Introduction to Embedded System Design

Professor Dhananjay V. Gadre
Netaji Subhas University of Technology, New Delhi

Lecture 27

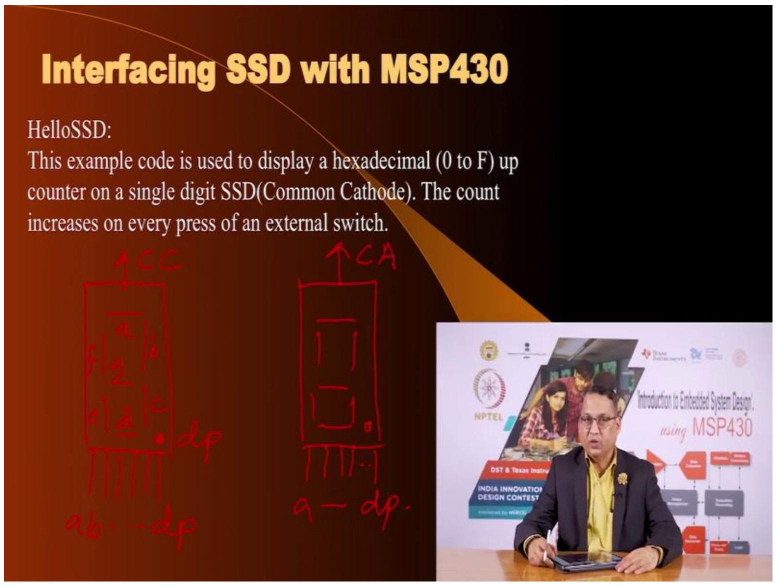
Interfacing Seven Segment Displays with MSP430; Low Power Modes in MSP430

Hello, and welcome to a new session, for the online course on introduction to Embedded System Design. In this session, we are going to look at two aspects, one is to increase our capability of the microcontroller to interact with the outside world through interactions, including displaying numbers. Till now, we have seen how we can use the micro controller to turn LEDs on and off, but an LED can only give you binary information, something is on, something is off. But if you would like the microcontroller to show you numbers or other information, we would have to use a display device, such as a 7 segment display.

Of course, we can also use a liquid crystal display and other graphics displays which we will come to, in a later lecture. But in this lecture, we are going to see how we can connect a single 7 segment display. For the reason that it is easy to connect a single 7 segment display or a couple of 7 segment displays to a microcontroller. But as the number of 7 segment displays increases, it would not be possible to connect to large number of 7 segment displays or large number of LEDs to a microcontroller without increasing the complexity from the software side.

Right now, we are going to look at how we can connect a single 7 segment digit display. Now, as you know, in our electronic component inventory, which we had shared with you earlier, and I hope you are able to get your hands on a four-digit 7 segment display. We are you going to use the same four-digit 7 segment display in our experiment but we are only going to enable only one of the digits. In a subsequent lecture, we will show how we can control a four digit or for that matter, a larger digit, larger number of digits display. But let us begin with our interactions here today, on how to display information on a single 7 segment LED display.

(Refer Slide Time: 02:34)



Interfacing SSD with MSP430

HelloSSD:
This example code is used to display a hexadecimal (0 to F) up counter on a single digit SSD(Common Cathode). The count increases on every press of an external switch.

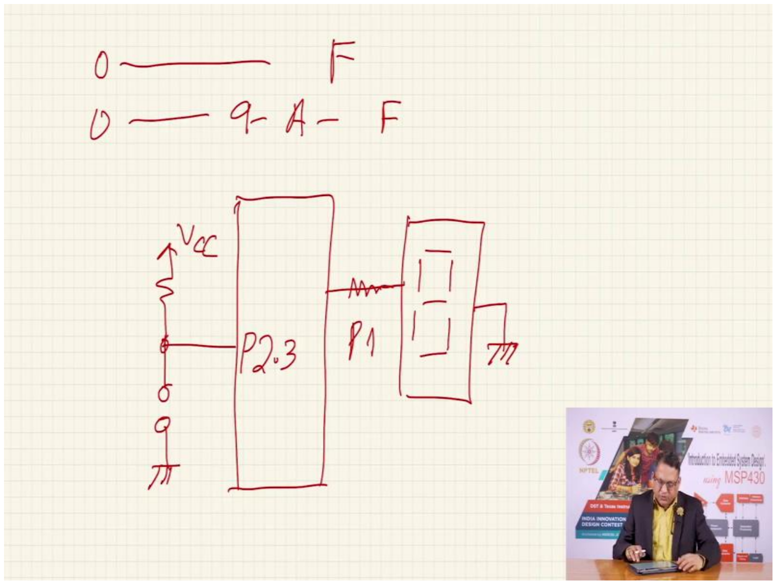
CC
a
b
g
f
e
d
c
dp
a b ... dp
CA
a – dp.
NPTEL
Introduction to Embedded System Design using MSP430
INDIA INNOVATION DESIGN CONTEST

We call this experiment, we call this exercise, HelloSSD because we are talking to a 7 segment display. As we have seen in the past, 7 segment displays are of two types; one can be called as common cathode or, so common cathode will be, here we have the display LEDs, this will be common cathode. And then you would have the 8 segment pins, a, b upto dp and this is a, b, c, d, e, f, g and here you have dp. Similarly, you can also have a common anode. So now, the common part is the common anode and you have the same display with a decimal point and you have a to dp.

You can choose either of these displays, in our case we have a common cathode 7 segment display. Now, let us see how we can interface it to our MSP430 microcontroller, using our MSP430 lunchbox. The connection to a 7 segment display is not very different from connecting a microcontroller to a LED, it is just that a 7 segment display consists of 8 LEDs. So, it is just a matter of connecting 8 pins to a single 7 segment display and turning appropriate segments on and off, so that you can display the number that you wish.

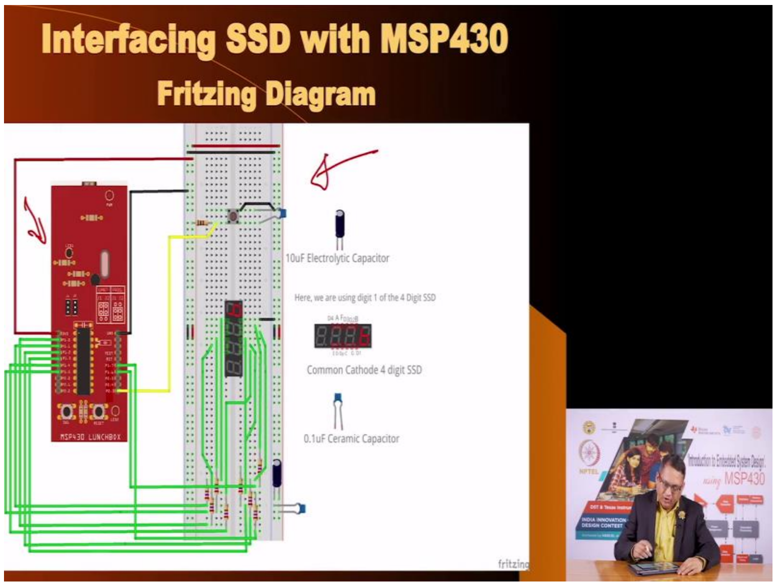
In our exercise, we what we are doing is we are connecting one switch to one of the port pins and we are connecting 8 pins of the microcontroller to the 8 pins of the 7 segment display. And each time we press and release a button, each time we press and release that switch it will count the value, from count going from 0, 1, 2, upto the maximum value.

(Refer Slide Time: 05:06)



We are showing the count in a hexadecimal fashion, which means we can display numbers from 0 to F, 0 to 9, and then A, B upto F. So, this is what we are going to do in this exercise. So, let me draw a sort of block diagram here is my MSP430 microcontroller, here is my switch connected to a pin which happens to be P2.3 in a pull-up mode and the port one pins P1 pins all 8 of them are being used to connect to the 7 segment display with appropriate current limiting resistors and because this is a common cathode display, we are going to ground the common cathode pin of the 7 segment display. Of course, in this exercise, we do not have a single digits, we have four digits.

