

**Introduction to Embedded System Design**  
**Professor Dhananjay V. Gadre**  
**Netaji Subhas University of Technology, New Delhi**  
**Lecture - 24**  
**MSP430 Clock System and Reset**

Hello and welcome to a new session for this online course on Introduction to Embedded System Design. I am your instructor Dhananjay Gadre. In this lecture, we are going to look at one of the most important aspect which forms an important part of the ecosystem for a microcontroller to function effectively, namely clock and reset sources. We have seen in a previous lecture that a microcontroller require four very important elements clock, reset, power supply, and an ability to download code into the memory of the microcontroller.

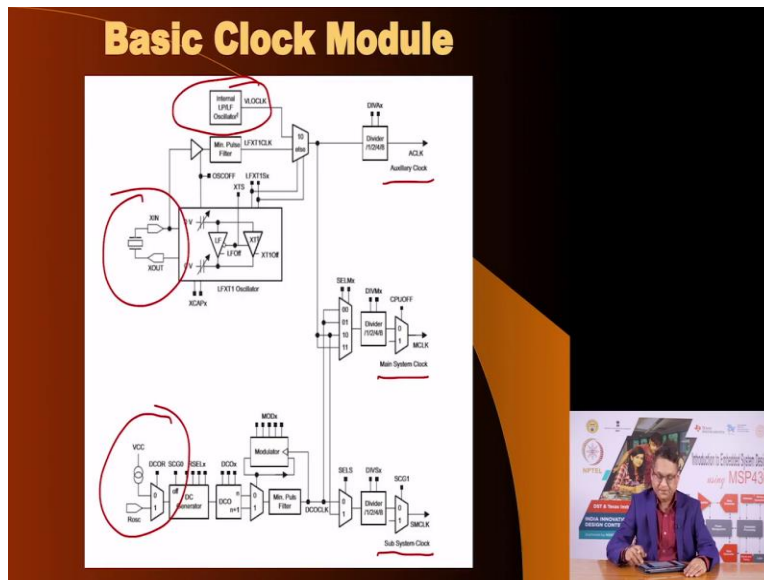
In this lecture, we are going to deal with the first two issues, namely clock and reset. Now, why does a microcontroller require a clock? The reason is a microcontroller is an example of a synchronous digital circuit. And because I have used the word synchronous that means it is going to use a clock signal, and therefore I must provide a, such a signal. Also, the value that is the frequency of this clock signal will determine lot of things.

The frequency, higher the frequency, higher the performance because your microcontroller will be able to execute more instructions per second if the frequency is high, but it will come with a price. And the price is that as the frequency is gets higher and higher, the power dissipation will proportionately increase because the power dissipation of a CMOS circuit is directly proportional to the frequency of operation.

And so we must decide what is an optimum frequency for the operation of the microcontroller at any given point of time. Fortunately, and it was one of the salient features which I had mentioned in the salient features of microcontrollers, MSP430 offers the ability to be able to dynamically change the clock frequency using a software.

Using the user program, the user can decide at any given point of time do they want a higher frequency for operation because the performance requires so or if there is no work to be done, there is no point in clocking the microcontroller at a high frequency and instead, a lower frequency operation could be selected so as to conserve available power. So let us look at the clocking options of MSP430.

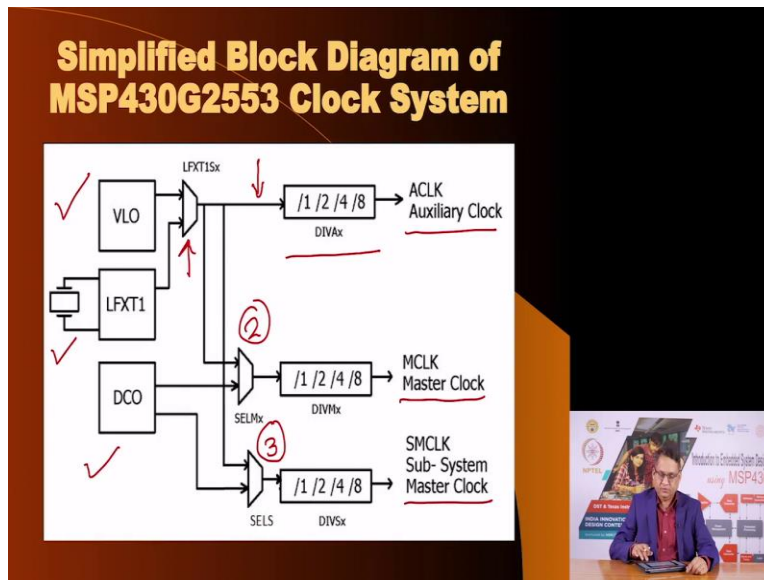
(Refer Slide Time: 02:49)



This is the block diagram of the clock module, which is there inside MSP430. On the left, you have the sources of clock and on the right of this diagram, you see signals which are derived from these three sources. There are three sources. This is one of them. An internal VLO clock that is called, an external crystal-based oscillator and an internal digital, digitally controlled oscillator. These are the sources and using a particular combination of these three sources, the MSP430 clock module offers three signals, clock signals.

It is called A clock that is auxiliary clock, the main system clock, and a subsystem clock. Let us see what these clock signals do, what part of the microcontroller do they serve, and how their frequency can be changed.

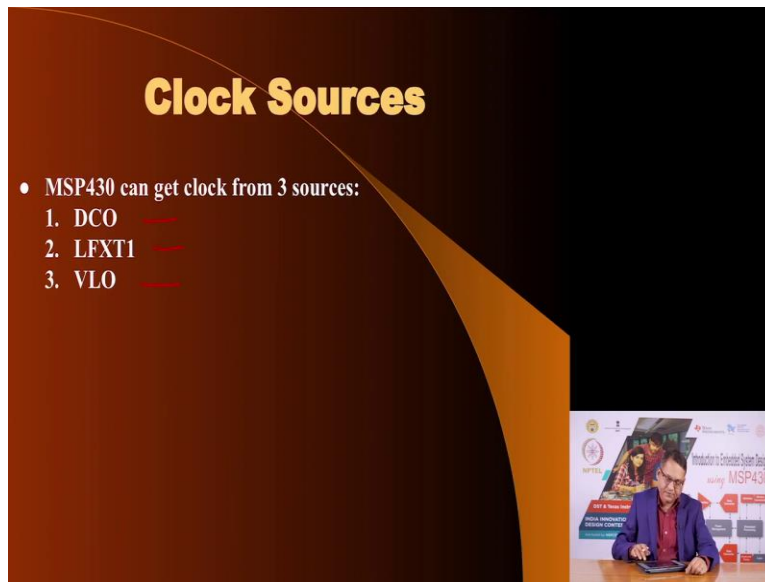
(Refer Slide Time: 03:54)



This is a simplified block diagram of the clock module as applicable in MSP430G series. You have a very low-frequency internal oscillator, it operates at about 60 kilohertz, we will come to the details. You have a low-frequency crystal oscillator; the oscillator is inside the microcontroller. The crystal has to be connected to external signals, external pins, and apart from that, you also have a digitally controlled oscillator.

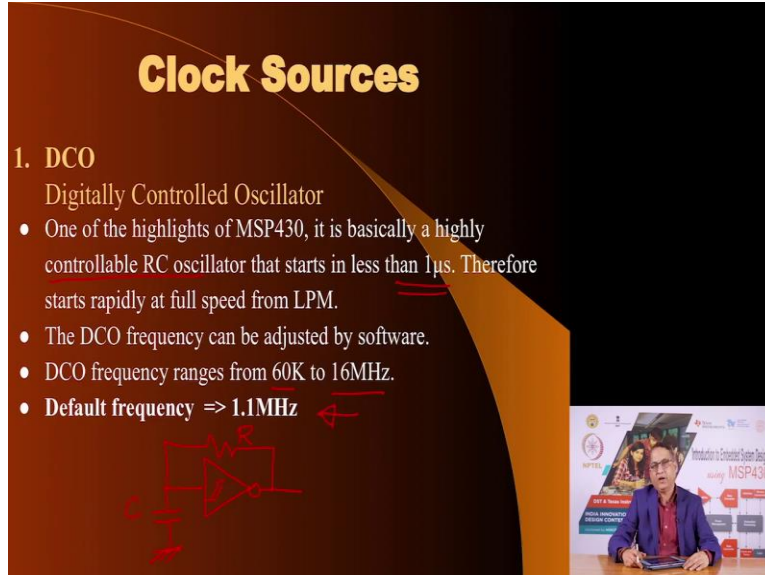
There are three multiplexers, multiplexer number 1, number 2, and number 3, and each multiplexer has two inputs broadly. And you can write software to select one or the other source of clock. Once you decide the clock source, it can be further divided with a clock divider using an option of divide by 1, 2, 4, or 8, to derive three clock signals. And these three clock signals are auxiliary clock, master clock, and subsystem master clock. Let us see what these clock signals are used for.

(Refer Slide Time: 05:09)



So, as I mentioned, there are three sources, a digitally controlled oscillator, a low-frequency crystal oscillator, and a very low-frequency oscillator inside the MSP430.

(Refer Slide Time: 05:23)

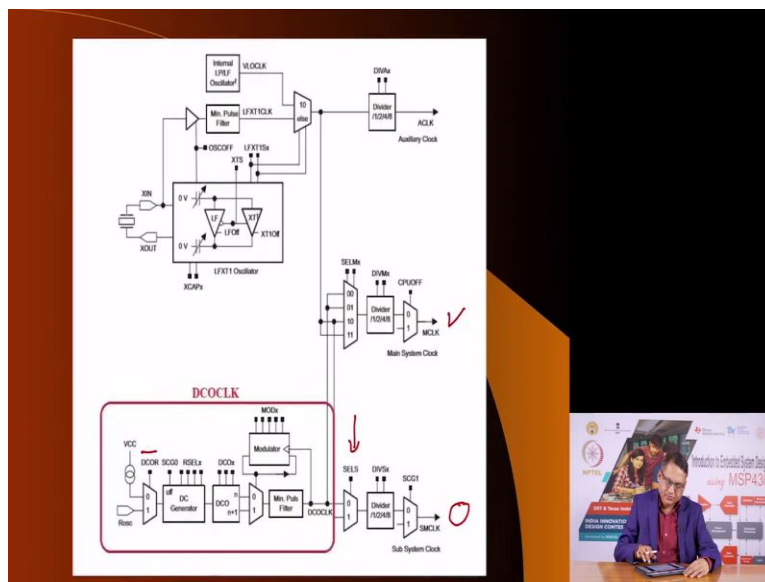


The digitally controlled oscillator is based on an RC, an internal RC oscillator, and let me give you a very simple circuit that may illustrate how an RC circuit can be made using a logic gate such as an inverter here. This will oscillate at a certain frequency as determined by the value of this R and this C. This is not to suggest that the DCO uses this. This is just an example.

Anyway, the digital controlled oscillator allows you to change the frequency operation in this range from 60 kilohertz to 16 megahertz for a certain supply voltage. If the supply voltage is changed, these numbers, these upper and lower limits would change. It is a RC, controlled RC oscillator, as I have mentioned, and because it is a very quick-start oscillator, it can start working in less than a microsecond, it can be used to get out of low-power modes of operation, we are going to consider low-power modes of operation of MSP430 in a subsequent lecture, but this digital controlled oscillator allows you to switch from low-power modes into active modes of operation.

