

Introduction to Embedded System Design
Professor Dhananjay V. Gadre
Netaji Subhas University of Technology, New Delhi
Lecture - 23
MSP430 Digital I/O: Switch Interfacing

Hello and welcome to a new session. I am your instructor, Dhananjay Gadre, and I am conducting the session for the online course on Introduction to Embedded System Design. In the previous lecture, we connected the MSP430 microcontroller using the MSP430 lunchbox to LEDs, and we saw how we could toggle the LEDs or we could turn the LEDs on and off; we called those program HelloLED or HelloToggle. Now in this session, we are going to consider input devices and the most common input device is a switch. So let us see how we can interface an MSP430 to switches.

A switch is a mechanical device it consists of two metal plates with the shorting cover over it, which when the user presses the plastic button, the cover, metal cover inside shorts the two contacts of the switch; when the user releases it, the cover retracts and the connection is broken. So using this arrangement, the contacts of the switch toggle between or move between two states, switch is open or switch is closed. To ensure that the switch retracts there is usually a spring connection, spring arrangement.

(Refer Slide Time: 01:54)

Digital Switch Interfacing

Possible ways for a switch connection :

- Pull up configuration ✓
- Pull down configuration ✓

open & Close

↓ ↓

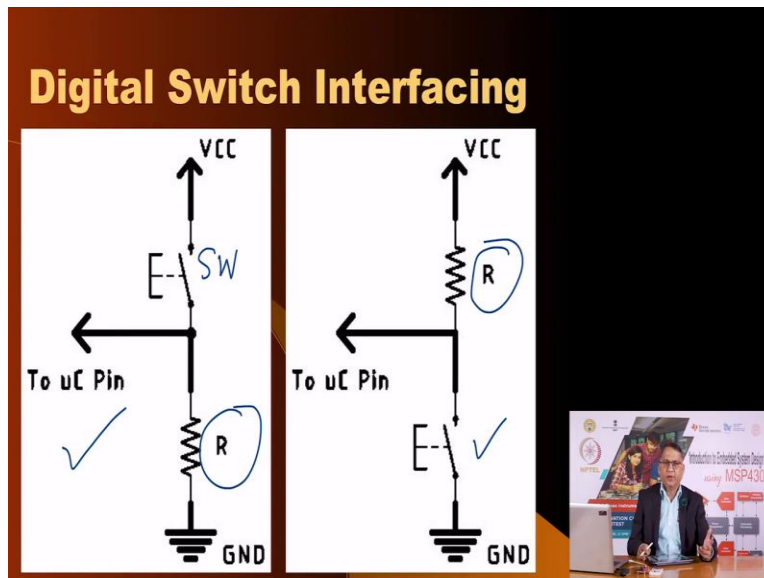
1 0

0 1

Now because of this, the switch will often bounce as we have discussed, but before that, the switch has two states; open and close. And these states we have to translate into logic 1, if we

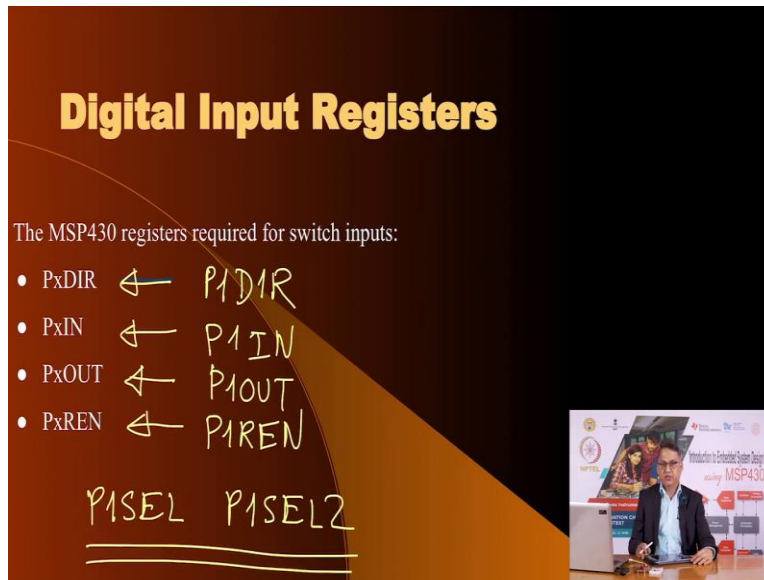
choose logic 1 for open, then the close must translate to 0 or we can have another possibility that we can have open translate to logic 0 and then the close will translate to logic 1. And this is achieved by using a pull-up resistor configuration or pull-down resistor configuration. Let us see how these configurations look like.