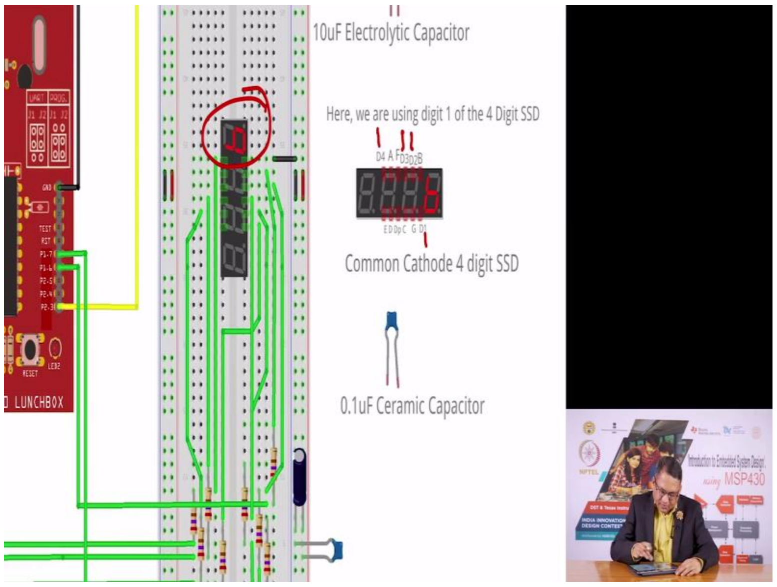
(Refer Slide Time: 06:08)



Interfacing SSD with MSP430

Fritzing Diagram

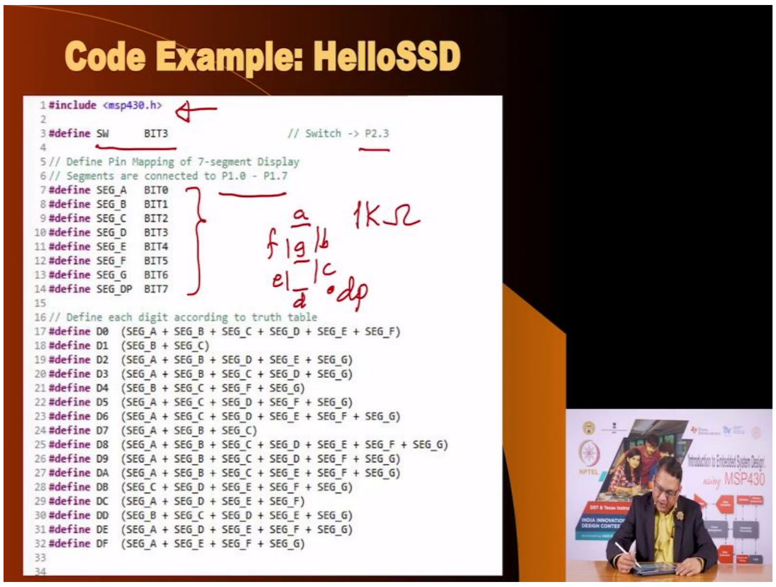
10uF Electrolytic Capacitor
Here, we are using digit 1 of the 4 Digit SSD
Common Cathode 4 digit SSD
0.1uF Ceramic Capacitor
MSP430 LUNCHBOX
fritzing


10uF Electrolytic Capacitor
Here, we are using digit 1 of the 4 Digit SSD
D4 A F D3 D2 B
E D Dp C G D1
Common Cathode 4 digit SSD
0.1uF Ceramic Capacitor
LUNCHBOX
Introduction to Embedded System Design using MSP430

This is the Fritzing diagram for the arrangement. Here we see our lunch box, here we see our breadboard and onto our breadboard we have four digit display, but as you see only one digit is being used. This 7 segment display as you see here, it has digit 1, digit 2, digit 3 somewhere, and digit 4, here digit 3 and digit 4. These are the common pins of each of these displays and then you have a, b, c, d, e, f, g and dp. We are going to connect all the 8 segments a through dp, and then we are going to only use one of the digit common pins as we see in here.

It does not matter which one you use; you can use first digit or the second digit. All you have to do is, you have to ground that digit common part, common signal. Of course, we have also connected to capacitors here, this is to you know reduce noise in this case. And then we have this switch here, with the pull-up resistor. And we also have connected a capacitor here, so as to debounce that although we are doing debounce in the software also. So, let us see what the code looks like.

(Refer Slide Time: 07:50)



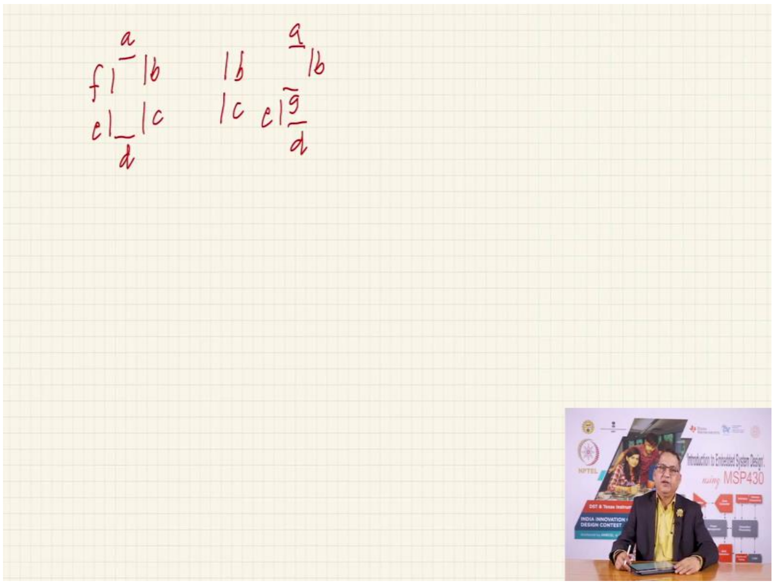
Code Example: HelloSSD

```
#include <msp430.h>

#define SW      BIT3                    // Switch -> P2.3

// Define Pin Mapping of 7-segment Display
// Segments are connected to P1.0 - P1.7
#define SEG_A   BIT0
#define SEG_B   BIT1
#define SEG_C   BIT2
#define SEG_D   BIT3
#define SEG_E   BIT4
#define SEG_F   BIT5
#define SEG_G   BIT6
#define SEG_DP  BIT7

// Define each digit according to truth table
#define D0  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_E + SEG_F)
#define D1  (SEG_B + SEG_C)
#define D2  (SEG_A + SEG_B + SEG_D + SEG_E + SEG_G)
#define D3  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_G)
#define D4  (SEG_B + SEG_C + SEG_F + SEG_G)
#define D5  (SEG_A + SEG_C + SEG_D + SEG_F + SEG_G)
#define D6  (SEG_A + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)
#define D7  (SEG_A + SEG_B + SEG_C)
#define D8  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)
#define D9  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_F + SEG_G)
#define DA  (SEG_A + SEG_B + SEG_C + SEG_E + SEG_F + SEG_G)
#define DB  (SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)
#define DC  (SEG_A + SEG_D + SEG_E + SEG_F)
#define DD  (SEG_B + SEG_C + SEG_D + SEG_E + SEG_G)
#define DE  (SEG_A + SEG_D + SEG_E + SEG_F + SEG_G)
#define DF  (SEG_A + SEG_E + SEG_F + SEG_G)
```

Now, here as we always seen that we include MSP30.h header file. Then, we have defined our switch to be on BIT3 and we actually connected to port 2, BIT3. And here we have our segments of the 7 segment display, A through DP, they are connected to port 1.0 to port 1.7 pins. We have used limiting resistors, and if you choose 1 kilo ohm or value of resistor around that, that should be, that should work fine.

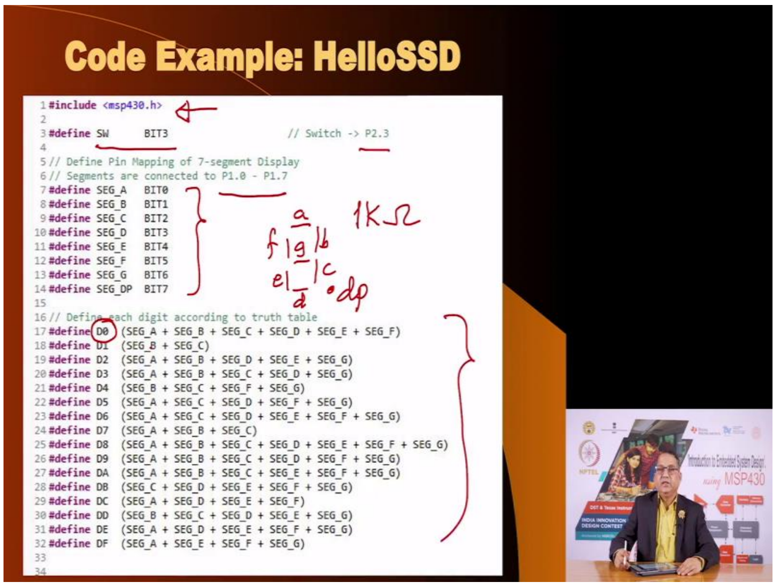
So, what we are doing is, we are defining that the segment A of our display, which is like this, a, b, c, d, e, f, g and dp that we have connected segment A to BIT0 that means P1.0, segment B to P1.1, and so on. Now, on this display, what would you like to display, we would like to display numbers.