The DCO frequency, digitally controlled oscillator frequency can be adjusted by software. By writing appropriate values in various registers in the program, you can change the frequency. The default frequency after reset is 1.1 megahertz. So if you do not do anything and you just reset your microcontroller, you do not have to make any selections, the value of the DCO frequency will be 1.1 megahertz.

(Refer Slide Time: 07:28)



And this is the source of the clock, as you see this part in the block diagram here, lower side is highlighting the DCO clock and all these names that you see here, these refer to various bits in various registers that we will see very soon, which allow you to change the frequency of the DCO oscillator, digitally controlled oscillator.

Further, you have a multiplexer here, as we had seen in the simplified block diagram, which allows you to select the SM clock here as well as, as you see here, this signal goes and feeds to the master clock also. We come to that shortly but so this is the part about the source of clock, clock source that is the DCO.

(Refer Slide Time: 08:14)

**Clock Sources**

2. LFXT1

- Low or high frequency crystal oscillator.
- Used with low frequency watch crystal (32 kHz) ←
- Also used with a high frequency crystal (400 kHz to 16MHz)  
(Absent in MSP430G2553)

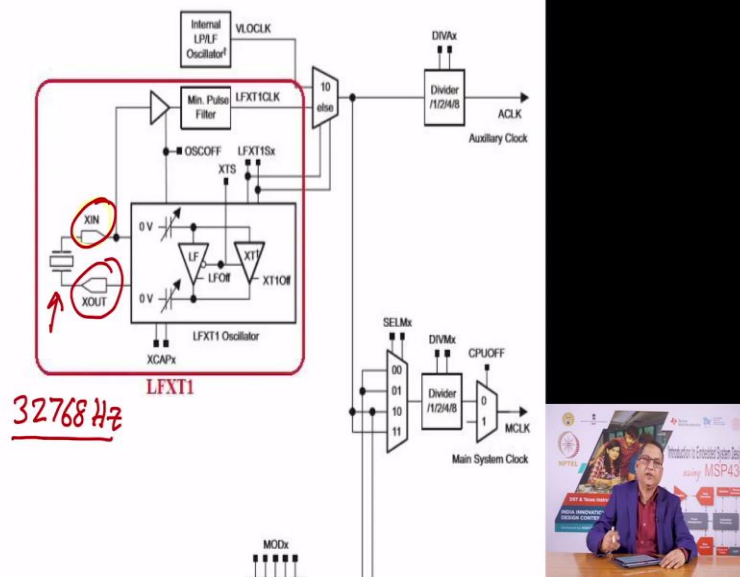
32768 Hz

The slide features a dark blue background with a white curved shape on the right side. A small inset image in the bottom right corner shows a man in a blue shirt 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 'MSP430'.

Then, apart from the DCO, we also have the low-frequency crystal oscillator. You can use a low or a higher frequency crystal, but for the G255 series, you are restricted to a low-frequency oscillator and that is recommended at 32 kilohertz. And in fact, the exact frequency is 32768 hertz. This is a crystal used in real-time clocks and so is very commonly available, and this is low-frequency, which is used with for the low-frequency crystal oscillator, which is inside, the oscillator is inside the microcontroller, this crystal has to be connected to external pins X, as we have seen.

If you had a different microcontroller, apart from other than this, you could use a higher frequency crystal also.

(Refer Slide Time: 09:13)



32768 Hz

Here is the part highlighting the low-frequency crystal oscillator. These are the XIN and XOUT pins onto which you, between which you connect a crystal, this is the crystal. And in the case of our current microcontroller MSP430G2553, this crystal can be only 32768 hertz. It can be other frequencies also. But it just so happens that you will have to get such a crystal custom made. This is the most common commercially available crystal.

(Refer Slide Time: 09:47)

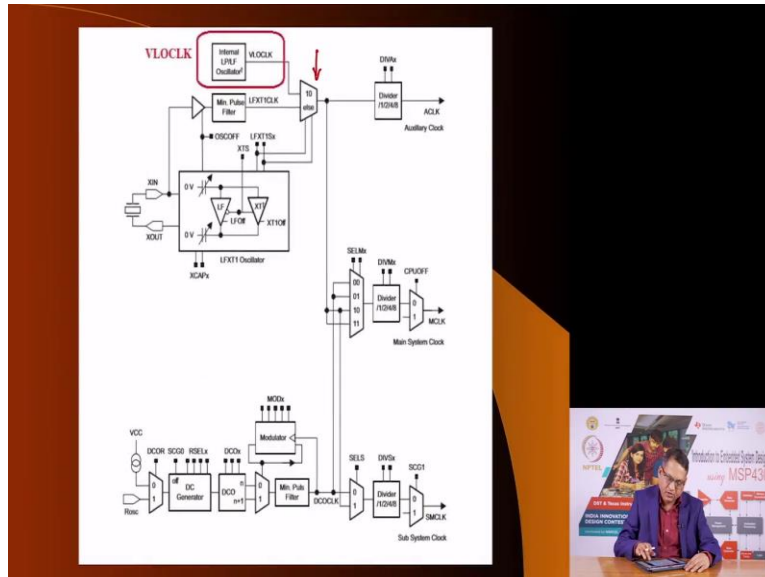
## Clock Sources

- 3. VLO
  - Internal very low-power, low-frequency oscillator
  - Typical frequency 12kHz

The third source of clock is very low-frequency oscillator. It is also internal RC based oscillator, and the typical frequency operation is just a mere 12 kilohertz, and you can imagine that from 16

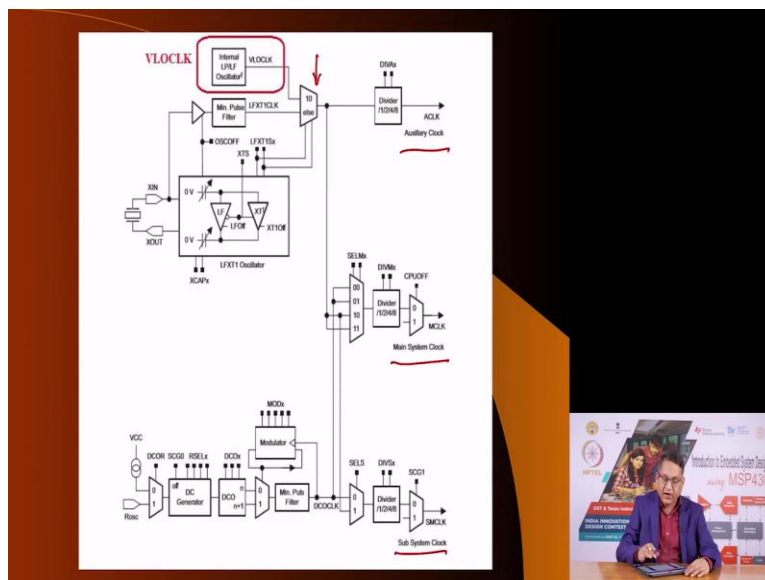
megahertz for the DCO, down to 12 kilohertz here, you can really change the frequency operation and therefore change the way the microcontroller performs, functions and the amount of power that it consumes.

(Refer Slide Time: 10:22)



Here is the source, clock source, and as you see it feeds a multiplexer here.

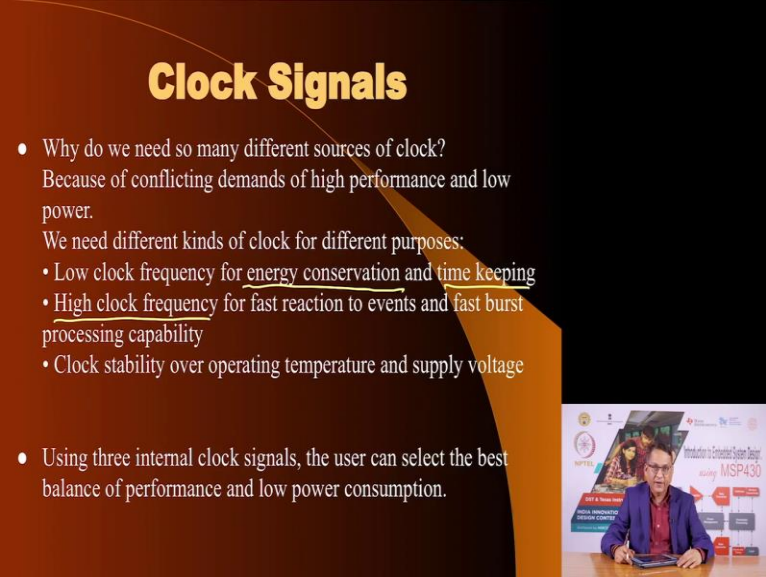
(Refer Slide Time: 10:36)



Now the question is, we have seen here that we need auxiliary clock, we need a main system clock, and we need a subsystem clock.



(Refer Slide Time: 10:47)



## Clock Signals

- Why do we need so many different sources of clock?  
Because of conflicting demands of high performance and low power.  
We need different kinds of clock for different purposes:
  - Low clock frequency for energy conservation and time keeping
  - High clock frequency for fast reaction to events and fast burst processing capability
  - Clock stability over operating temperature and supply voltage
- Using three internal clock signals, the user can select the best balance of performance and low power consumption.

Production & Content: Sreeniwas Reddy using MSP430

What is the purpose of having multiple clock signals on a microcontroller MSP430? And the reason is low-frequency operation is best for energy conservation, it is also good for timekeeping. The high clock frequency will allow you to react to external events in a short time and therefore that is beneficial when you want to respond quickly. And if you want a stable clock, then having a crystal-based oscillator is the best option and MSP430 allows you all the three options.


It offers you very-low-frequency RC based oscillator, it allows you 32.768-kilohertz crystal-based oscillator, and it allows you another RC based oscillator which whose frequency can be digitally altered to go from 60 kilohertz on one end to 16 megahertz on the other side, and so you could select that if you would like to have fast response time. And that is the reason why multiple sources have been provided and the clock signals can be derived from these clock sources.

(Refer Slide Time: 12:01)

# Clock Signals

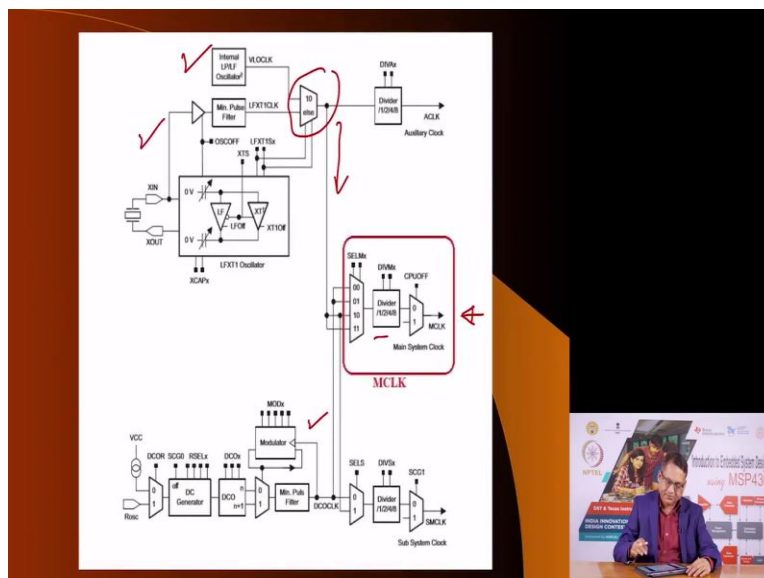
1. **MCLK: Master clock**

- Used by the CPU and a few peripherals.
- Supplied by the DCO with a frequency of around 1.1MHz. Stabilized by PLL.
- MCLK is software selectable as LFXT1CLK, VLOCLK, DCOCLK (or XT2CLK, but not present on MSP430G2553).



