and so on. If somebody had programmed that ATM machine to read the matrix of switches as a blocking function, the machine would have blocked. Why?

It is asking you to enter the pin number and you are refusing to enter the pin. So, it is going to block the operation. But what actually happens? If you have seen an ATM machine operation, if you enter your insert your card and you do not enter the pin number for some time, the machine simply ejects your card saying, “Timeout”. Which indicated to you that whoever, the embedded designer who design ATM machine was a smart fellow.

He or she programmed the switch matrix and wrote a subroutine of a non-blocking function. So if you did not pressed the pin withing a stipulated amount of time, it will go back to the main program with the return value set to FALSE meaning there was timeout; nobody pressed the switch and the main subroutine will not look at the second variable.

On the other hand, if you press the switch within the timeout period, the first variable written value will be set to TRUE and the second variable will be set to the code of the switch of the key that was pressed. And so please remember whenever you have an option, whenever you have a freedom to write a subroutine you actually have 2 options. You can write the subroutine in blocking format or you can write it in the non-blocking format.

With this we come to the end of this exercise. We come to the end of this lecture where we have looked at various programming aspect with the view of making you a better embedded programmer. I hope with all the sessions that we have had till now, you feel happy that you know much about embedded system. You know so many features of embedded microcontrollers and you are now able to program more efficiently.

And we would culminate, we would integrate all this knowledge that you have gained by having a lecture later on and illustrate how this knowledge could be integrated together to implemented

a complete stand-alone project based on MSP430. So, I will see you very soon in the new lecture. Thank you very much and bye-bye.