(Refer Slide Time: 09:03)



Now, displaying numbers on a 7 segment display, one has to be very careful. For example, if you want to display 1, then or let us start with 0, so display 0 will look like this. That means all the digits except g and dp are off. This is a, b, c, d, e, and f when you want to display 1 then B and C are on, when you want to display 2 then you have a, b, d, e, g are on, and so on. So, you had to create a map that for a given digit to display which of the segments need to be turned on.

(Refer Slide Time: 09:47)



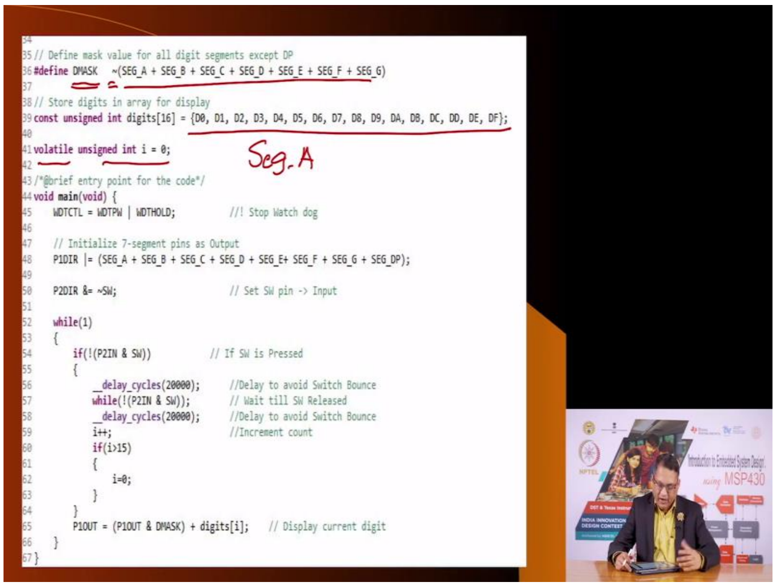
Code Example: HelloSSD

```
#include <msp430.h>

#define SW      BIT3                    // Switch -> P2.3

// Define Pin Mapping of 7-segment Display
// Segments are connected to P1.0 - P1.7
#define SEG_A   BIT0
#define SEG_B   BIT1
#define SEG_C   BIT2
#define SEG_D   BIT3
#define SEG_E   BIT4
#define SEG_F   BIT5
#define SEG_G   BIT6
#define SEG_DP  BIT7

// Define each digit according to truth table
#define D0  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_E + SEG_F)
#define D1  (SEG_B + SEG_C)
#define D2  (SEG_A + SEG_B + SEG_D + SEG_E + SEG_G)
#define D3  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_G)
#define D4  (SEG_B + SEG_C + SEG_F + SEG_G)
#define D5  (SEG_A + SEG_C + SEG_D + SEG_F + SEG_G)
#define D6  (SEG_A + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)
#define D7  (SEG_A + SEG_B + SEG_C)
#define D8  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)
#define D9  (SEG_A + SEG_B + SEG_C + SEG_D + SEG_F + SEG_G)
#define DA  (SEG_A + SEG_B + SEG_C + SEG_E + SEG_F + SEG_G)
#define DB  (SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)
#define DC  (SEG_A + SEG_D + SEG_E + SEG_F)
#define DD  (SEG_B + SEG_C + SEG_D + SEG_E + SEG_G)
#define DE  (SEG_A + SEG_D + SEG_E + SEG_F + SEG_G)
#define DF  (SEG_A + SEG_E + SEG_F + SEG_G)
```

And that we have listed in this sort of a hash define, where we are saying when we want to display 0, this D0 means when we want to display 0. Which of the segments should be on, so we are saying segment A, B, C, D, E, and F nothing else. When we want to display 1 it is segment B and C, as I mentioned and so on. And for this is mentioned for all the 16 digits, from 0 to 9 and then a, b, c, d, e, and f.

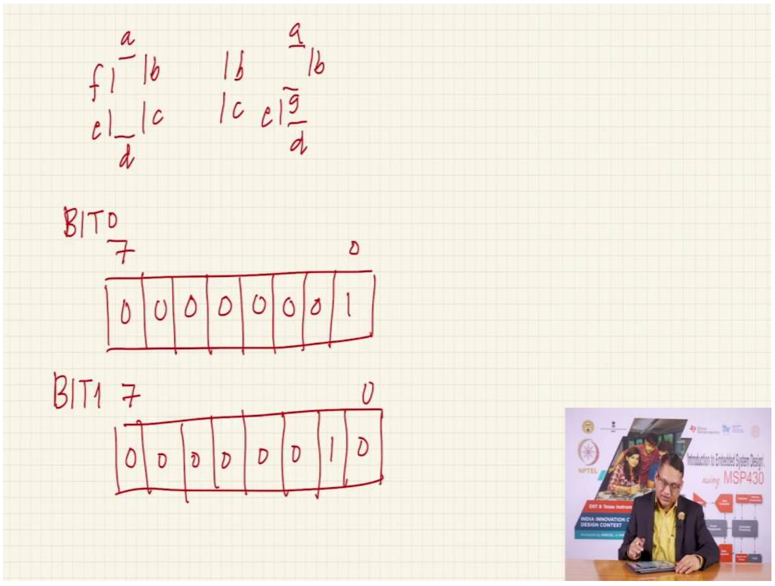
Of course, some of the alphanumeric characters will appear in capitals and some will appear in small letters and one can see what they will look like by simulating these numbers on a graph paper. So, this part of the code tells you that, when you want to display a number which of the digits should be, which of the segment should be turned on.

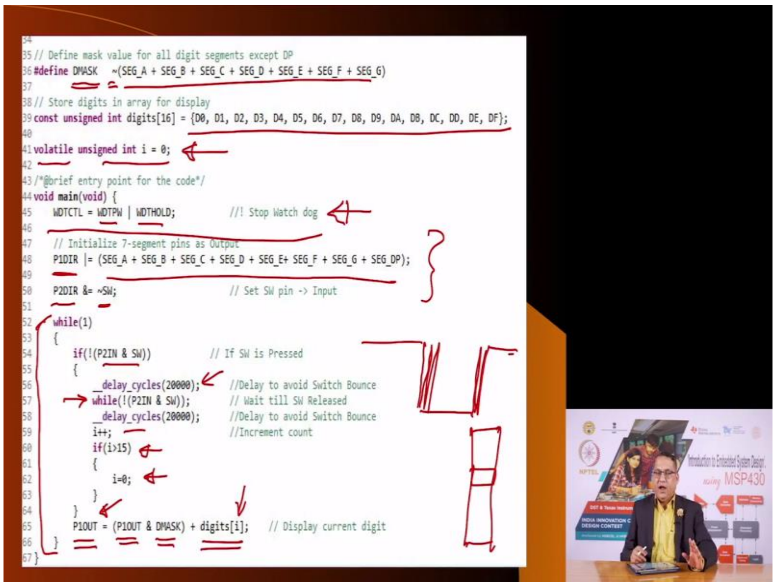
(Refer Slide Time: 10:42)



```
// Define mask value for all digit segments except DP
#define DMASK   ~(SEG_A + SEG_B + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)

// Store digits in array for display
const unsigned int digits[16] = {D0, D1, D2, D3, D4, D5, D6, D7, D8, D9, DA, DB, DC, DD, DE, DF};

volatile unsigned int i = 0;

/*@brief entry point for the code*/
void main(void) {
    WDTCTL = WDTPW | WDTHOLD;           //! Stop Watch dog

    // Initialize 7-segment pins as Output
    P1DIR |= (SEG_A + SEG_B + SEG_C + SEG_D + SEG_E+ SEG_F + SEG_G + SEG_DP);

    P2DIR &= ~SW;                       // Set SW pin -> Input

    while(1)
    {
        if(!(P2IN & SW))            // If SW is Pressed
        {
            __delay_cycles(20000);      //Delay to avoid Switch Bounce
            while(!(P2IN & SW));        // Wait till SW Released
            __delay_cycles(20000);      //Delay to avoid Switch Bounce
            i++;                        //Increment count
            if(i>15)
            {
                i=0;
            }
        }
        P1OUT = (P1OUT & DMASK) + digits[i];    // Display current digit
    }
}
```