What are the signals that we need? As we mentioned, one of them is master clock. It is used by CPU. CPU has only one clock signal and that is master clock. Master clock signal can also be fed to other peripherals. As I mentioned, after reset the default master clock is derived from the DCO with a frequency of 1.1 megahertz, but you can select the master clock signal to come from low-frequency crystal oscillator, to come from VLO or the DCO and XT2, that is high-frequency crystal 2, but this option is not available on our microcontroller, this particular microcontroller part.

(Refer Slide Time: 12:47)



This is the part where you are selecting the master system clock here, and as you see, you have a divider which allows you to further divide the frequency, you have a multiplexer which allows you to select which source, clock source can be used. A master clock can come from here and this itself allows you to select either the VLO or the low-frequency crystal and the other part of this multiplexer is being fed by the DCO. So master clock can be derived out of VLO or low-frequency crystal or DCO.

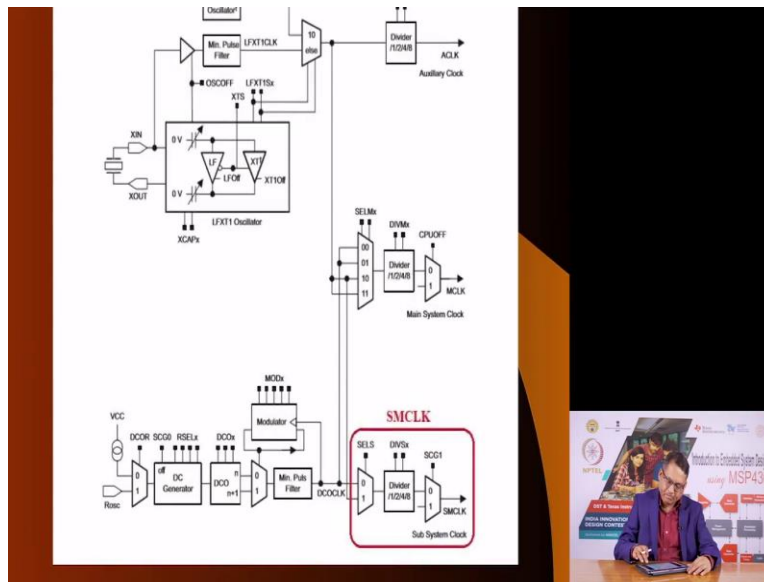
(Refer Slide Time: 13:28)

## Clock Signals

- 2. **SMCLK**: Sub-system Master Clock
  - - Distributed to peripherals.
    - Often same as MCLK. ✓
    - Supplied by the DCO with a frequency of around 1.1MHz. Stabilized by PLL. ✓
    - SMCLK is software selectable as LFXT1CLK, VLOCLK, DCOCLK (or XT2CLK, but not present on MSP430G2553). ✓

Then the other signal that we need is the subsystem master clock. It is distributed only to peripherals, this is not fed to the CPU. Oftentimes it is the same as master clock and if at reset, the value is from the DCO and the frequency is 1.1 megahertz and again, it can be selected from low-frequency, it can be sourced from the low-frequency crystal oscillator or the VLO oscillator or of course, the DCO clock.

(Refer Slide Time: 14:04)



Here is the part about subsystem master clock.

(Refer Slide Time: 14:08)

## Clock Signals

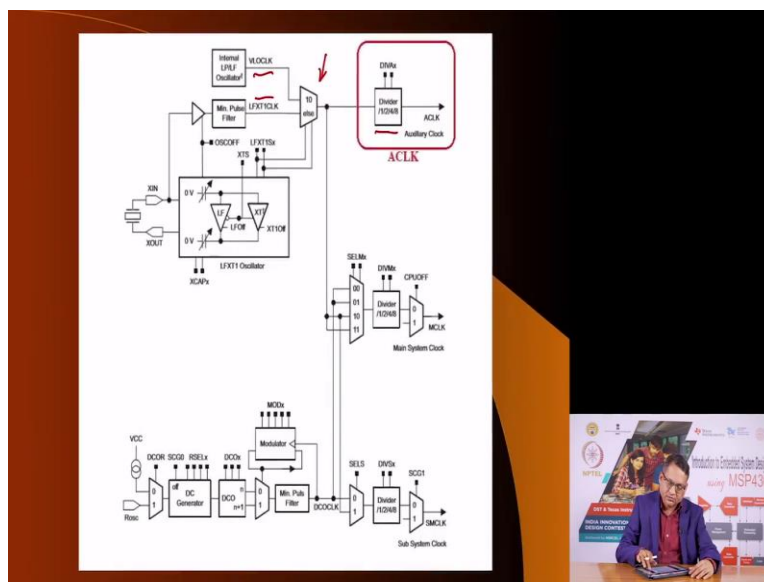
### 3. ACLK: Auxiliary Clock

- Distributed among peripherals. ✓
- Sourced from LFX1TCLK or VLOCLK.
- Typically much slower and usually  $\leq 32\text{kHz}$ .
- ACLK is software selectable as LFX1TCLK or VLOCLK.

And the third is the auxiliary clock. This is again, only distributed to peripherals and the source of auxiliary clock can only be the low-frequency crystal or the internal low-frequency oscillator. If, now both these oscillators are slow oscillators meaning when you turn, apply power to them, they do not start oscillating quickly. They take some time to build up and for the oscillations to stabilize. And because of this, if any peripheral is being fed from the auxiliary clock, you must make sure that the oscillators have stabilized.

If the oscillator has not stabilized, the microcontroller will not connect the clock source to the clock signal and therefore to the peripheral, and this we will see later how we can detect whether the oscillator is stable or not, and if it is not stable, we can wait. And why we can wait? Because the CPU which decides which clock is to be fed to which peripheral, the CPU is being fed by master clock and we can choose the master clock to be from the DCO so it can continue to perform, the microcontroller can continue to run the program, but in that program, you can wait for these oscillators to stabilize before you apply to any peripherals.

(Refer Slide Time: 15:39)



Here is the selection for the auxiliary clock. Again, it has a divider, as you see you can divide the source, which is either this or this by 1, 2, 4, or 8.

(Refer Slide Time: 15:51)

**Calibrated Frequencies of DCO**

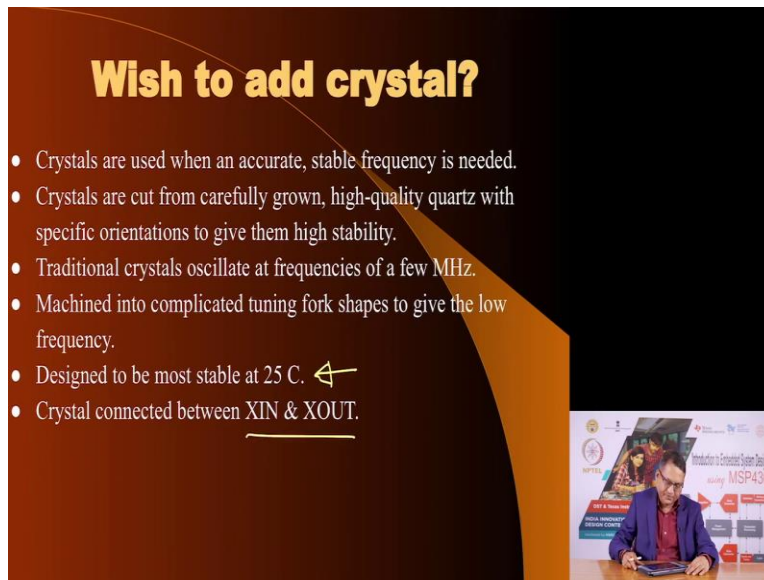
- 4 Calibrated frequencies:
  - 1 MHz ✓
  - 8 MHz ✓
  - 12 MHz ✓
  - 16 MHz ✓
- Sample CCS Statements:
  - BCSCTL1 = CALBC1\_1MHZ; (For range selection)
  - DCOCTL = CALDCO\_1MHZ; (For Freq. selection)

The slide also features a small inset image of a person sitting at a desk with a laptop, and a book titled 'Introduction to Embedded System Design using MSP430'.

Now, in the DCO, apart from the ability to vary the frequencies, it also has four calibrated frequencies and those are 1 megahertz, 8, 12, and 16. And you can write values into appropriate registers to select whatever frequency you want. This is a sample code that two registers, one of them is called BCS control, the other is called DCO control, by writing appropriate values, these are basically bitmasks, by writing this into these two registers will result in a DCO frequency of 1 megahertz calibrated frequency. Not 1.1 megahertz, but quite accurate 1 megahertz frequency.

Now, sometimes and often times when you want to measure time accurately, you may not want to use the DCO, you may want to use the low-frequency crystal because a crystal is a very stable and accurate source of frequency, in which case you have the option of selecting the low-frequency crystal oscillator. Traditionally, oscillators, crystal oscillators offer you large range from 32 kilohertz up to few tens of megahertz.

(Refer Slide Time: 17:18)



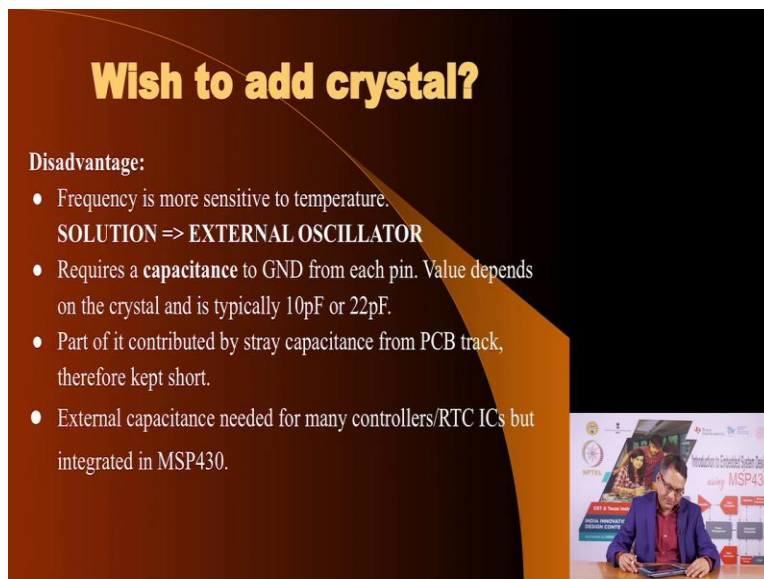
## Wish to add crystal?

- Crystals are used when an accurate, stable frequency is needed.
- Crystals are cut from carefully grown, high-quality quartz with specific orientations to give them high stability.
- Traditional crystals oscillate at frequencies of a few MHz.
- Machined into complicated tuning fork shapes to give the low frequency.
- Designed to be most stable at 25 C. ←
- Crystal connected between XIN & XOUT.

The slide features a dark background with a light-colored curved shape on the right. A small inset image in the bottom right corner shows a man in a blue jacket 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 'SITA UNIVERSITY CHENNAI'.

In our MSP430G255 series, we are restricted to 32-kilohertz crystal and it is designed to give you the printed frequency at 25 degree centigrades. In the case of MSP430, it has to be connected between these two pins that is XIN and XOUT.