(Refer Slide Time: 02:33)



Here is the pull-down configuration. As you see here, we have, this is the mechanical switch, this is the pull-down resistor, and in the second configuration we have the switch here and the pull-up resistor and the connection between the resistor and the switch in both the cases is going to a microcontroller pin, in this case, some MSP430 pin which can be configured as input.

(Refer Slide Time: 03:05)



Now let us look at what resistors to engage, what resistor to program so that we can connect a switch on a input-output pin, configure the pin as input and without having to use external resistors, invoke the internal facility, internal feature to either use a pull-up function or use a pull-down function. And for that, we have four registers that is PxDIR, which means if we are talking about port 1, it will become P1DIR then we need to, so we need to configure P1DIR, appropriate bit of P1DIR, where we want to connect a switch as input.

Then we need to be able to read P1IN because this will give us information that what is the state of our switch, whether it is 1 or 0. If we want to engage internal pull-up or pull-down resistors, then we have to enable that using this function, using this resistor and whether we want to have a pull-up or pull-down will be exercised by writing appropriate value of 0 or 1 in the appropriate bit in the P1OUT register, P1OUT and this is P1REN.

So this is the registers that we need to deal about. There are other registers that is P1SEL and P1SEL2 but these are only utilized when we want to invoke alternative functions on a given pin, and since we have decided that we want to use this pin as input, we are not going to be bothered by these two registers, we will not use them in our program. Now to read a switch we will not read a single bit, we have connected the switch to one of the pins, which means to one of the bits, but when we read the port we will get all the 8 bits and we will have to then isolate that bit so

that we can take a decision is the bit sending a 0 or is the bit sending a 1, and we can choose to implement a particular function if the bit is 0, what to do; if the bit is 1, what to do.

(Refer Slide Time: 05:53)

Reading inputs

Suppose there is a switch connected to P1.3 in a pull up state (As is the case in LunchBox), and we need to read the value of the pin.

Here is a snippet of code example for testing individual bits in a register:

```
if ((P1IN & BIT3) == 0)
    // to be done when P1.3 = 0
else
    // to be done when P1.3 = 1
```

00001000
↑

The slide also features a circuit diagram of a switch connected to pin P1.3, with a pull-up resistor to VCC and a pull-down resistor to ground. A small inset video shows a person at a desk with a computer monitor displaying a presentation.

Now in our case we have, on our lunchbox, we have a switch available on the lunchbox and it has been connected to P1.3 and an external resistor has been used in the pull-up state and therefore, we should ensure that in our program it takes cognizance of this feature that a switch is connected to P1.3 pull-up resistor has been used and so the program must appropriately recognize this configuration.

So to be able to identify whether the switch is pressed, when the switch is not pressed it will give a logic 1 because there is a pull-up resistor, here is how it is. Here is the switch connected to ground, here is the pull-up resistor, and the junction is going to P1.3. So when the switch is not pressed, it will give you logic 1; when the switch is pressed, it will give you logic 0. So I am, by reading P1IN and then ANDing it with bit 3, bit 3 nothing but a mechanism to identify and isolate bit 3, its value is 00001000 why this is bit 0, bit 1, bit 2, and bit 3.

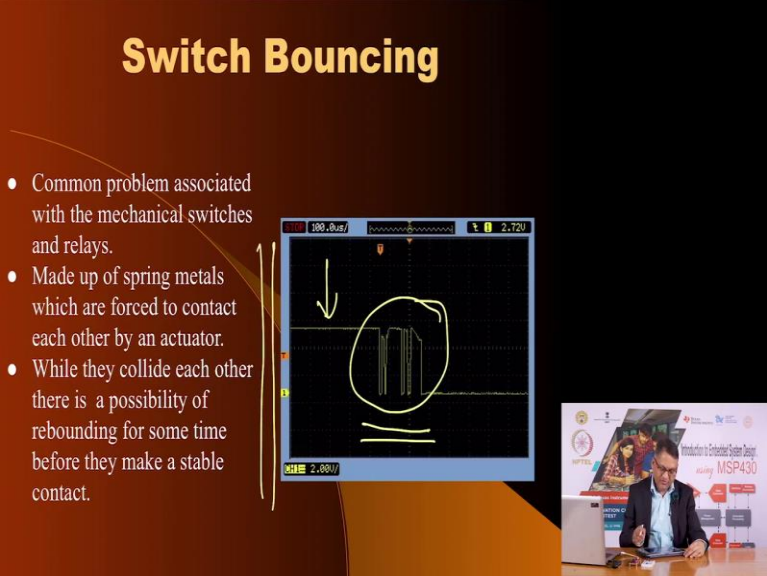
So on bit 3, there is a 1, this is a mask, mask value, and by ANDing whatever has been read from port 1 IN, by ANDing it with this you will get binary result, either the result is 0, but because this bit 1, the third bit is 1, if the result is 0, the ANDing operation, bit-wise ANDing operation leads to 0 that means the bit 3 which is read from the pin has reported a 0. If it is 1, the ANDing will become 1, so if this is equal to 0, then do something whatever you want to do if the switch has