Seg. A





```
// Define mask value for all digit segments except DP
#define DMASK   ~(SEG_A + SEG_B + SEG_C + SEG_D + SEG_E + SEG_F + SEG_G)

// Store digits in array for display
const unsigned int digits[16] = {D0, D1, D2, D3, D4, D5, D6, D7, D8, D9, DA, DB, DC, DD, DE, DF};

volatile unsigned int i = 0;

/*@brief entry point for the code*/
void main(void) {
    WDTCTL = WDTPW | WDTHOLD;           //! Stop Watch dog

    // Initialize 7-segment pins as Output
    P1DIR |= (SEG_A + SEG_B + SEG_C + SEG_D + SEG_E+ SEG_F + SEG_G + SEG_DP);

    P2DIR &= ~SW;                       // Set SW pin -> Input

    while(1)
    {
        if(!(P2IN & SW))            // If SW is Pressed
        {
            __delay_cycles(20000);      //Delay to avoid Switch Bounce
            while(!(P2IN & SW));        // Wait till SW Released
            __delay_cycles(20000);      //Delay to avoid Switch Bounce
            i++;                        //Increment count
            if(i>15)
            {
                i=0;
            }
        }
        P1OUT = (P1OUT & DMASK) + digits[i];    // Display current digit
    }
}
```

Now, let us come to the more definitions. Here we have a hash define we are saying, some sort of mask we have created which is nothing but all the digits, as you see except the decimal point we are not using the decimal point in this experiment, so apart from that we have (())(10:58) all the bits. So, when I say segment A, when I say seg A, and we have seen what is seg A, let us see here, seg A is BIT0 and what is BIT0, BIT0 is, BIT0 is of all the 8 bits the 0th bit is 1, that is the meaning of bits 0.

So, this is all 0, this is 0 to 7, 0, 0 all are 0 except 1. What is BIT1, instead of the 0th bit the 1th bit is on. This is 0 to 7, in this case it means this is 1 and these are all zeros. So, what we have done with that mask is, we are oring all the masks values for segment A, B, C, D, and so on. And then we are, we have inverted that value, we have toggled that value and we will see why. Then we have created an array, and this by the way is an array which has 16 digits, which have numbers from 0 to, to display numbers from 0 to F.

And then we have defined a variable, we have defined it to be volatile, we have covered interrupts and we know that volatile is usually when you are interacting between the main program and interrupt subroutine. In this case, there was no need, there is no need to declare this variable as volatile. Also, it is not needed that this variable, because it is outside the main loop it appears to be declared as a global variable, there is no need for this variable to be a global variable, but it does not hurt.

Now, let us go inside, now we have the main code and as we have always seen we have disabled the watchdog timer by oring the password with the bit required to stop the watchdog timer. And then we write this bit combination into the watchdog timer control register, that is the meaning of this statement, once we are disabled this.

Now, once you reset the system, it is going to, what clock it is going to use for the CPU, it is going to use the M clock and M clock will be derived out of DCO and the DCO will be working at 1.1 megahertz, that we have not changed that is the reset state of MSP430 and we are happy to use our lunchbox at that rate.

Then, the first task we do is to define the direction of the port 1 bits. We have defined all the bits as outputs from segment A to the decimal point, we are using all the bits, so all the pins of port 1 we have defined as outputs. And then for port 2, we have defined this bit as input by making it 0, when we make it 0 it becomes input, when we make it 1 it becomes output. Once these two port directions are defined, we have entered a infinite loop while 1 and from here we are going to remain into this loop for all the time.

In this what we are doing, we are waiting for a switch to be pressed. So, if the switch is not pressed you are going to wait, the moment the switch is pressed you are going to go inside. Which means, when the switch is not pressed you are going to get a logic 1, the moment the switch is pressed it is going to go here. It is likely to bounce, although we have put a capacitor outside so the capacitor is going to filter this out. But even if it does, we debounce it because of this delay Loop. Then we wait for the switch to be released, which means for all the time that it remains 0 you are going to wait here.

And then when it is released, again it is likely to bounce and then after the delay, it is going to go high and when the switch is released, we are going to increment our variable i is going to count up. The initial value of that variable was set to 0, and the moment you increment the count you want to check the bounds because we want to count only from 0 to F, that is 0 to 15. If you keep on pressing at some point the value of i will become 16 but 16 is, you cannot display here. So, you are going to reset the value of the variable to 0.

So, we are checking that after incrementing if the value of the variable is more than 15 you set the value to 0, if not you proceed. And what you do, you are going to output something on port 1 so as to display the number but before that whatever was being previously displayed you bring that value and, and it with the DMASK.

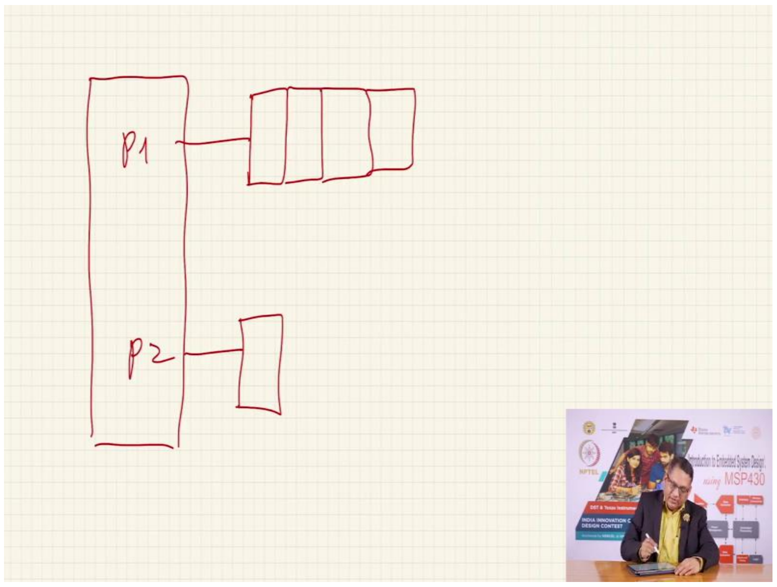
And DMASK as you know, is nothing but the or of all these bits inverted which means it is going to disable those LEDs which are on and it is going to or it with a number which is nothing but into an array, which the array name is digits you are going to fetch a value depending upon the value of I, depending upon the value of i that you see here, you are going to fetch that bit from one of these, which will be one of these numbers.

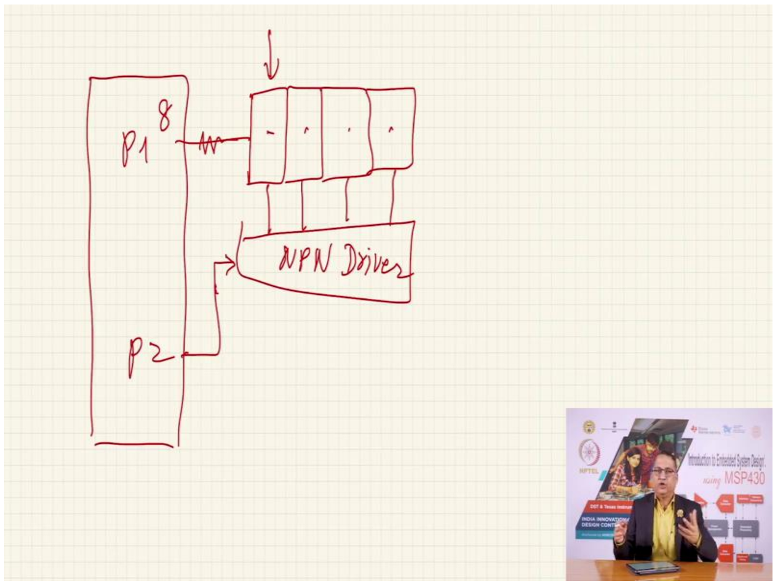
And these numbers we have already seen are defined here, here. So, you are going to fetch one of these values depending upon what is the value of i and that value you are going to output on port 1. So, what will it do, it will display that number and that number will remain as long you do not press a switch because it is going to go back here and wait for you to press a switch.

So, this I recommend that you rebuild this code and download it into your MSP430 lunch box and play with it, so that you understand what is happening, how the 7 segment display is working. You can play with the value of the resistors as I mentioned here, if we are using a 1k kiloohm resistor, you can increase it. You should notice that the display intensity should go down, if you reduce it you should see that the intensity of the display will increase and so on.