(Refer Slide Time: 17:28)



## Wish to add crystal?

**Disadvantage:**

- Frequency is more sensitive to temperature.

**SOLUTION => EXTERNAL OSCILLATOR**

- Requires a **capacitance** to GND from each pin. Value depends on the crystal and is typically 10pF or 22pF.
- Part of it contributed by stray capacitance from PCB track, therefore kept short.
- External capacitance needed for many controllers/RTC ICs but integrated in MSP430.

The slide features a dark background with a light-colored curved shape on the right. A small inset image in the bottom right corner shows a man in a blue jacket 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 'SITA UNIVERSITY CHENNAI'.

Where do you use crystals? When you want to measure time or when you want to measure frequencies of events then you should consider using a crystal. A crystal oscillator also requires capacitors. Some capacitance is already on the microcontroller but you, depending upon the recommendations of the datasheet of the crystal that you use, if it requires more capacitance,

then it can be connected to the X, these two pin XIN and XOUT pins. Extra capacitance on these pins as the requirement may be for a particular crystal that you choose.

(Refer Slide Time: 18:12)

## Clock Registers

- DCOCTL (DCO Control Register)
- BCSCCTL1 (Basic Clock System Control Register 1)
- BCSCCTL2 (Basic Clock System Control Register 2): For MCLK & SMCLK manipulation
- BCSCCTL3 (Basic Clock System Control Register 3): For external crystal and capacitor selections



Okay. Now that we have seen the sources of clock and the clock signals that we need, here are the registers which allow us to select various sources, decide their frequencies, and route these clock sources to appropriate auxiliary clock or master clock or subsystem master clock.


(Refer Slide Time: 18:38)

## DCOCTL: DCO Control Register

DCOx				MODx			
7	6	5	4	3	2	1	0
rw-0	rw-1	rw-1	rw-0	rw-0	rw-0	rw-0	rw-0

**DCOx**  
Bits 7:5 DCO frequency select. These bits select which of the eight discrete DCO frequencies within the range defined by the RSELx setting is selected.

**MODx**  
Bits 4:0 Modulator selection. These bits define how often the  $f_{DCO}$  frequency is used within a period of 32 DCOCLK cycles. During the remaining clock cycles (32-MOD) the  $f_{MOD}$  frequency is used. Not usable when DCOx = 7.





The most important register is the DCO control register, and it has two sets of bits. One is called the DCO bit and these are three bits here, and the, you have five bits here, which are the mod bits. The DCO bit selects a frequency, broad frequencies which are dictated by another set of bits, which is which are these bits which we will see in the other register that we will briefly see.

These RSEL bits allow you to go from 16 kilohertz to 16 mega, 60 kilohertz to 16 megahertz, and within that, these bits will tell you what particular frequency you can operate at. If you want to have a frequency selection even finer than you can get with DCO, DCOx, then you have to play with the MODx bits. For introductory applications, for simple applications, you do not have to worry about this.


(Refer Slide Time: 19:40)

## BCSCTL1: Basic Clock Control Register 1

7	6	5	4	3	2	1	0
XTZOFF	XTS <sup>(1)</sup>	DIVA <sub>x</sub>			RSEL <sub>x</sub>		
rw(1)	rw(0)	rw(0)	rw(0)	rw(0)	rw-1	rw-1	rw-1
<p><b>XTZOFF</b> Bit 7</p> <p>XTZ off. This bit turns off the XT2 oscillator.</p> <p>0 XT2 is on</p> <p>1 XT2 is off if it is not used for MCLK or SMCLK.</p>	<p><b>XTS</b> Bit 6</p> <p>LFXT1 mode select.</p> <p>0 Low-frequency mode</p> <p>1 High-frequency mode</p>	<p><b>DIVA<sub>x</sub></b> Bits 5-4</p> <p>Divider for ACLK</p> <p>00 /1</p> <p>01 /2</p> <p>10 /4</p> <p>11 /8</p>			<p><b>RSEL<sub>x</sub></b> Bits 3-0</p> <p>Range select. Sixteen different frequency ranges are available. The lowest frequency range is selected by setting RSEL<sub>x</sub> = 0. RSEL<sub>3</sub> is ignored when DCOR = 1.</p>		

(1) XTS = 1 is not supported in MSP430C20xx and MSP430G2xx devices (see Figure 5-1 and Figure 5-2 for details on supported settings for all devices).

(2) This bit is reserved in the MSP430AFE2xx devices.



Now, this is the second important register, Basic Clock Control Register. Here, I want to make a point. Do you notice that some of these bits here, it says, when it says rw means these bits can be written to as well as read from? And then it says, rw, here it says rw-0 and here it says rw-0 within a bracket.

These two refer to different values of zero depending upon the source of reset and we will consider this when we consider the reset case, reset facilities in MSP430 but I just wanted to bring that to your notice. In this register, basic clock control register, we do not have this option because as I mentioned, it does not offer you high-frequency crystal, you can select this, only this option is available, and using this you can decide whether you want to which divider you


want to use for the auxiliary clock. And these are the RSEL bits, these four bits, which decide broad frequency ranges that run the DCO oscillator.

(Refer Slide Time: 20:59)

## BCSCTL2: Basic Clock Control Register 2

7	6	5	4	3	2	1	0
SELMx		DIVMx		SELS	DIVSx		DCOR <sup>(1)(2)</sup>
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0
<b>SELMx</b> Bits 7-6 Select MCLK. These bits select the MCLK source.							
	00	DCOCLK					
	01	DCOCLK					
	10	XT2CLK when XT2 oscillator present on-chip. LFX1CLK or VLOCLK when XT2 oscillator not present on-chip.					
	11	LFX1CLK or VLOCLK					
<b>DIVMx</b> Bits 5-4 Divider for MCLK							
	00	/1					
	01	/2					
	10	/4					
	11	/8					
<b>SELS</b> Bit 3 Select SMCLK. This bit selects the SMCLK source.							
	0	DCOCLK					
	1	XT2CLK when XT2 oscillator present. LFX1CLK or VLOCLK when XT2 oscillator not present					
<b>DIVSx</b> Bits 2-1 Divider for SMCLK							
	00	/1					
	01	/2					
	10	/4					
	11	/8					
<b>DCOR</b> Bit 0 DCO resistor select. Not available in all devices. See the device-specific data sheet.							
	0	Internal resistor					
	1	External resistor					

(1) Does not apply to MSP430G20xx or MSP430G21xx devices.  
(2) This bit is reserved in the MSP430AFE2xx devices.



Then you have the Basic Clock Control Register 2. Here this decides whether what is the source of the master clock, meaning these bits, these two bits will decide will select the multiplexer, which is feeding the master clock source. This will, these two bits will decide what sort of divider do you want to use for the master clock source.

This will decide whether do you want, how do you select the sub-master clock and you only have this option here because this is not available on our MSP430G2553. And then these two bits will allow you to choose the divider for SM clock. And then this bit allows you whether you want to have an external resistor. This is not available on our G2553 series.

(Refer Slide Time: 22:08)

### Register 3

7	6	5	4	3	2	1	0
XT2Sx	LFX1Sx <sup>(1)</sup>		XCAPx <sup>(2)</sup>		XT2OF <sup>(1)</sup>	LFX1OF <sup>(2)</sup>	
no-0	no-0		no-0		no-1	i(1)	

**XT2Sx** Bits 7:6 XT2 range select. These bits select the frequency range for XT2.  
00 0.4- to 1-MHz crystal or resonator  
01 1- to 3-MHz crystal or resonator  
10 3- to 16-MHz crystal or resonator  
11 Digital external 0.4- to 16-MHz clock source


**LFX1Sx** Bits 5:4 Low-frequency clock select and LFX1T1 range select. These bits selected between LFX1T1 and VLO when XTS = 0, and select the frequency range for LFX1T1 when XTS = 1.  
When XTS = 0:  
00 32768-Hz crystal on LFX1T1  
01 Reserved  
10 VLOCLK (Reserved in MSP430F21x1 devices)  
11 Digital external clock source  
When XTS = 1 (Not applicable for MSP430x20x devices, MSP430G2xx(1/2))  
00 0.4- to 1-MHz crystal or resonator  
01 1- to 3-MHz crystal or resonator  
10 3- to 16-MHz crystal or resonator  
11 Digital external 0.4- to 16-MHz clock source  
LFX1Sx definition for MSP430AFE2xx devices  
00 Reserved  
01 Reserved  
10 VLOCLK  
11 Reserved

**XCAPx** Bits 3:2 Oscillator capacitor selection. These bits select the effective capacitance seen by the LFX1T1 crystal when XTS = 0. If XTS = 1 or if LFX1Sx = 11 XCAPx should be 00.  
00 -1 pF  
01 -4 pF  
10 -10 pF  
11 -12.5 pF

**XT2OF** Bit 1 XT2 oscillator fault  
0 No fault condition present  
1 Fault condition present

**LFX1OF** Bit 0 LFX1T1 oscillator fault  
0 No fault condition present  
1 Fault condition present

(1) MSP430G22x0. The LFX1Sx bits should be programmed to 10b during the initialization and start-up code to select VLOCLK (for more details refer to Digital I/O chapter). The other bits are reserved and should not be altered.  
(2) This bit is reserved in the MSP430F21x1 device.




This is the third register, Basic Control Register, Clock Control Register 3, and here you have several bits. But the most important bit is these, which allow you to choose what sort of frequency crystal you want to use and whether you want to use internal capacitance, what will be the value of these capacitance.

(Refer Slide Time: 22:32)

### Setting the frequency of the DCO

The frequency of DCOCLK is set by the following functions:


- The four RSELx bits select one of sixteen nominal frequency ranges for the DCO. These ranges are defined in the datasheet.
- The three DCOx bits divide the DCO range selected by the RSELx bits into 8 frequency steps, separated by approximately 10%.
- The five MODx bits, switch between the frequency selected by the DCOx bits and the next higher frequency set by DCOx+1. When DCOx = 07h, the MODx bits have no effect because the DCO is already at the highest setting for the selected RSELx range.



Now, to set the frequency of the DCO, you have to write into the DCO register as well as the second register, that is Basic Clock Control Register 1 and I suggest that you go through this slide to understand all the options.