been pressed, else if it is not pressed, it will return a 1 and then you can do something else. This is the basic logic by which we detect that a switch is pressed or not pressed and we can choose to do one or the other activities in this program.

(Refer Slide Time: 08:06)

Switch Bouncing

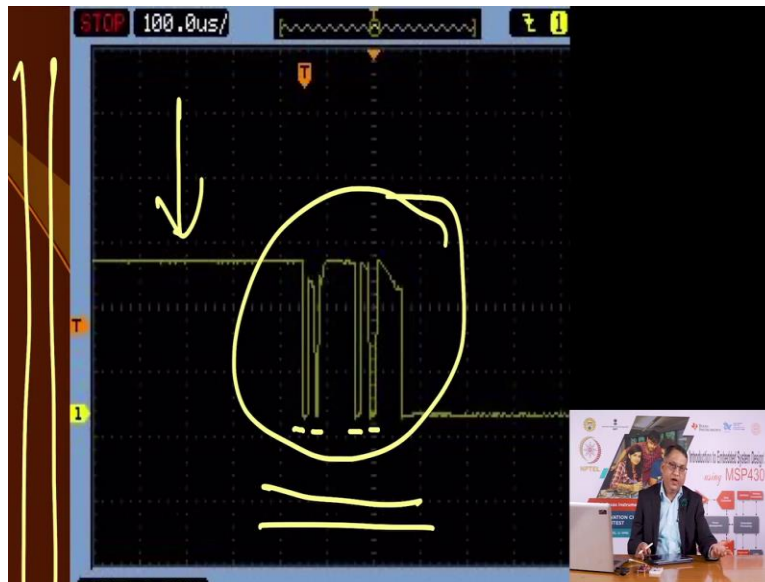
- Common problem associated with the mechanical switches and relays.
- Made up of spring metals which are forced to contact each other by an actuator.
- While they collide each other there is a possibility of rebounding for some time before they make a stable contact.



Now, because mechanical switches will exhibit bouncing and here is the recap of what happens, when the switch is not pressed you will get a logic 1, when the switch is pressed you might observe a phenomenon like this. This has been captured on a digital storage oscilloscope, you could try recording such a event at your own end. However, it does not mean that every switch press will leads to so many ups and downs. In your case they may be more, in it is also possible that you will get even less bounce, it is also possible you do not get bounce, any bounce at all.

So do not get alarmed this is an extreme situation and we have taken several recordings on our oscilloscope just to identify a situation where it shows several bounces and the idea is to convey that switch, switches do bounce and this an extreme situation, it is possible that you do not get any bounce. And as mentioned here, the bouncing is because of the mechanical nature of the switch, spring, and it would have fluctuations and that is why you get bounce.

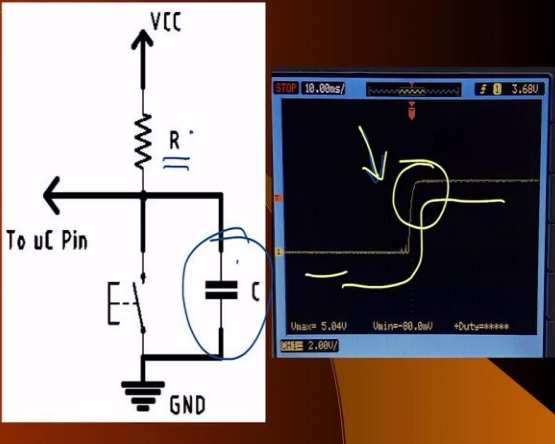
(Refer Slide Time: 09:31)



Now the question is we cannot have a bounce in our program, because the bouncing would lead to being interpreted that the, even though the switch was pressed once if you see here, even though the switch was pressed only once but if the program does not remove this bouncing, it will conclude that it was pressed once, twice, thrice, four, five times and it might do five things but we know that it was pressed only once, so we have to find a mechanism to remove these fluctuations.