So, I recommend that you experiment with the setup as much as you would like, so as to gain useful insight into how to use a 7 segment display for showing information from a embedded application to the outside world. Now, of course if we have only 1 digit to display, it is not of much use. And in real life you may want a display a larger number. Now if you use a larger number what options do we have.

(Refer Slide Time: 18:03)





Here, we can use two digits. So, on a MSP430 I can easily do that, I have two ports. So, on port 1 I can connect one display and port 2 I can connect under the display, provided I am not using an external crystal because external crystal takes up some of the port pins. But if I want to do display numbers beyond this, I have a problem because I do not have enough pins. And the way to handle this is what we have covered in the previous lecture, of using a multiplexing technique.

In multiplexing technique, we will use multiple displays and, in our case, we have access to a four digit display, so we will connect port 1 to 4 digits and the segments, the digits will be controlled here, here, here, here through NPN transistor drivers. From, so this is NPN driver and of course here we have resistors, see these are all the 8 pins, we have 8 resistors and 4 pins from port 2 we can use to control multiple 7 segment displays.

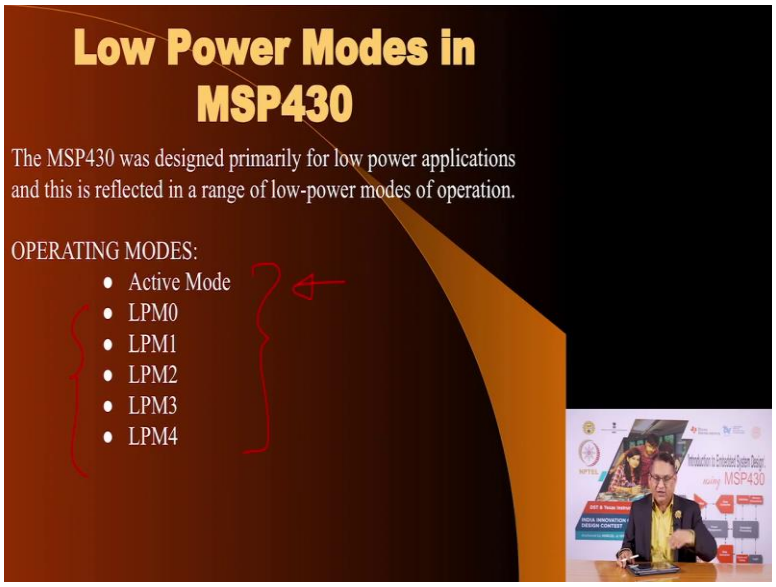
But the advantage that you get by controlling such a display using less number of pins you have to pay in some way. And in this case, the payment is through a software overhead, which means at any given time only one digit is going to be on. And how do you change it? this is the topic we are going to cover in a subsequent lecture, how we can change from first to second to third to fourth back to first and keep on doing it in the background while your main loop can do other jobs of counting events, and counting numbers and having them displayed on these four digit display.

We will show you one experiment using, after we have covered some information about timers, we will show how we can use the timer infrastructure of MSP430 to multiplex multiple 7 segment displays, thereby reducing the usage of physical pins of MSP430 microcontroller. This is going to be a topic of a subsequent lecture, till in this one you please play with that setup of controlling a single digit display and you can gain some insight.

Having connected a 7 segment display to the MSP430 microcontroller, we are now ready to evaluate and understand another important aspect of MSP430, one which is hallmark and salient feature of MSP430 which is its ability to go from active mode where all the peripherals, the CPU and everything in the microcontroller is working to your, to the maximum capabilities.

You can switch from that to a low-power mode and there are several low power modes, which allow you to conserve power. So, if you find that your application does not need to be active, you can switch on to the low power modes and we will see how many lower power modes MSP430 offers, and how to invoke them, and how to get back from the low power modes into active mode so that you can process information as and when it is deemed necessary.

(Refer Slide Time: 21:27)



Low Power Modes in MSP430

The MSP430 was designed primarily for low power applications and this is reflected in a range of low-power modes of operation.

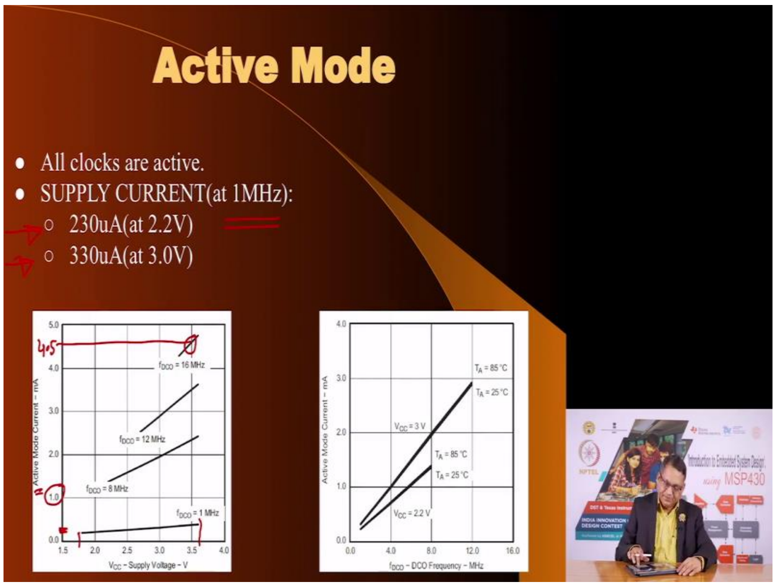
OPERATING MODES:

- Active Mode
- LPM0
- LPM1
- LPM2
- LPM3
- LPM4

So, we are going to consider the low power modes, the operating modes of MSP430 are 6 in all. The first is the active mode where everything is working, the CPU is working, the peripherals are working all the clock sources are working, all the clock signals are working you can choose to route whichever clock signal into appropriate peripherals. But once you are, once you have decided, once you have deemed that in a particular situation, you do not want to be in active mode, you can choose various low power modes.

And there are five low power modes and we will see how much power reduction happens in each of these modes and you can choose one appropriately. We will see how to enter those modes and how to exit those modes.

(Refer Slide Time: 22:13)



Active Mode

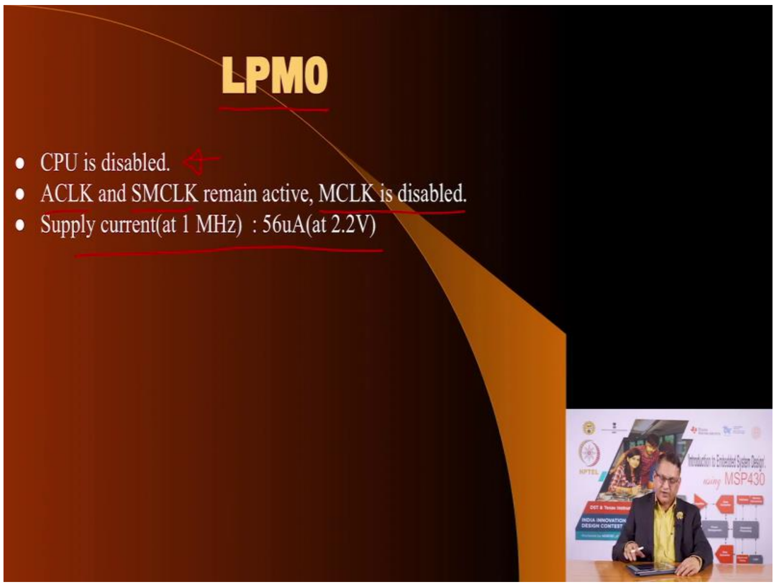
- All clocks are active.
- SUPPLY CURRENT(at 1MHz):
  - 230uA(at 2.2V)
  - 330uA(at 3.0V)

Now, in the active mode if your processor is working, if your system is working CPU is working at 1 megahertz, then this is, these are the operating diagram from the datasheet of MSP430. And you see that, for 1 megahertz you are going to consume this is the active current and this is the supply voltage, you can go from roughly 1.8 volts to 3.6 volts that when the DCO is working at 1.1 approximately 1 megahertz the amount of current is, this is 1 milliampere. So, this is roughly 2-300 micro amperes, so this is what it is. At 2.2 volts supply voltage you can go from to 230 microamperes to at 3 volts it is about 330 microamperes.

And here if the temperature changes then there is a impact on the frequency, DCO frequency and then you see the current consumption also changes. So, this is as far as the current consumption of MSP430 in full active mode at various frequencies, you can estimate from here. For example, if your system is running at 16 megahertz at 3.5 volts, you can imagine that its going to consume about 4.5 milliamperes, at 16 megabytes and that is a significantly small amount of current.