(Refer Slide Time: 22:52)

## Setting the frequency of the DCO



TEXAS  
INSTRUMENTS


MSP430G2x53  
MSP430G2x13

www.ti.com | SLA0736-APRIL 2011-REVISED MAY 2013

### DCO Frequency

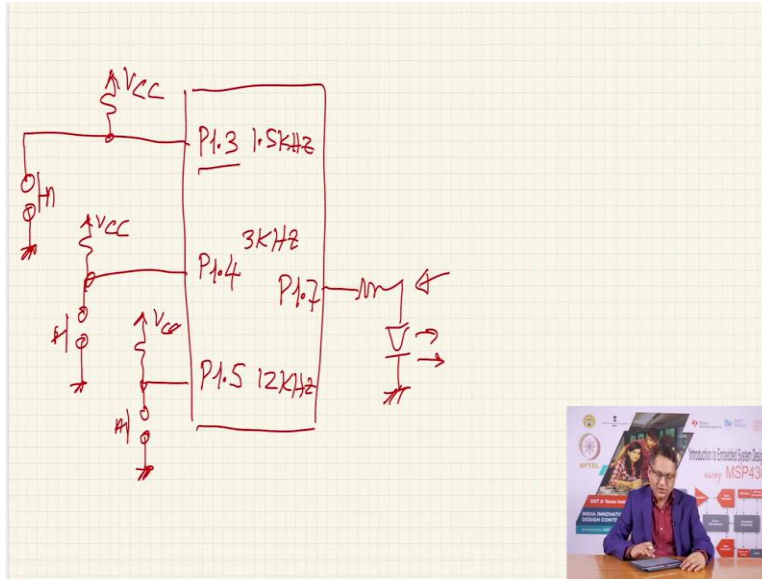
over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted)

PARAMETER	TEST CONDITIONS	V <sub>CC</sub>	MIN	TYP	MAX	UNIT	
V <sub>CC</sub>	Supply voltage	RSELx = 14	1.8	2.0	2.2	V	
		RSELx = 14	2.2	2.6	3.0		
		RSELx = 15	3	3.6	4.2		
f <sub>DCO0.0</sub>	DCO frequency (0, 0)	RSELx = 0, DCOx = 0, MODx = 0	0 V	0.06	0.14	MHz	
f <sub>DCO0.3</sub>	DCO frequency (0, 3)	RSELx = 0, DCOx = 0, MODx = 0	0 V	0.07	0.17	MHz	
f <sub>DCO0.9</sub>	DCO frequency (1, 3)	RSELx = 1, DCOx = 0, MODx = 0	0 V	0.15		MHz	
f <sub>DCO1.8</sub>	DCO frequency (2, 3)	RSELx = 2, DCOx = 0, MODx = 0	0 V	0.21		MHz	
f <sub>DCO3.6</sub>	DCO frequency (3, 3)	RSELx = 3, DCOx = 0, MODx = 0	0 V	0.30		MHz	
f <sub>DCO4.5</sub>	DCO frequency (4, 3)	RSELx = 4, DCOx = 0, MODx = 0	0 V	0.41		MHz	
f <sub>DCO5.4</sub>	DCO frequency (5, 3)	RSELx = 5, DCOx = 0, MODx = 0	0 V	0.50		MHz	
f <sub>DCO6.3</sub>	DCO frequency (6, 3)	RSELx = 6, DCOx = 0, MODx = 0	0 V	0.54	1.08	MHz	
f <sub>DCO7.2</sub>	DCO frequency (7, 3)	RSELx = 7, DCOx = 0, MODx = 0	0 V	0.60	1.50	MHz	
f <sub>DCO8.1</sub>	DCO frequency (8, 3)	RSELx = 8, DCOx = 0, MODx = 0	0 V	1.0		MHz	
f <sub>DCO9.0</sub>	DCO frequency (9, 3)	RSELx = 9, DCOx = 0, MODx = 0	0 V	2.3		MHz	
f <sub>DCO10.9</sub>	DCO frequency (10, 3)	RSELx = 10, DCOx = 0, MODx = 0	0 V	3.4		MHz	
f <sub>DCO11.8</sub>	DCO frequency (11, 3)	RSELx = 11, DCOx = 0, MODx = 0	0 V	4.25		MHz	
f <sub>DCO12.7</sub>	DCO frequency (12, 3)	RSELx = 12, DCOx = 0, MODx = 0	0 V	4.30	7.30	MHz	
f <sub>DCO13.6</sub>	DCO frequency (13, 3)	RSELx = 13, DCOx = 0, MODx = 0	0 V	6.00	7.8	9.60	MHz
f <sub>DCO14.5</sub>	DCO frequency (14, 3)	RSELx = 14, DCOx = 0, MODx = 0	0 V	6.60	13.0	MHz	
f <sub>DCO15.4</sub>	DCO frequency (15, 3)	RSELx = 15, DCOx = 0, MODx = 0	0 V	13.0	18.0	MHz	
f <sub>DCO16.3</sub>	DCO frequency (16, 7)	RSELx = 15, DCOx = 7, MODx = 0	0 V	16.0	26.0	MHz	
f <sub>step</sub>	Frequency step between range RSEL and RSEL+1	f <sub>step</sub> = f <sub>DCO(RSEL+1)</sub> - f <sub>DCO(RSEL)</sub>	0 V	1.30		ratio	
f <sub>step</sub>	Frequency step between two DCOs and DCOx+1	f <sub>step</sub> = f <sub>DCO(RSEL, DCOx+1)</sub> - f <sub>DCO(RSEL, DCOx)</sub>	0 V	1.08		ratio	
Duty cycle	Measured at SMC1 output						



Based on the bits that you write into these registers, you see, this is part of the datasheet that allows you to change the frequency from 60 kilohertz to 16 megahertz. Now we want to illustrate the great flexibility that MSP430 offers by dynamically changing the clock frequency and what are we illustrating here? Let me show our plan.

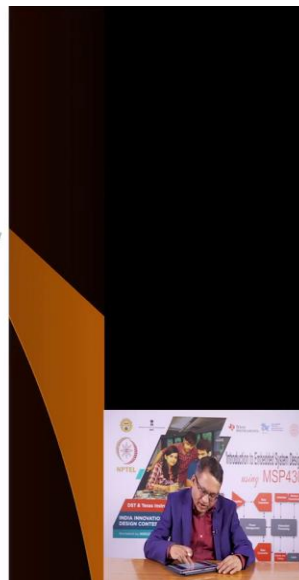
(Refer Slide Time: 23:26)

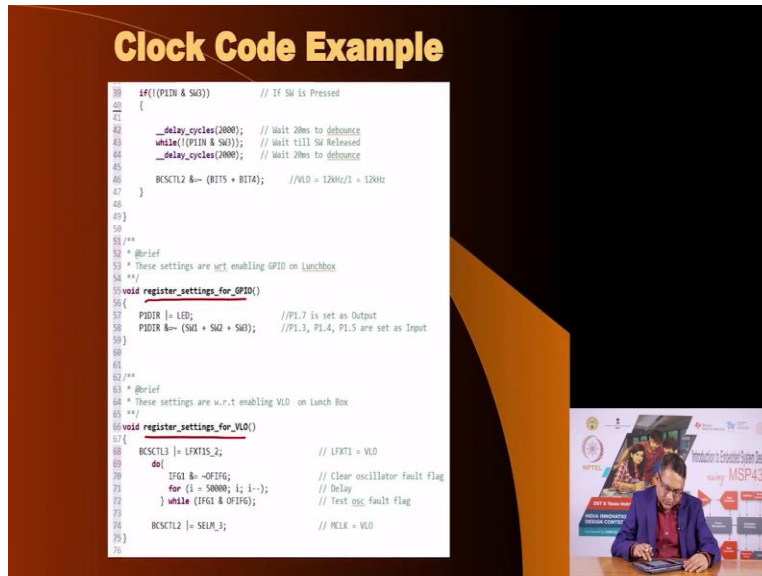


```

2
3 #define LED BIT7
4
5 #define SW1 BIT3 // 1.5 kHz
6 #define SW2 BIT4 // 3 kHz
7 #define SW3 BIT5 // 12 kHz
8
9 volatile unsigned int i; // Volatile to prevent removal
10
11 /**
12  * @brief
13  * Function to take input from 3 switches and change CPU clock accordingly
14  */
15 void switch_input()
16 {
17
18     if(!(P1IN & SW1) // If SW1 is Pressed
19     {
20         __delay_cycles(2000); // Wait 20ms to debounce
21         while(!(P1IN & SW1)); // Wait till SW Released
22         __delay_cycles(2000); // Wait 20ms to debounce
23
24         BCSCCTL2 &=~ (BIT5 + BIT4); //Reset VLO divider
25         BCSCCTL2 |= (BIT5 + BIT4); //VLO = 12kHz/8 = 1.5kHz
26     }
27
28
29     if(!(P1IN & SW2) // If SW2 is Pressed
30     {

```





What we are doing is we are going to take a MSP430. As you know that there are three sources VLO, crystal low-frequency crystal oscillator, and DCO. We are going to use, select the very low-frequency oscillator for the master clock, which means the processor will operate at 12 kilohertz but the 12-kilohertz basic frequency can be divided by 1 or 2 or 4 or 12, which means by selecting an appropriate divider, we can reduce the frequency operation for the CPU. Because we are, I am saying we will select the master clock from master clock signal will come from VLO.

What we want to show is as follows, we want to have a LED like this, and in fact, we are going to use the existing LED on the lunchbox, but we want to connect three external switches with pull-up resistors, one more, here VCC. And where are these connected? Where are these three switches connected? They are connected to P1.3, 1.4, and 1.5. So let me write here, P1.3 pin, P1.4, and P1.5. And we, as we know, the LED is connected to P1.7.

What we will, what do we hope to achieve? That we will start the oscillator, we will select the VLO to go through the multiplexer and provide the signal for the master clock, and then we will have a program which will continuously poll these three switches and if one switch is, if this switch is pressed, for example, it will divide the VLO frequency by 1. If the second switch is pressed, it will divide it by 4, and if the third switch is pressed, it will divide it by 8.

And thereby you will get these three resultant frequencies. You see? If you divide by 1, the source is 12 kilohertz, therefore the frequency will be 12 kilohertz. The CPU frequency that is

master clock frequency will be 12 kilohertz. If you divide it by, if you choose the divide by 4 option, you will get 3 kilohertz, and if you choose the 8, you will get 1.5 kilohertz.

And you can, how would you know that the oscillator, that the CPU is working at different frequencies? Well, what we will do is we are going to blink this LED, we are blinking this LED at a certain rate which is derived out of the clock frequency. Basically what we are saying is turn the LED on for some clock cycles and turn the LED off for some clock cycles.

Now, if the frequency of operation reduces you will see that the duration of the on and off increases, that is frequency goes down. And so by operating it at, let us see, P1.5 is, here you get a 1.5-kilohertz clock. Here you get 3-kilohertz clock and here you get 12-kilohertz clock. Of course, we are further used delays so that the LED will be blinking in visible range, you can make, you can observe that the LED is turning on and off.

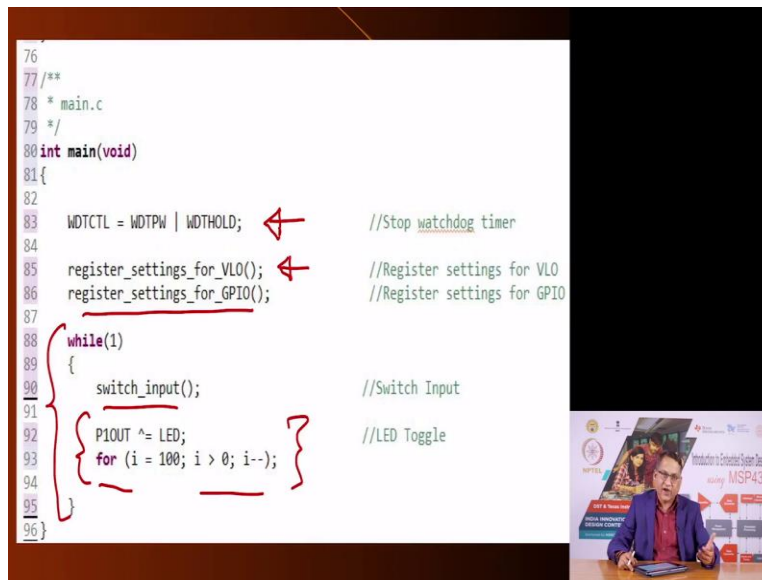
It is not that it is turning on and off at such a high rate that it is beyond the persistence of vision. No. We have written the code in the manner that it is, you can see that it is turning on and off, but the rate of blinking will be perceptibly different when you choose these clock frequencies.