(Refer Slide Time: 09:56)


Hardware Debouncing



The diagram shows a circuit for hardware debouncing. It consists of a pull-up resistor R connected to VCC and a switch connected to GND . The output of the switch is connected to a microcontroller pin (labeled "To uC Pin") and a capacitor C connected to GND . The oscilloscope trace shows a signal that is initially high, then drops to low when the switch is pressed, and exhibits a characteristic "bouncing" behavior before settling to a stable low state. The oscilloscope settings are 10.00ns/div and 3.68V.

Switch Bouncing

- Common problem associated with the mechanical switches and relays.
- Made up of spring metals which are forced to contact each other by an actuator.
- While they collide each other there is a possibility of rebounding for some time before they make a stable contact.



The oscilloscope trace shows a signal that is initially high, then drops to low when the switch is pressed, and exhibits a characteristic "bouncing" behavior before settling to a stable low state. The oscilloscope settings are 100.0ns/div and 2.72V.

And one method is to use a hardware debounce technique and a simple capacitor here of a suitable value, which must take into account the value of the resistor, the pull-up resistor so that R and C makes a low-pass filter. And so if we know that our switch bounces say, in this case as you see the couple of 100 microseconds. So if you have the low-pass filter frequency to be much lower than 1 by, in these case, 1 by 300 microseconds whatever is the resultant frequency, if you choose the, if you select the value of the R and C such that the resultant frequency is much lesser than that, then this will be able to debounce the switch in hardware.

However, I do not recommend this solution. Why? Because in real application, such a capacitor would add to the cost. If you have 10 switches, it would mean having to use 10 capacitors. And the reason is that these capacitors can be easily avoided by using an alternative method, which is called software debouncing. But here the capture on a DSO shows that now the transition is much smoother, there is no bounce on it, and because the capacitor you see this charging sort of feature on the transition of the voltage from 0 to 1.

So let us see what is the software debouncing. In the case of software debouncing, we simply, once we detect that there is a transition of the input of the pin that at which the switch is connected, we simply stop looking at that pin for a certain period of time.

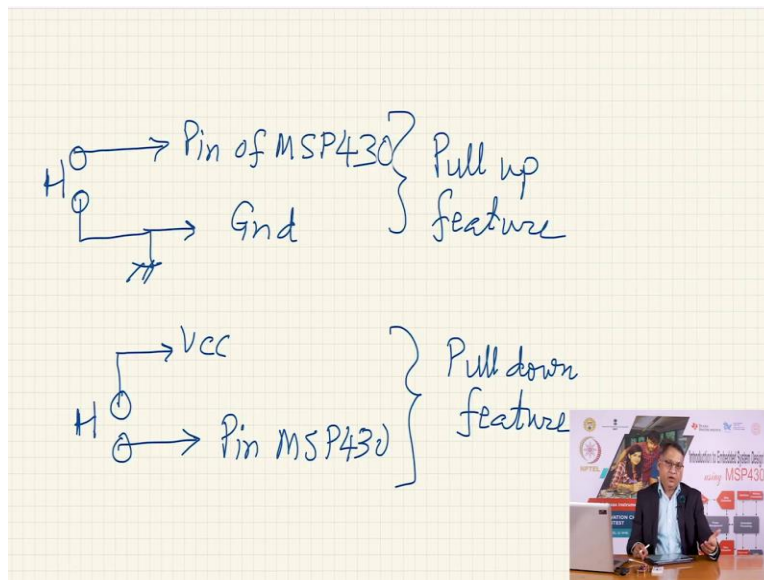
(Refer Slide Time: 11:32)

The slide is titled "Software Debouncing" in a large, bold, yellow font. Below the title, there is a bullet point: "• Provide a delay of ~20 milliseconds after the first change is detected." To the right of the text, there are two horizontal lines representing a delay period. Below these lines, the handwritten text "200µs" and "1ms" is visible. In the bottom right corner of the slide, there is a small inset image of a man sitting at a desk with a laptop, with a presentation board behind him that includes the text "Introduction to Embedded System Design using MSP430".

And for example, we could use 20 milliseconds delay, but you can alter this to be of different values. For example, if you know your switch only bounces for about 200 microseconds, then you could decide to use a 1 millisecond debounce delay period and you can play with this to get an optimum value.

In our examples that we will show subsequently just after this, we have used a 20 milliseconds delay and let us now move over to our hardware configuration, where we have this laptop, we have also connected the MSP430 lunchbox. There is switch on the lunchbox already connected to P1.3, but we want to illustrate that how we could utilize the internal pull-up and pull-down resistor configurations.