So, you should refer to these graphs whenever you have to make estimates as to how much current budget, how much, how much power supply specifications you must provide for so that your system works flawlessly.

(Refer Slide Time: 23:55)

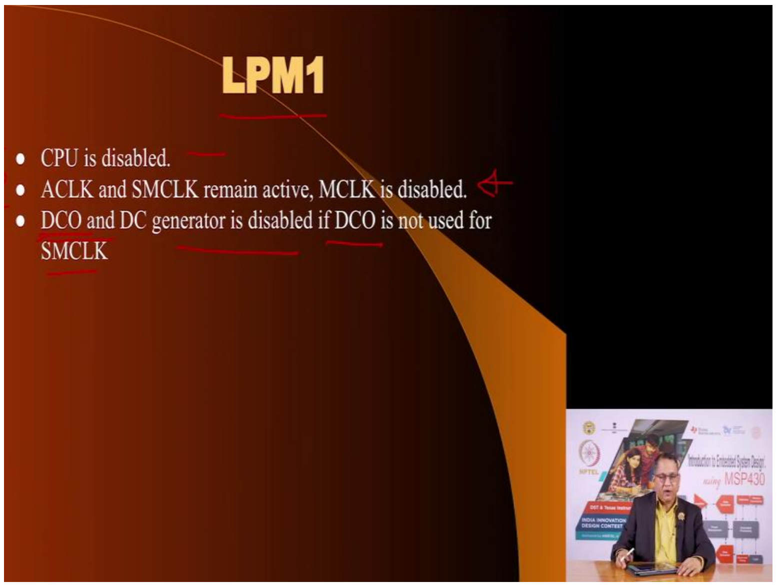


LPM0

- CPU is disabled.
- ACLK and SMCLK remain active, MCLK is disabled.
- Supply current(at 1 MHz) : 56uA(at 2.2V)

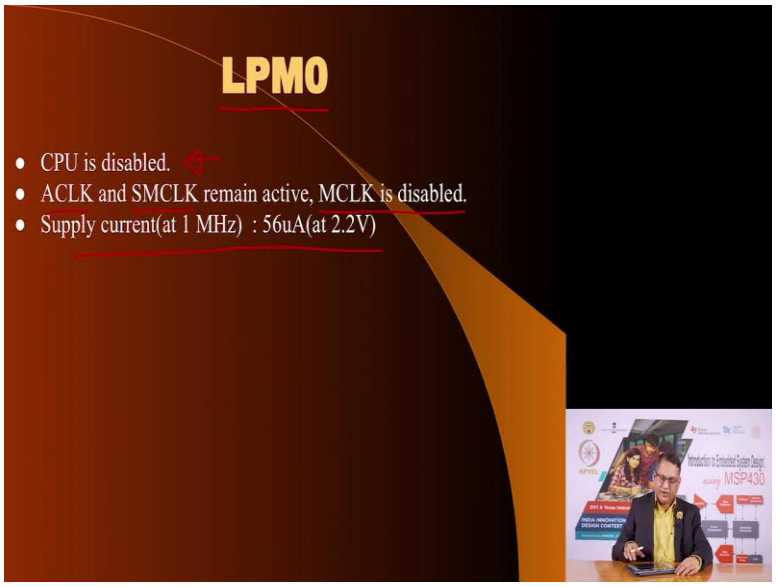
If you are not interested in remaining in active mode, then the first low power mode is called LPM0, LPM stands for, Low Power Mode 0. What happens in that, the CPU is disabled which means you are not executing any program but some of the peripherals, depending upon which clog they choose to use some of those peripherals can continue to work. Why, because ACLK and SMCLK are active. MCLK which is the Master Clock, which feeds the CPU is off which means CPU is not working. And the supply current at 1 megahertz drops from that 230 micro amperes down to 56 micro amperes. I must reiterate here, that because A clock and SM clock are active the peripherals, if you have chosen A clock or SM clock as the clock for those peripherals, those peripherals will continue to work.

(Refer Slide Time: 24:49)



LPM1

- CPU is disabled.
- ACLK and SMCLK remain active, MCLK is disabled.
- DCO and DC generator is disabled if DCO is not used for SMCLK



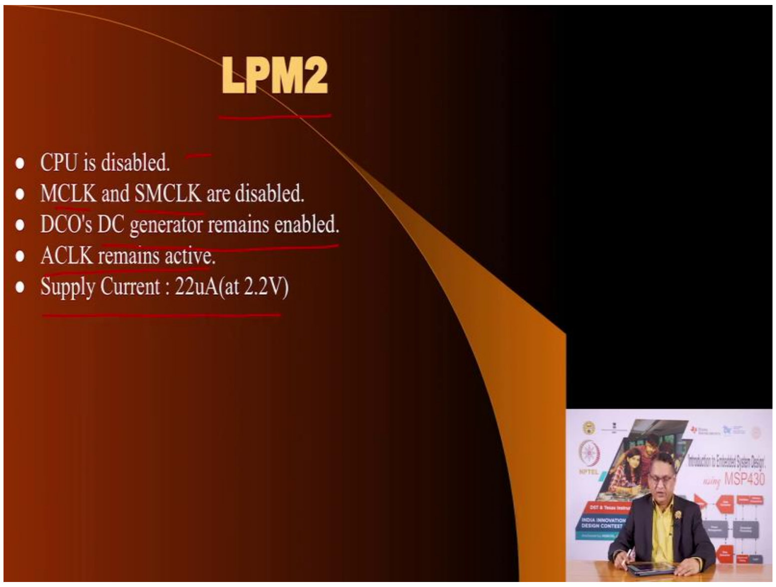
LPM0

- CPU is disabled.
- ACLK and SMCLK remain active, MCLK is disabled.
- Supply current(at 1 MHz) : 56uA(at 2.2V)

Beyond LPM1, LPM0 you can choose to go for LPM1. Here also, the CPU is disabled. Your A clock and SM clock are still active, your MKCL is disabled. Now, what is the difference between LPM0 and LPM1, if the first two features are common in both of them, you see here also you have. The part is that, if the DCO, the Digitally Controlled Oscillator is disabled, if the DCO is not being used to power the SM clock. As you know, SM clock derive the clock from DCO as well as the, VLO and low-frequency crystal oscillator.

If you have chosen the SM clock not to be source from DCO, then the DCO that is the oscillator as well as the DC bias generator for the DC oscillator are disabled and this saves you power.

(Refer Slide Time: 25:49)

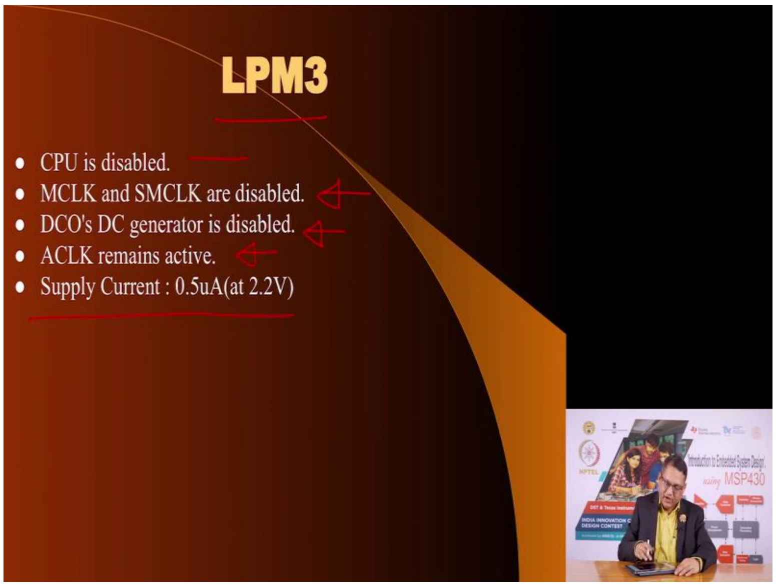


LPM2

- CPU is disabled.
- MCLK and SMCLK are disabled.
- DCO's DC generator remains enabled.
- ACLK remains active.
- Supply Current : 22uA(at 2.2V)

Then you can go to LPM2 in which CPU is disabled. Now, M clock and SM clock both are disabled the DCO can be on, the DCO’s DC generator remains enabled. And the reason why, why the DC generator is enabled is because the DC generator biases the digitally controlled oscillator. If the bias is on, the DCO, the DCO can be turned on quickly. The is, A clock remains active and the supply current drops to about 22 micro amperes.