So let us go through the code to understand how it works. At the beginning we see, as usual, we have included the header file, then we have defined that we are going to put the LED on bit 7, but bit 7 is not telling the actual port pin, bit 7 is simply a mask bit as we have seen. Then we are putting one switch at bit 3, bit 4, and bit 5, and these are all P1.X. So here it is P1.7, 1.7, 1.3, 1.5.

Then we have a function which we call as switch input, I will come to that later. We have another function called register settings for GPIO, I will come to that also. And we have third function, which is called register settings for the VLO.

(Refer Slide Time: 29:00)

```
76
77 /**
78  * main.c
79  */
80 int main(void)
81 {
82
83     WDCTL = WDTPW | WDTOLD; //Stop watchdog timer
84
85     register_settings_for_VLO(); //Register settings for VLO
86     register_settings_for_GPIO(); //Register settings for GPIO
87
88     while(1)
89     {
90         switch_input(); //Switch Input
91
92         P1OUT ^= LED; //LED Toggle
93         for (i = 100; i > 0; i--);
94     }
95 }
96 }
```



And then we have the main program, so the main program is really very simple. What is it doing? The, when you reset the system, the first instruction that is executed is to stop the watchdog timer. We do not want to be bothered by watchdog timer overflowing and resetting us, which we will see in the next part of this lecture.

Then we are calling this subroutine, which is basically selecting the VLO, and based on the switches, it is going to select a particular frequency operation, and then we are calling another subroutine where we are deciding the direction of the pins of the microcontroller. The P1.7 pin has to be output pin and the other three pins have to be input pin that will be done in this second function.

And then we have a infinite loop, while 1, and we are saying read the switch. So I am going to execute this function in which I am reading, waiting whether switch one is pressed or two is pressed or three is pressed. If a particular switch is pressed, it will go and change the frequency operation of the VLO oscillator and will come back here, and then it is going to, so this part of the code is simply toggling the LED.

So you toggle it once, go back, wait for the switch to be pressed if it is not pressed, you come back, again toggle the LED. So you keep on doing it, which means most of the time you are toggling. But after every toggle, you are going and checking whether the any switch has been




pressed. If any switch has been pressed, you wait for it to be released and then based on which switch was pressed you are changing the VLO frequency.


And so after that when it comes here, you will see that the LED toggle rate changes. Why? Because the CPU clock itself has changed and this is therefore a great example to illustrate how MSP430 offers you dynamic clock stability. Let us go through the code again, switch input.

(Refer Slide Time: 31:12)

```
11/**
12 * @brief
13 * Function to take input from 3 switches and change CPU clock accordingly
14 **/
15 void switch_input()
16 {
17     if(!(P1IN & SW1)) // If SW1 is Pressed
18     {
19         _delay_cycles(2000); // Wait 20ms to debounce
20         while(!(P1IN & SW1)); // Wait till SW Released
21         _delay_cycles(2000); // Wait 20ms to debounce
22     }
23     BCSCCTL2 &=~ (BITS5 + BIT4); //Reset VLO divider
24     BCSCCTL2 |= (BITS5 + BIT4); //VLO = 12kHz/8 = 1.5kHz
25 }
26
27
28
29 if(!(P1IN & SW2)) // If SW2 is Pressed
30 {
31     _delay_cycles(2000); // Wait 20ms to debounce
32     while(!(P1IN & SW2)); // Wait till SW Released
33     _delay_cycles(2000); // Wait 20ms to debounce
34 }
35 BCSCCTL2 &=~ (BITS5 + BIT4); //Reset VLO divider
36 BCSCCTL2 |= (BIT4); //VLO = 12kHz/4 = 3kHz
37 }
38
```



```
48
49 }
50
51 /**
52 * @brief
53 * These settings are w.r.t enabling GPIO on Lunchbox
54 **/
55 void register_settings_for_GPIO()
56 {
57     P1DIR |= LED; // P1.7 is set as Output
58     P1DIR &=~ (SW1 + SW2 + SW3); // P1.3, P1.4, P1.5 are set as Input
59 }
60
61
62 /**
63 * @brief
64 * These settings are w.r.t enabling VLO on Lunch Box
65 **/
66 void register_settings_for_VLO()
67 {
68     BCSCCTL3 |= LFX1S_2; // LFX1 = VLO
69     do{ // Clear oscillator fault flag
70         IFG1 &= ~OFIFG; // Delay
71         for (i = 50000; i; i--); // Test osc fault flag
72     } while (IFG1 & OFIFG);
73
74     BCSCCTL2 |= SELM_3; // MCLK = VLO
75 }
76
```



Here we are waiting whether switch one is pressed, if it is pressed, this is to debounce and then you are selecting that the VLO divider should be by 8, therefore you will get a frequency of 1.5

kilohertz. If switch two is pressed, then you debounce it again and select this divider so that your resultant clock frequency CPU frequency is 3 kilohertz, and the third option is, if the third switch is pressed, your VLO will not be divided, it will be the same as the VLO frequency. Therefore, the CPU clock will be 12 kilohertz.

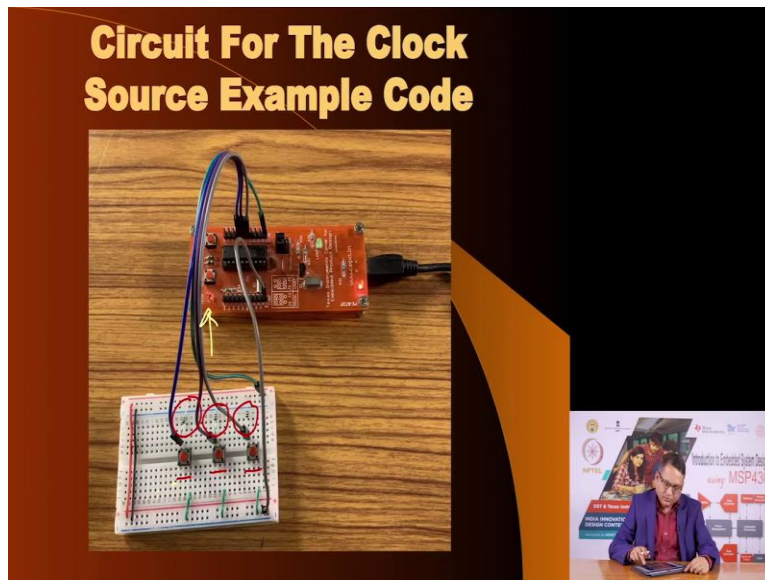
Then this is very simple, it simply turns the pin which is driving the LED as output and the other three pins P1.3, 4, and 5 as inputs. It does not do anything else and then this second function, which allows you to select various, select VLO as the source. As we saw earlier, you have to play in the Basic Clock Source Register 3 to select the VLO oscillator.

Now, as I mentioned, this is a slow oscillator, and therefore after power on it may not quickly turn on, and if you go through the datasheet of MSP430, it will tell you to wait for a certain amount of time before you can expect this clock to work.

And the way to check is to reset a particular flag in this register we, which we will see in the reset part, and you turn this bit to zero and wait for some time, and if this bit remains zero that means the oscillator is stable. If the oscillator is not working properly, the microcontroller will set OFIFG bit again. And so you are going to wait in this loop till this bit is reset to 0. If this bit remains reset to 0, that means the oscillator is stable.

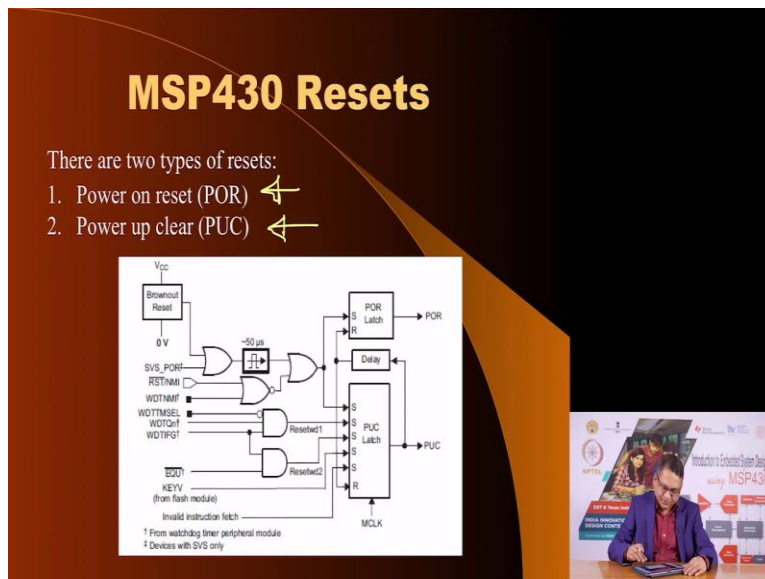
Now, you can make this setting will allow you to route the VLO oscillator as the master clock. So you are doing those two things. One, you are selecting VLO to come out of the first multiplexer, and then you are using the clock signal multiplexer to select master clock from VLO.

(Refer Slide Time: 34:01)



And so here is the implementation we have collected, as you see these are the three switches one, two, and three, these are the pull-up resistors, and this LED, the user LED on P1.7 will blink. So I suggest that you have downloaded this code, rebuild it and upload it into the lunchbox and see how this the blinking rate of this LED changes as you press this or this or the other one. The rate is quite perceptibly different and this is the proof that by, the program can decide what can be the frequency operation for the CPU, also, what can be the frequency operation of other peripherals.

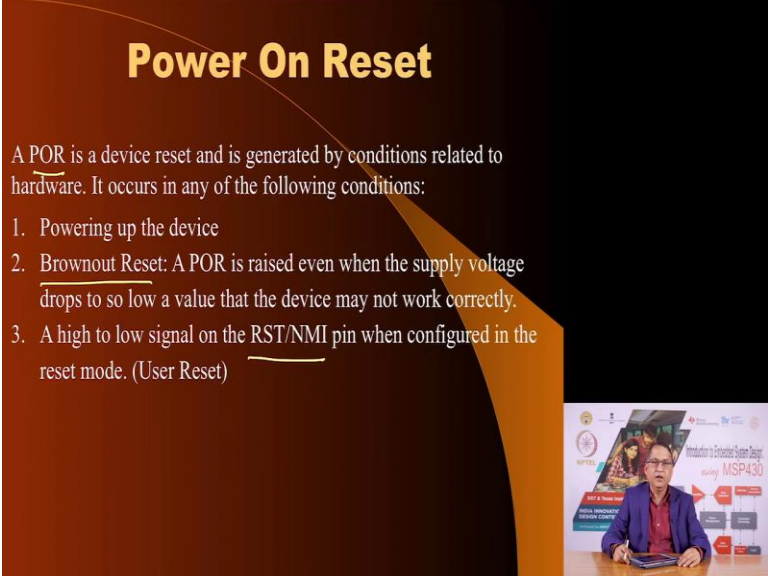
(Refer Slide Time: 34:49)



So this is as far as the clock frequency operation, the clock sources, and clock signals of MSP430 microcontroller are concerned, we have covered that. Now we are going to look at the reset part. Now, why does a microcontroller require reset? It is because a microcontroller is a logic circuit and logic circuit uses flip-flops and the flip-flops may have arbitrary values when power is first applied to them. And this may lead to non-uniform operation. And therefore it is very important that all the internal registers are initialized to a known value when the system is powered on for the first time. And that is the function of reset.