(Refer Slide Time: 12:36)



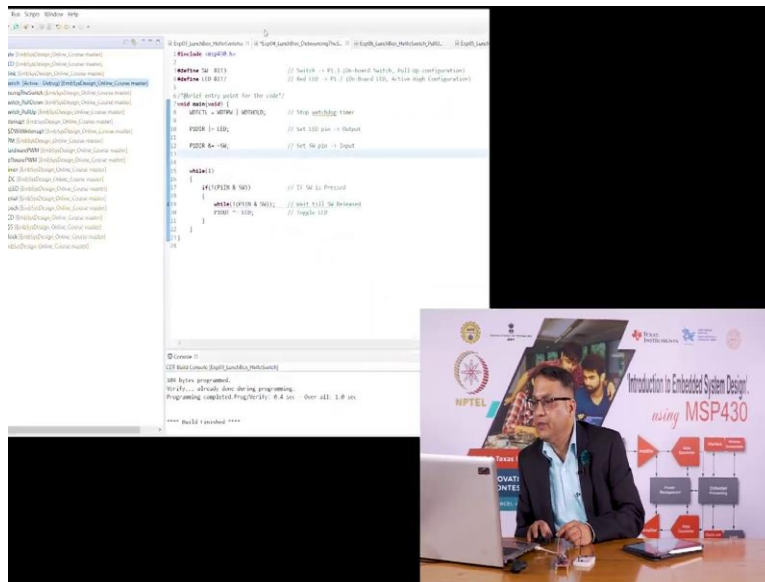
We have a breadboard here, on this breadboard there are two switches which have been configured in the pull-up and pull-down configurations, meaning we have a switch in one case where the switch is connected. Two terminal of the switch are available here. If we want to use in the pull-down mode, we ground it. So now these two connections are available, we will connect it to an appropriate pin, pin of MSP430 and this will be connected to ground.

We have another switch in which case the switch, now for this configuration we have to invoke the pull-up feature, we have another switch here where the switch is connected to VCC, here, and the other pin is available here and this will go to VCC of the MSP430, this will go to another pin of MSP430, and this configuration allows us to test the pull-down feature. So we have, we will in, test several versions of the program, one with no debouncing either in hardware and software, and then you can observe what happens with the result of that switch, switch press.

In other case we will involve a hardware debounce, then we will add a software debounce, using the onboard switch on the lunchbox. Then we will have two versions of the program where we use this breadboard, we will use jumper wires to connect to one switch and then the other switch, one which is pull-up feature, and the other which uses the pull-down feature.

So make sure that your hardware is ready and you follow the program examples to test these experiments. Here we are going to connect switches in various configurations and I am going to show you five versions of the program, I have four versions of the program.

(Refer Slide Time: 14:49)



The first version is, I am going to use the onboard switch, as you know this switch is connected to the bit P1.3, and if you look at the code which you can, which I am sure you have downloaded, you will see that the switch is connected to bit 3, the LED onboard of MSP430 lunchbox is already connected on bit 7, in a previous exercise we had seen, experimented with this LED and we had turned it on and we had toggled it. So it is the same LED configuration.

Then, now we come to the main code part of the program, here the first instruction talks about stopping the watchdog timer. We talked about it yesterday also that the default state after reset is, one, the watchdog timer is turned on; two, the master clock frequency is 1.1 megahertz. So our program is running at 1.1 megahertz, we are not changing anything else, 1.1 megahertz is okay for our experiments, so we will continue with that.

Now in this, we have, so as I mentioned we have to first determine the direction of our pin, our pin, in this case, is connected to the switch is connected to P1.3, our LED is connected to P1.7. So we have to determine and dictate the direction of these two pins that is bit 3 and bit 7. The first instruction after turning off the watchdog timer turns the P1 direction on bit 7 so that it is 1, and by making that pin 1, it becomes an output pin.

And the second instruction talks about, so the first instruction is P1 direction resistor is equal to whatever value of P1 direction resistor is ORed with a constant here which is actually a masking value of LED, LED is bit 7, that means the most significant bit is 1, the rest are all 0, when we

OR it, in P1DIR bit 7, you get a 1. And when you send it out to P1DIR resistor that bit becomes 1 that means the, that particular bit becomes output.