(Refer Slide Time: 26:30)

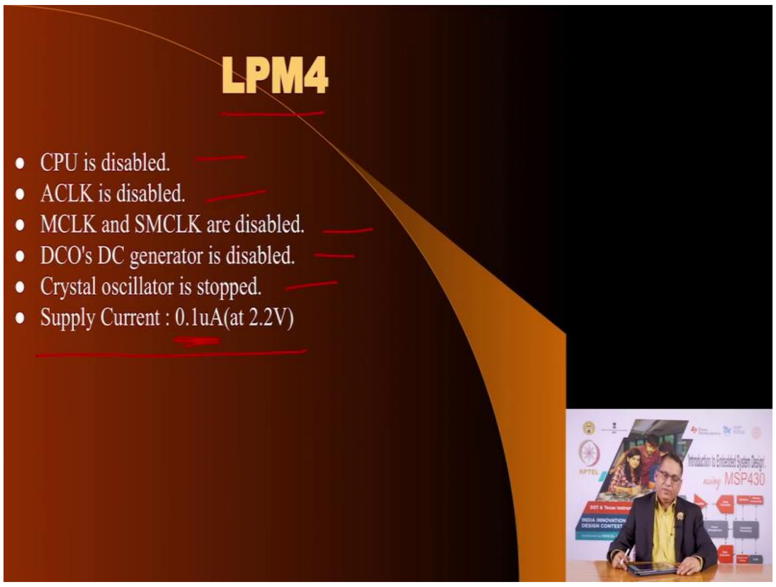


LPM3

- CPU is disabled.
- MCLK and SMCLK are disabled.
- DCO's DC generator is disabled.
- ACLK remains active.
- Supply Current : 0.5uA(at 2.2V)

Beyond LPM2, you can go for LPM3. Here, the CPU is disabled your M clock and S clock are also disabled the DCO’s DC generator is disabled. Now, if the DC generator is disabled, if you want to switch back to active mode it is going to take some time because first the biasing will be turned on, and then the DC oscillator will turned on it will take some time to come back to active mode. A clock of course remains active which means some peripherals can choose to use A clock for its operation and the supply current drops to a staggeringly low 0.5 micro amperes at 2.2 volts.

(Refer Slide Time: 27:09)

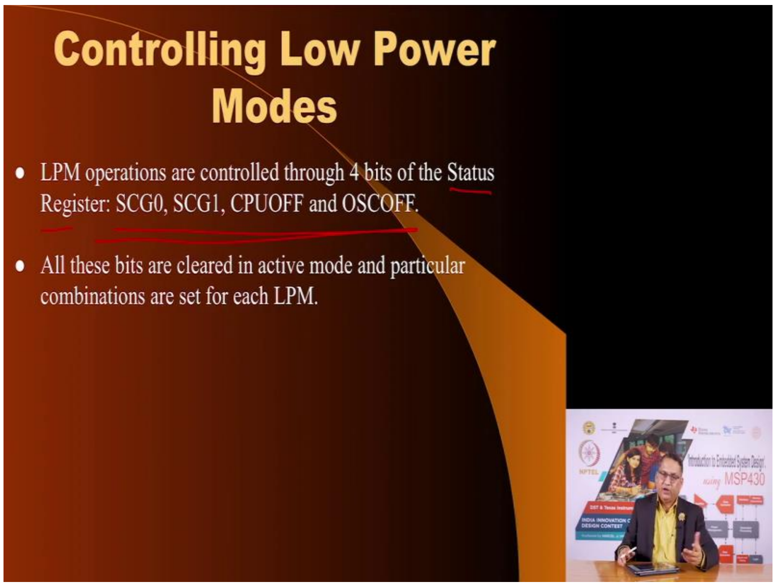


LPM4

- CPU is disabled.
- ACLK is disabled.
- MCLK and SMCLK are disabled.
- DCO's DC generator is disabled.
- Crystal oscillator is stopped.
- Supply Current : 0.1uA(at 2.2V)

Here we have LPM4. And now the CPU is disabled, the A clock is also disabled the rest of the clocks are also disabled, your DCO generator is disabled, the crystal oscillator is stopped. And now you see, this is the least power consuming mode where only the contents of the register will be maintained, rest nothing is going to work because all the clocks are disabled and your supply current is going to drop to merely 0.1 microampere.

(Refer Slide Time: 27:42)

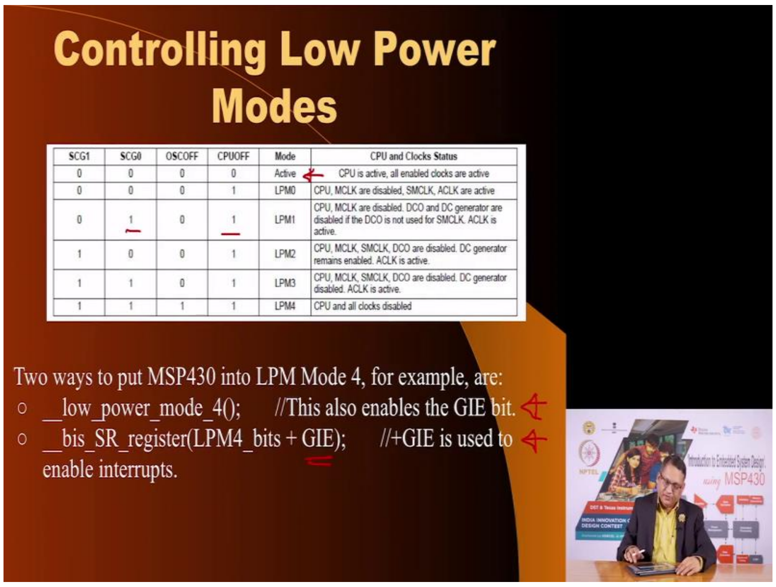


Controlling Low Power Modes

- LPM operations are controlled through 4 bits of the Status Register: SCG0, SCG1, CPUOFF and OSCOFF.
- All these bits are cleared in active mode and particular combinations are set for each LPM.

Now, you can use choose to enter these low power modes using the 4 bits in the status register and these 4 bits are these and we will see how these bits, bit combinations could be invoked appropriately to enter one or the other low power modes.

(Refer Slide Time: 28:00)



Controlling Low Power Modes

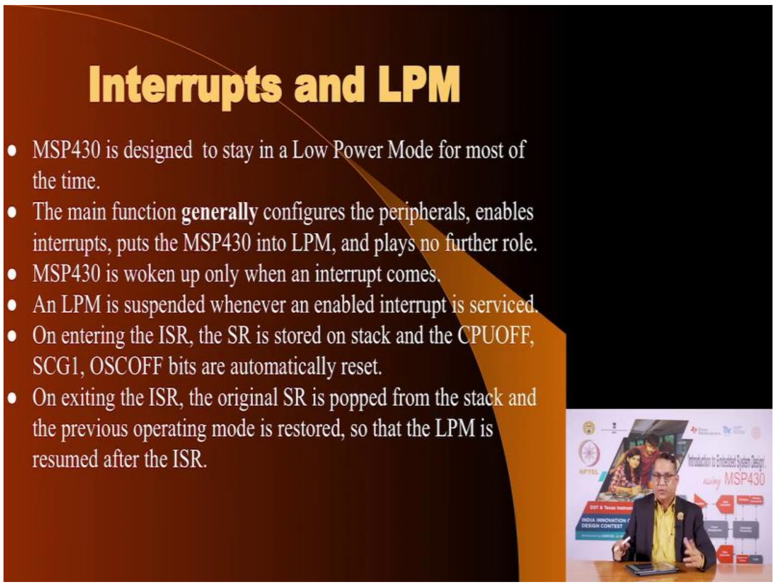
SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled, SMCLK, ACLK are active
0	1	0	1	LPM1	CPU, MCLK are disabled. DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active.
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO are disabled. DC generator remains enabled. ACLK is active.
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO are disabled. DC generator disabled. ACLK is active.
1	1	1	1	LPM4	CPU and all clocks disabled

Two ways to put MSP430 into LPM Mode 4, for example, are:
- __low_power_mode_4();   //This also enables the GIE bit.
- __bis_SR_register(LPM4_bits + GIE);   //+GIE is used to enable interrupts.

So, here we have when all the bits are 0 you are in the active mode and you see this is the contents of the status register. And we have always mentioned, that when you reset your microcontroller, the status register is cleared, which means all the bits in status registers are 0. Which means, after reset your video enter in the active mode, after that you can choose if you want to make the, enter the LPM0 mode you make CPU of bit equal to 1 and LPM1 you bit turn this, bit on and on and so on.