There are broadly two types of resets, one is called power-on reset, and the other is called power up reset. There is a fine difference between the two. Power-on reset happens, as the name suggests, when the power is applied for the first time. But power up clear can happen from other sources also.

(Refer Slide Time: 35:50)



**Power On Reset**

A POR is a device reset and is generated by conditions related to hardware. It occurs in any of the following conditions:

1. Powering up the device
2. Brownout Reset: A POR is raised even when the supply voltage drops to so low a value that the device may not work correctly.
3. A high to low signal on the RST/NMI pin when configured in the reset mode. (User Reset)

The slide includes a video inset in the bottom right corner showing a presenter in a blue jacket sitting at a desk with a laptop. Behind him is a presentation board with the title 'Introduction to Embedded System Design using MSP430' and various logos.

Now, power-on reset is generated whenever you turn power for the first time. It can also be generated because there is a brownout. Brownout means that the supply voltage to the microcontroller is not stable. If it dips below a certain value, it will reset the, it will generate a power-on reset for the microcontroller as well as if you press the reset NMI pin that will also generate a power-on reset signal.

Now, what reset the system? Is it possible to find out what was the source of reset? Was it brownout, was it a power-on reset that is turning the power off and on, or was it the user pressing


the switch on the RST/NMI pin? Is it possible to find out? Yes, MSP430 microcontroller has registers, which capture the source of that reset. And in fact, in this segment, we are going to write up code which will show which sources of reset was the reason the system was reset.

(Refer Slide Time: 37:04)

## Power Up Clear

A PUC is generated by software conditions. It is also always generated when a POR is generated, but a POR is not generated by a PUC. The following events trigger a PUC:


1. A POR signal
2. Watchdog timer expiration when in watchdog mode only ←
3. Watchdog timer security key violation (An attempt is made to write to the watchdog control register WDTCTL without the correct password 0x05A)
4. A Flash memory security key violation ←
5. A CPU instruction fetch from the peripheral address range of 0000h to 01FFh



## Power On Reset

A POR is a device reset and is generated by conditions related to hardware. It occurs in any of the following conditions:

1. Powering up the device ←
2. Brownout Reset: A POR is raised even when the supply voltage drops to so low a value that the device may not work correctly.
3. A high to low signal on the RST/NMI pin when configured in the reset mode. (User Reset)



Power up clear on the other hand, is generated because of software conditions. So the primary difference between power-on reset and power up clear is that power-on reset is because of external conditions. Why? You are powering the device up, so it generates a power-on reset or the supply voltage is not stable, so it generates a reset or a user presses a switch on the RST/NMI pin, these are all external events.

But power up clear is generated because internally something has happened and one of the reasons could be the watchdog timer. Of course, whenever a power-on reset signal happens, that also generates a power up signal, but additional to that, a watchdog timer or a security flash memory access violation or a CPU trying to fetch something from the peripheral address range, each of these four events could lead to a power up clear signal being generated.

(Refer Slide Time: 38:06)

**Device Initial Conditions After System Reset**

After a POR, the initial MSP430 conditions are:

- The RST/NMI pin is configured in the reset mode. **RST**
- I/O pins are switched to input mode.
- Other peripheral modules and registers are initialized as described in the user guide. **POR PUC**

The value is shown under each bit in the description of the registers. For example, **rw-0** means that a bit can be both read and written and is initialized to 0 after a PUC. Whereas, **rw-(0)** shows that a bit is initialized to 0 only after a POR; it retains its value through a PUC. **rw-0 rw-(0)**

What happens in them? Whether it is power-on or power up, the system is going to restart fetching the first instruction as pointed to by the reset vector. The after power-on reset, the RST/NMI pin is in the reset mode that is it works as RST pin, not NMI pin. The I/O pins are all switched to input mode and other peripheral modules and registers are initialized to a known value, which you should refer to the datasheet and the user guide for exact information.

Now, this is what I had mentioned earlier, each of the registers, when it is reset or when it is cleared, meaning when it is POR or PUC, will be indicated under that register. If it is written like this, that means, one, rw means it can be read and written and 0 means that the, after PUC the value will be 0. But if it is this, that means only on power-on reset the value will be 0, only on power-on reset, which means if the power up clear condition has been generated because of some internal event, the state of that bit, maybe 1 if prior to that event, this bit was 1, that is the difference.

So this is because of POR and this is because of PUC, this nomenclature that is rw-0 indicates a PUC condition, and rw-(0) means this is only achieved on a power-on reset condition, not power up clear condition.

(Refer Slide Time: 40:18)

**Device Initial Conditions After System Reset**

- Status register (SR) is reset. This means that the device operates at full power, even though it might have been in a low-power mode before the reset occurred.
- The watchdog timer powers up active in watchdog mode.
- Program counter (PC) is loaded with address contained at reset vector location (FFFEh).

==  
FFFE : FFFF

The slide features a dark background with a brown diagonal stripe. In the bottom right corner, there is a small inset image of a man in a blue jacket sitting at a desk with a laptop, with a presentation board behind him that includes the text 'Introduction to Embedded System Design using MSP430'.


After system reset, which means whether it is because of power-on reset or power up clear, the status register is set to 0 that means if you have been operating on, in low-power mode, you will come out of it. The watchdog timer becomes active in the watchdog mode, and the program counter is loaded with the reset vector location which is this and as we have mentioned, two locations are required for the address of the memory where our program is located. And these two addresses are FFFE and FFFF.

(Refer Slide Time: 41:02)

## Software Initialisations after System Reset

After a system reset, the software must:

- Initialize the watchdog timer, usually to turn it off
- Configure peripheral modules


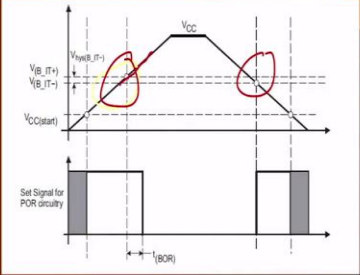


After the system reset, you must, if you wish, you must initialize the watchdog timer and usually you turn it off and you must configure the peripheral modules that is the responsibility of the program.

(Refer Slide Time: 41:20)

## Brownout Reset

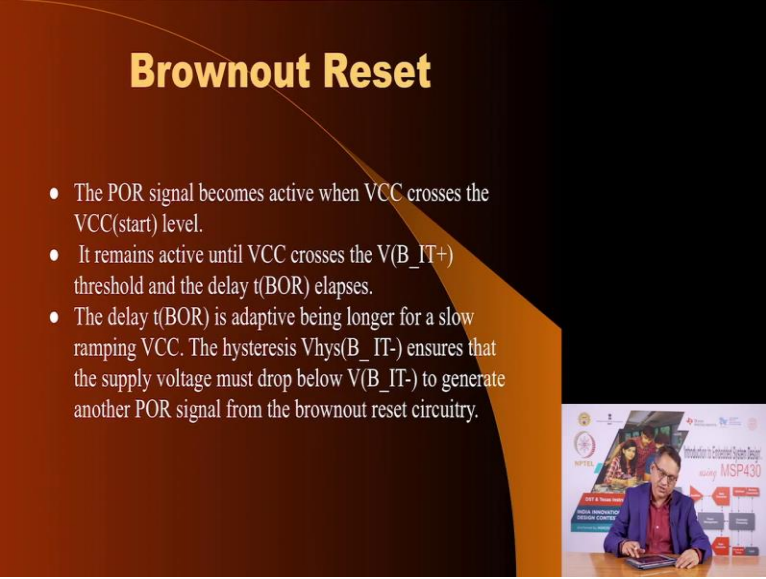
- The brownout reset circuit detects low supply voltages such as when a supply voltage is applied to or removed from the VCC terminal.
- The brownout reset circuit resets the device by triggering a POR signal when power is applied or removed.



This indicates brownout reset if the voltage falls below a certain level here and here, the system is reset. Only when the voltage exceeds this can the system start working.



(Refer Slide Time: 41:36)



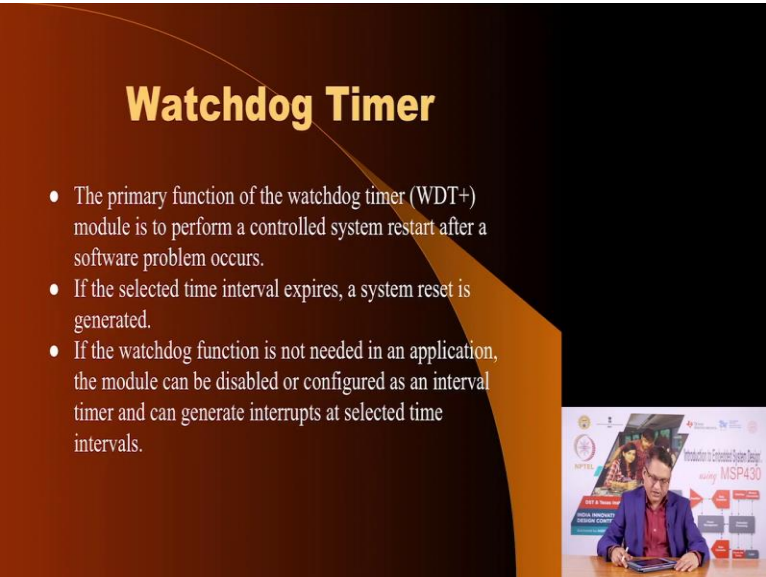
## Brownout Reset

- The POR signal becomes active when VCC crosses the VCC(start) level.
- It remains active until VCC crosses the V(B\_IT+) threshold and the delay t(BOR) elapses.
- The delay t(BOR) is adaptive being longer for a slow ramping VCC. The hysteresis V<sub>hys</sub>(B\_IT-) ensures that the supply voltage must drop below V(B\_IT-) to generate another POR signal from the brownout reset circuitry.

The slide features a dark blue background with a light blue curved graphic on the left. A video inset in the bottom right shows a man in a blue shirt sitting at a desk with a laptop, with a presentation slide titled 'Introduction to Embedded System Design using MSP430' visible behind him.

Please go through this slide to understand how brownout reset works.

(Refer Slide Time: 41:43)



## Watchdog Timer

- The primary function of the watchdog timer (WDT+) module is to perform a controlled system restart after a software problem occurs.
- If the selected time interval expires, a system reset is generated.
- If the watchdog function is not needed in an application, the module can be disabled or configured as an interval timer and can generate interrupts at selected time intervals.