In the second instruction here P1DIR, you are saying P1DIR is equal to P1DIR ANDed with the inverted value of the mask constant called SW. SW is on bit 3 and therefore, essentially it will make the P1.3 bit in P1DIR resistor is equal to 0. This will make the pin as input. Now once these two instructions are executed, we are ready, we have decided, we have converted P1.7 as output and P1.3 as input. And these are all on board that is on the lunchbox, switch is connected on P1.3, and LED, user LED is connected on P1.7.

Then the next instruction is a infinite loop which says, while 1. While 1 means you enter the program and then you are not going to exit from this program. In this program, we are waiting for the switch to be pressed, meaning if the switch on pin 1.3 is not pressed, it is not going to enter the second loop and it is going to wait there, but the moment you find that the switch is pressed, you will get a logic 0. Why? Because in this configuration we have used an external pull-up resistor.

So when the switch is not pressed, you will get a 1, the moment the switch is pressed the logic inside the if-statement here will become 0, invert of that will become 1. So if this condition is true, that condition is true, when the switch is pressed you are going to enter the if-loop. In that you are again waiting for the switch to be released because you press it, usually, human interaction means you are going to press the switch for much longer duration and so this will hold the state of that pin to 0, you are waiting for that to be released. That is the second instruction.

When you release the switch, the switch, the P1.3 bit will become 1, and then at that time it will come out of that while-loop, the inside while-loop and it will toggle the LED. P1OUT is equal to the value to the LED invert of that, ANDed with the invert of that. So it is going to toggle the LED value and now once you compile this program, you rebuild the project. This will be compiled and then downloaded. Yes, now it is ready.

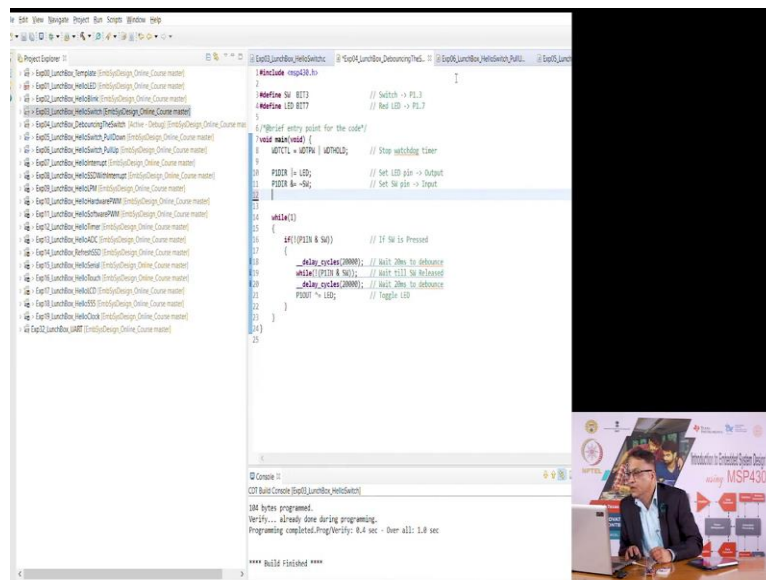
Now when I press this and release you will see that most of the time it toggles, correct, but sometimes it misses. That is even if I press and release, the LED state does not toggle from 0 to 1 or 1 to 0, sometimes it remains. So I recommend that you play with it to realize that this problem

exist because you have allowed the switch bounce to propagate through the program and therefore sometimes it would miss, sometimes it will not be able to toggle in a way that you expect it.

Now you can re-run this program, you can continue experimenting this program by connecting a capacitor between the switch and the ground and we recommend a value of about 0.1 microfarad of ceramic capacitor between P1.3 pin and ground and repeat this, and you would find that the performance as far as toggling the LED is concerned, might improve, would improve. So I am not doing it here but I am recommend you try that, the same program by connecting a capacitor. This will do hardware debouncing.

But as I mentioned, in embedded applications our aim is the cost and cost means less components, which means we should avoid using an external component as much as possible, try to utilize it through that functionality, try to gain that functionality by using software. So we are going to avoid using a capacitor and instead we are going to involve in software debouncing.