And of course, your CCS, the Code Composer Studio allows you many macros to invoke various modes without having to worry about which bits to turn on and off. Now, if you enter any of the low power modes, the only way you can get out of it is through interrupts and interrupts will only work if you have enabled the interrupts, which means the global interrupt enable flag must be turned on. So, you can use either this macro and use it in your code to get out of the low-power mode or you can use this combination of command to get out of the low-power mode.

(Refer Slide Time: 29:22)



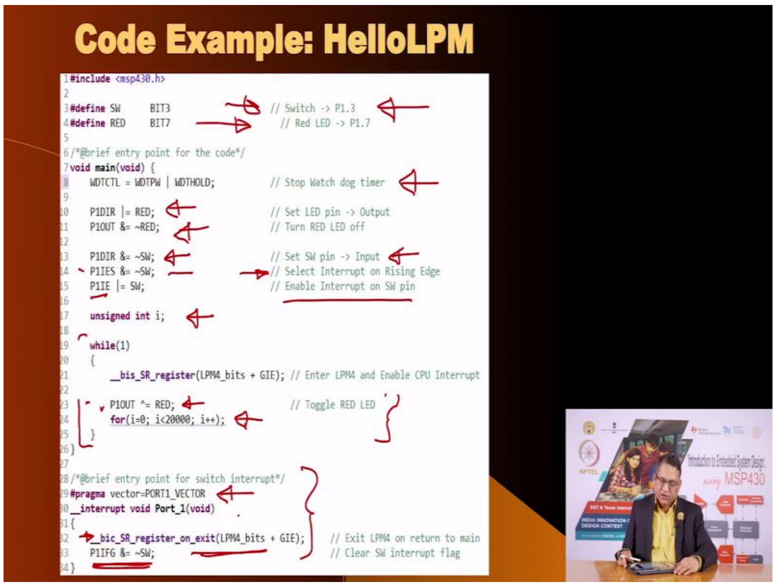
Interrupts and LPM

- MSP430 is designed to stay in a Low Power Mode for most of the time.
- The main function **generally** configures the peripherals, enables interrupts, puts the MSP430 into LPM, and plays no further role.
- MSP430 is woken up only when an interrupt comes.
- An LPM is suspended whenever an enabled interrupt is serviced.
- On entering the ISR, the SR is stored on stack and the CPUOFF, SCG1, OSCOFF bits are automatically reset.
- On exiting the ISR, the original SR is popped from the stack and the previous operating mode is restored, so that the LPM is resumed after the ISR.

We will now illustrate this with the help of a code to show this operation. Now, we have seen the various modes how to invoke them. We must put together, we must consider the low power modes and the interrupts together because if the MSP is designed to remain in low-power mode for most of the time, that is fine if you want to get out of it you must take the support of interrupts. And oftentimes, the main code will you know program the peripherals to do their job depending upon which low power mode you have chosen.

If you want certain peripherals to operate, you have to select appropriate clocks for feeding those peripherals and you have to select appropriate low power modes. But, if you want to get back into active mode, you must turn the general global interrupt enable bit as one and then hope that these peripherals can generate an interrupt whenever some action is encountered, so that the processor can come out for of the low-power mode and into the active mode.

(Refer Slide Time: 30:32)



Code Example: HelloLPM

```c
#include <msp430.h>

#define SW      BIT3                    // Switch -> P1.3
#define RED     BIT7                      // Red LED -> P1.7

/*@brief entry point for the code*/
void main(void) {
    WDTCTL = WDTPW | WDTHOLD;           // Stop Watch dog timer

    P1DIR |= RED;                       // Set LED pin -> Output
    P1OUT &= ~RED;                      // Turn RED LED off

    P1DIR &= ~SW;                       // Set SW pin -> Input
    P1IES &= ~SW;                       // Select Interrupt on Rising Edge
    P1IE |= SW;                         // Enable Interrupt on SW pin

    unsigned int i;

    while(1)
    {
        __bis_SR_register(LPM4_bits + GIE); // Enter LPM4 and Enable CPU Interrupt

        P1OUT ^= RED;                       // Toggle RED LED
        for(i=0; i<20000; i++);
    }
}

/*@brief entry point for switch interrupt*/
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
    __bic_SR_register_on_exit(LPM4_bits + GIE);    // Exit LPM4 on return to main
    P1IFG &= ~SW;                                  // Clear SW interrupt flag
}
```

Here is a code example, where what we have done is we are using the peripherals on the lunchbox. We are not doing anything else; we do not want you to connect anything other than the lunchbox to any other, you know connection on the breadboard whatever is available on the lunchbox is suitable. We have one switch which as you know is connected to port 1 bit 3, and we have a LED connected to port 1 bit 7.

Now, let us see what we are doing. What this code example shows, we and you will have to play with this code and I will show you what part to actually see the impact of this program. What this does is, that this program will enter into low power mode and by pressing a switch, that is the switch connected to port 1.3 you can get out of it and let us see what it does. So, first of all wherever we reset we are, the first instruction we are executing is to disable the watchdog timer.

Then we are turning the port 1.7 bit as output and port 1.3 as input and before that, we are turning the LED connected to port 1.7 off. And we have, this is very important, after we have converted after we have decided that port 1.3 pin should be input, we are also enabling the interrupt on port 1.3 in the rising edge mode and we have enabled the interrupt on that pin. So, we have access to registers, port 1 interrupt enable and port 1 edge selection register. So, we have decided that whenever the switch is pressed and released, at release you will get a rising edge it should interrupt the system.

Then we have a variable declared I, we will see what we are doing with it. And then we enter a infinite loop. Outside that here is your interrupt vector, where you are going to go whenever the switch is pressed. Why, because we have enabled interrupts on port 1 and so whenever the switch is pressed it will generate an interrupt, it will go into the vector. And as you see, we have gone through that earlier, you are saying hash pragma vector is equal to port 1 vector. But before that, let us see what, what you do here.

So, this instruction is very important, right now this instruction will execute. In this what you are doing, you are saying bit set in the SR register, so as to select LPM4 mode and enable the global interrupts. Now, the moment you execute this instruction the CPU has got to stop, which means nothing will happen till you get out of the low power mode, which means next instruction which is here is not going to execute. So, nothing is going to happen on the LED if the LED was off it will remain off, but if they really was on it will remain on.

The first time around the LED will be off. Now, what happens, you press the switch you are going to go into this interrupt and in this interrupt, you are exiting from the low-power mode why, you are clearing the appropriate bits in the status registers. You are also enabling the interrupts again, and you are resetting the bit in the port 1 interrupt register, so that next time around when the switch is pressed you can again register and interrupt. Now, because of this instruction you are going to go in the active mode, which means when you come back here, you will execute this instruction, which will turn the, which will toggle the LED.

Now, this instruction which deal is, is actually superfluous as far as running this code with this instruction active. So, you can play with this and you will see that every time you press the switch it toggles. And, but if you comment this instruction, if you comment this instruction out, which means interrupts have really no, you know, you have, you are not, you know turning the interrupts on, so interrupts are not going to work. Then, the only two instructions in this loop are toggle the LED, delay, toggle the LED, delay.

So, in this mode you are going to see that the LED is continuously going to toggle, when, when you comment out this instruction. If you comment out this instruction rebuild your code and download it, you are going to just see LED on port 1 bit 7 toggling, at a rate of roughly, you know few tags a second.

The switch has no impact because we are not reading it in the main program and the interrupt is not enabled but the moment you comment out means, when you make this instruction active as part of this code, then you will see that the if the LED is on it will remain on, it is not going to toggle, it is only going to toggle when you press the switch.

Why, because when you press the switch it is going to enable the interrupts, in the interrupt sub routine it is going to disable the low power mode, go back into active mode, and will toggle the LED, and will come back, and again enter the low power mode. So, once you press the switch, if it, if the LED was on it will be off, will remain off. Then when you press the switch again, it will again exit the low power mode, go in the active mode, toggle the LED, again go in the low power, and so on.

So, this is a interesting exercise to see how you can play with the low-power mode, how you can have external peripherals exit the low-power mode. And I strongly recommend, that you experiment with this code, specially by playing with this instruction, by commenting this instruction out or keeping it in your code to see the impact. So, we are done with the description of the low power modes, as well as how to connect simple 7 segment display to the pins of MSP430 microcontroller. In a subsequent lecture, we will deal with timer other issues. Thank you, for watching.