(Refer Slide Time: 21:35)



```
#include msp430.h
#define SW BIT3 // Switch -> P1.3
#define LED BIT7 // Red LED -> P1.7

/*Global entry point for the code*/
void main(void)
{
    WDTCTL = WDTPA | WDTCLD; // Stop watchdog timer
    P1DIR |= LED; // Set LED pin -> Output
    P1DIR &= ~SW; // Set SW pin -> Input
}

while(1)
{
    if(P1IN & SW) // If SW is Pressed
    {
        __delay_cycles(20000); // Wait 20ms to debounce
        __delay_cycles(20000); // Wait 20ms to debounce
        P1OUT ^= LED; // Toggle LED
    }
}
```

Console:
C:\Build\Code\Proj01_Lumibo_HelloWorld
104 Bytes programmed.
Verify... already done during programming.
Programming completed.Prog.Verify: 8.4 sec - Over all: 1.8 sec
**** Build Finished ****

So the second program that I want to show you here is a variation of this program but with a small modification that once the switch is detected to be pressed, I will call a delay subroutine of appropriate duration, once the switch is released, again I will call a delay subroutine and then I will toggle the LED. So the rest of the instructions are the same, you still have the while 1 loop,

you are waiting for the switch to be pressed, when a switch is pressed you will get a logic 0 in that operation, invert of that you will get a 1.

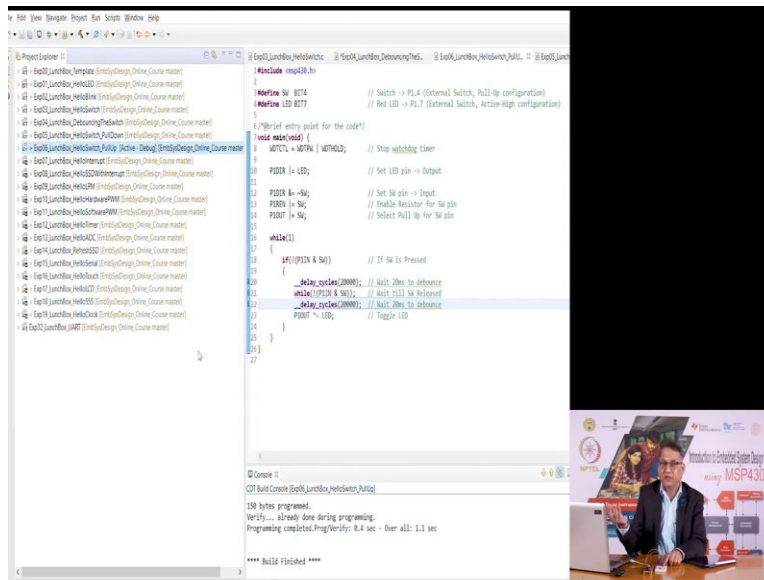
So if-condition becomes true, the moment you get inside there is a software delay of 20 milliseconds. This is a built-in function, you are saying delay for 20000 instructions that leads to about 20 milliseconds delay and then you wait for the switch to be released. This has debounced the part where you have pressed the switch, now you are waiting for the switch to be released, that is the first while-loop inside that if-loop. When you release the switch, again it delays, same 20 milliseconds debounce, and once you finish, once you complete that 20 milliseconds delay, you toggle the LED. And you are going to be in this if-loop which is inside a infinite while-loop.

Now let me go back to the first program that we tested in the previous lecture, where we showed that our code has a main-loop and we just either turn the LED on or we toggle the LED. Now I mentioned that that is an infinite loop, although there was no while 1 kind of a loop.

Now you see the reason why I mentioned it is, even though main program has only one instruction, the question you have to ask is what does the microcontroller do when that instruction in that main loop is over? You see the microcontroller have all the features, all the necessary, necessary ecosystem which supports continued execution of program, even though you have not provided an another instruction to execute, does the microcontroller stop or does it continue to execute a program? The answer is it continues to execute a program where it keeps on waiting, it keeps, goes in a infinite loop doing nothing, which means the microcontroller is still working, except that it is not doing any useful stuff.

So a main program is still an infinite loop, in this case, it is explicit, in the case of our switch experiments it is explicit because we have a, within the main loop we have a while 1 infinite loop. But even if it was not there, when the program is compiled, it will still continue to work like that.

(Refer Slide Time: 24:33)



Let us now go to the second, third version of our program, where we are not going to use an onboard switch. Instead, we are going to use external switch, we have a switch where we are going to enable the internal pull-up resistor or internal pull-down resistor connected to a different pin.

Now if you see this program this is called HelloSwitch with the option of pull-up resistor. And now the switch definition has changed, instead of the switch being on bit 3, it is on bit 4. The LED is still on bit 7, which means we are going to use the onboard LED which is on the MSP430 lunchbox. Our first instruction after we enter the main loop is still to stop the watchdog timer and now we have created an instruction where we have turned the bit on port 1, which is connected to the LED as output.

We have turned the switch which, the pin which is connected to the switch, in this case, P1.4 as input. But besides that, we have two more instructions, one says P1REN is equal to P1REN ORed with the value of switch.

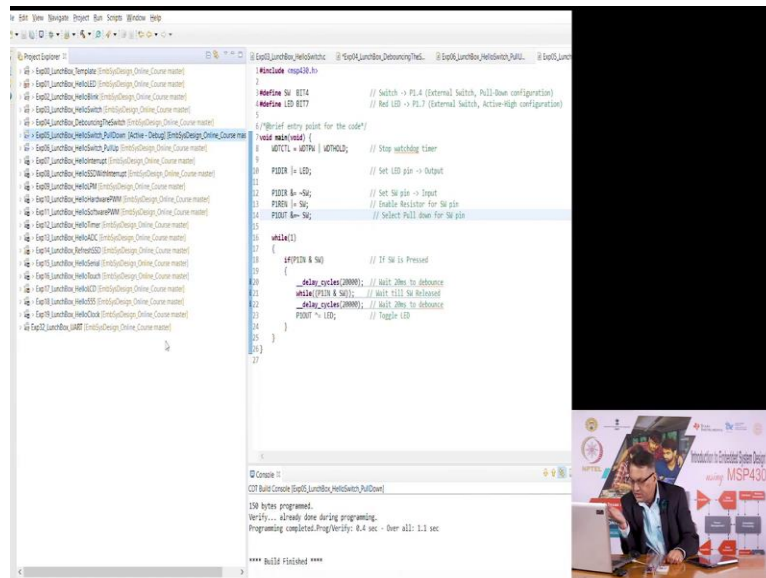
So we have enabled the internal pull-up feature with this instruction and the next instruction is P1OUT is equal to P1OUT ORed with the switch mask. What are we doing? We are making that P1OUT on port bit 1.4 equal to 1. What does this do? This enables the pull-up resistor. So I have connected the, I will connect the appropriate switch in which it is connected to the pull-up function and we will re-compile this program, here this wire, there are two wires, one is to

ground and VCC and the third wire is to an appropriate switch. In this case, the switch is such that it is a pull-up configuration. So we are going to use the internal pull-up feature.

I am going to, here the program is compiled and it has been downloaded. Now let me press this switch and you see the switch is toggling nicely, no external resistors has been used. Why? Because an internal pull-up resistor was used with debouncing, it also has the same debounce logic.

So we have, we have the same program compared to the earlier version of debouncing, we have just added a feature that we have switched the switch, moved the switch from P1.3 to P1.4 by connecting an external switch, the switch is configured in the pull-up configuration and we have used an internal pull-up resistor, nothing else changes. So now when I have compiled and downloaded the program, what I find is that the switch toggles nicely, the LED toggles nicely each time I press this switch, which is an external switch with the internal pull-up resistor connected, each time I press and release the switch, the LED on the board toggles nicely.

(Refer Slide Time: 28:19)



Now let us look at the fourth version of the program, which will use the same switch except now the switch is in the pull-down, it will use a pull-down resistor. And the initial programs are all, initial instructions are all same, except when we come to the point of enabling the pull-down value of the resistor. Here you see P1OUT is equal to P1OUT and invert of SW, so now that P1OUT on which pin? P1.4 is set to 0, this will enable the pull-down resistor. So I am now going

to change the input from this switch, which was configured for pull-up to another switch, which is configured for pull-down.

I am going to compile this program and download it by rebuilding this project. As you see on my laptop the program compiles and now it has downloaded, build is finished, the program is downloaded. Now I am going to press the other switch and now you see here also the switch is debounced internally because of the software.

If you see the, inside the while 1 loop, we have 20 milliseconds delays, we are waiting for the switch to be pressed and released. The only difference is now we are waiting because we have a pull-down resistor, the value is 0 when the switch is not pressed. So it is waiting for the switch to be pressed so it becomes 1, till it is 0 it is going to wait there. And then when it is released, again delay and then it will toggle the LED.

So I recommend that you play with all these four versions of the program. In the first version, there is no debouncing and you should be able to see that there is false toggling. It can improve when you connect an external capacitors of 0.1 microfarad and then the rest of the programs are all going to debounce the switch. The second version of the program uses the onboard switch, and the third and fourth version of the program uses external switch with internal pull-up or pull-down resistor as the case is both these versions.

So I hope this clarifies you the enormous capabilities of MSP430 microcontroller of offering internal pull-up and pull-down resistors, which you can invoke when you are connecting external mechanical devices such as a switch and this would we would use in subsequent programs and experiments of creating interrupts and things like that. So I will meet you in a new lecture very soon. Thank you very much, buh-bye.