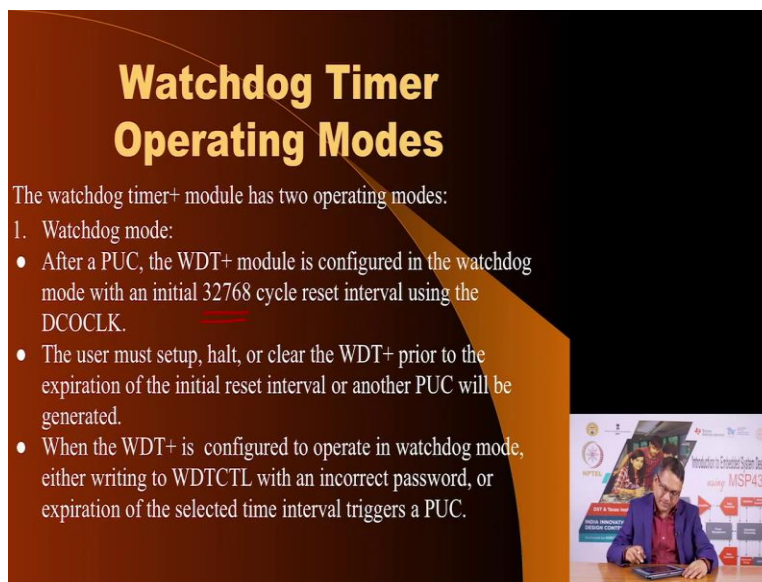
The slide features a dark blue background with a light blue curved graphic on the left. A video inset in the bottom right shows a man in a blue shirt sitting at a desk with a laptop, with a presentation slide titled 'Introduction to Embedded System Design using MSP430' visible behind him.

Watchdog timer is a very important peripheral, it is actually part of a general-purpose timer which can function either as a timer or it can function as a watchdog timer and we can choose which mode of operation do we want that timer to operate at. The primary function would be to as a watchdog timer because there are other timers which are available which you want, which you could use for measuring time. This particular timer is often dedicated for use as a watchdog timer.

Now, what does a watchdog timer do? That if you allow the watchdog timer to function, depending upon the source of clock to the watchdog timer, it will count, it will count up and if you do not reset it, eventually it will overflow and the overflow signal will reset, will generate a power up clear signal, which means you are going to reinitialize the system. You are going to start executing the program, using the vector at the reset vector.

Now, if you enable the watchdog timer, how do you ensure that the watchdog timer does not create a power up condition is to frequently reset the watchdog timer, because when you reset the watchdog timer, it will start from 0 again, it will count up and before it expires, meaning before it overflows it is your responsibility to reset it. In case you forget to do that it will create a power up condition. And we will see later in a code here. We will see how watchdog timer can create a power up clear condition.

(Refer Slide Time: 43:20)



## Watchdog Timer Operating Modes

The watchdog timer+ module has two operating modes:

1. Watchdog mode:
  - After a PUC, the WDT+ module is configured in the watchdog mode with an initial 32768 cycle reset interval using the DCOCLK.
  - The user must setup, halt, or clear the WDT+ prior to the expiration of the initial reset interval or another PUC will be generated.
  - When the WDT+ is configured to operate in watchdog mode, either writing to WDTCTL with an incorrect password, or expiration of the selected time interval triggers a PUC.


The slide also features a small inset image in the bottom right corner showing a person in a blue jacket sitting at a desk with a laptop, with a presentation board in the background that includes the text 'Introduction to Embedded System Design using MSP430'.

The watchdog timer can count up to these many bits that is, it is a 15 bit, although the timer itself is 16 bits, for watchdog purposes it has a 15-bit limit. You can also choose lesser values, which means the watchdog timer can be made to count lesser number of clock signals, and then it can generate a power up clear signal.

(Refer Slide Time: 43:45)

## Watchdog Timer Operating Modes

- Interval mode
  - This mode can be used to provide periodic interrupts.
  - In interval timer mode, the WDTIFG flag is set at the expiration of the selected time interval.
  - A PUC is not generated in interval timer mode at expiration of the selected timer interval and the WDTIFG enable bit WDTIE remains unchanged.
  - When the WDTIE bit and the GIE bit are set, the WDTIFG flag requests an interrupt. The WDTIFG interrupt flag is automatically reset when its interrupt request is serviced, or may be reset by software. The interrupt vector address in interval timer mode is different from that in watchdog mode.




In the interval mode, it can work as a conventional timer and we will see this when we are talking about the timer operations.

(Refer Slide Time: 43:57)

## Watchdog Timer Registers

- WDTCTL: Watchdog Timer+ Control Register:
- IE1: SFR Interrupt Enable Register 1
- IFG1: SFR Interrupt Flag Register 1




Now to control the watchdog timer there are three registers to worry about. One is the watchdog timer control register. The other is an interrupt enable register. This is a special function register and the other is the interrupt flag register.

(Refer Slide Time: 44:16)

## WDTCTL Register


- WDTCTL is a 16-bit, password-protected, read/write register.
- Any read or write access must use word instructions and write accesses must include the write password 05Ah in the upper byte.
- Any write to WDTCTL with any value other than 05Ah in the upper byte is a security key violation and triggers a PUC system reset regardless of timer mode.
- Any read of WDTCTL reads 069h in the upper byte.



In the watchdog timer control register, it is a 16-bit number, a 16-bit register, and it is password protected. It is a read-write register, which means if you want to write, you must also supply the password and we will see what is the password. This is the password. And if you read it, you will in one part of the register, you will get this value.

(Refer Slide Time: 44:46)

Bit	Field	Description
7	WDTCTL	Watchdog timer+ counter clear. Setting WDTCTL = 1 clears the count value to 0000h. WDTCTL is automatically reset.
6	WDTSEL	Watchdog timer+ clock source select.
5	WDTMSEL	Watchdog timer+ mode select.
4	WDTNMI	Watchdog timer+ NMI select. This bit selects the function for the RST/NMI pin.
3	WDTNMIES	Watchdog timer+ NMI edge select. This bit selects the interrupt edge for the NMI interrupt when WDTNMI = 1. Modifying this bit can trigger an NMI. Modify this bit when WDTIE = 0 to avoid triggering an accidental NMI.
2	WDTNMI	Watchdog timer+ NMI select. This bit selects the function for the RST/NMI pin.
1	WDTNMI	Watchdog timer+ NMI select. This bit selects the function for the RST/NMI pin.
0	WDTNMI	Watchdog timer+ NMI select. This bit selects the function for the RST/NMI pin.



This is these are the bits, as you see this is a 16-bit register, this is for password, read or write and these are the effective 8 bits. Let us go through them, the important ones. This is used to hold the timer, meaning with this bit, you can stop the timer. With this bit, you can decide whether

you want the RST/NMI pin to act as RST pin or NMI pin, as you see here if you write a 1, it will function as NMI pin and if you see go back here, you see this indicates that at power up, it will retain the previous value. But on power-on reset, it will be 0. This bit allows you to select the watchdog timer as watchdog or a timer.

And this is, this bit allows you to reset the watchdog timer from so that you can restart from 0. And this bit allows you to select a source of clock for the watchdog timer. You can select either the SM clock or the auxiliary clock and this allows you to decide what is the number of counts that it will count before it overflows and generates the power up clear signal and you can go from 64, a count of 64 to 32768. This will give you the maximum amount of time before the watchdog timer kicks in. So this interrupt enable register, we will see which bits are useful for our operation.

(Refer Slide Time: 46:42)

### IE1: SFR Interrupt Enable Register 1

Address	7	6	5	4	3	2	1	0
00h			ACCIE	NMIE			OFIE	WDTIE
			rw-0	rw-0			rw-0	rw-0


**WDTIE** Watchdog Timer interrupt enable. Inactive if watchdog mode is selected. Active if Watchdog Timer is configured in interval timer mode.

**OFIE** Oscillator fault interrupt enable

**NMIE** (Non-maskable interrupt enable)

**ACCIE** Flash access violation interrupt enable

- **WDTIE** bit enables the WDTIFG interrupt for interval timer mode. It is not necessary to set this bit for watchdog mode.



If you set this to 1 that means you want to use this for as a timer function. If the, this bit is inactive in the watchdog mode, the other is if the oscillator fault, if there is a fault in the oscillator, this will be set to one.

This is when the NMI interrupt happens and the last one is when whenever you try to access regions of flash memory which are not accessible to you, this interrupt will be enabled, meaning if you enable this, an interrupt will be generated when you try to access flash memory, which

you should not, and so on. So as the important point to note here is that this bit is not, is only applicable for the watchdog timer in a timer mode, not in the watchdog mode.

(Refer Slide Time: 47:39)

### IFG1: SFR Interrupt Flag Register 1

Address	7	6	5	4	3	2	1	0
0zh				NMIFG	RSTIFG	PORIFG	OFIFG	WDTIFG
				rw-0	rw-0	rw-1	rw-1	rw-0

**WDTIFG** Set on watchdog timer overflow (in watchdog mode) or security key violation. Reset on V<sub>CC</sub> power-on or a reset condition at the RST/NMI pin in reset mode.

**OFIFG** Flag set on oscillator fault.

**PORIFG** Power-On Reset interrupt flag. Set on V<sub>CC</sub> power-up.

**RSTIFG** External reset interrupt flag. Set on a reset condition at RST/NMI pin in reset mode. Reset on V<sub>CC</sub> power-up.

**NMIFG** Set via RST/NMI pin.

And then we have the SFR interrupt flag register. Now, this is very important because this is telling me lots of flags related to various sources of resets, and we are going to use this in our code. If this bit is 1 that means the source of reset was a watchdog timer. So as we see, set on watchdog timer overflow, meaning if you have enabled the watchdog timer and this the watchdog timer overflows for whatever setting of the clock and whatever bits to count, if this bit becomes 1, that means the source, it would reset the, it will generate the power up there and this bit will be 1.

On the other hand, if the let me look at this reset parts. If the power-on reset was generated, this bit will be 1. If the external reset, meaning the pin was pressed, this bit will be set and if the NMI pin was used, then this will be set to 1. And what is this? This is the oscillator fault, meaning whenever the VLO or the low-frequency crystal oscillator is unable to start, this bit will be set to 1 and the user has to keep writing 0. And if the oscillator is not starting to function, the microcontroller will set it set to 1.

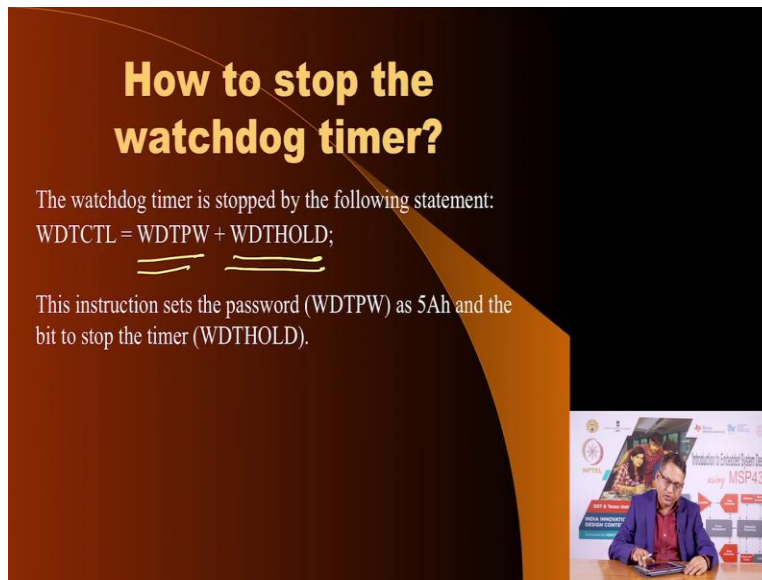
So you write a 0 and wait for some time and then check. Is it still 0? If it is still 0 that means your oscillator is working, and so this is very the part I was referring to earlier. Let us see what we do now.

(Refer Slide Time: 49:33)

## How to stop the watchdog timer?

The watchdog timer is stopped by the following statement:  
`WDCTL = WDTPW + WDTHOLD;`

This instruction sets the password (WDTPW) as 5Ah and the bit to stop the timer (WDTHOLD).

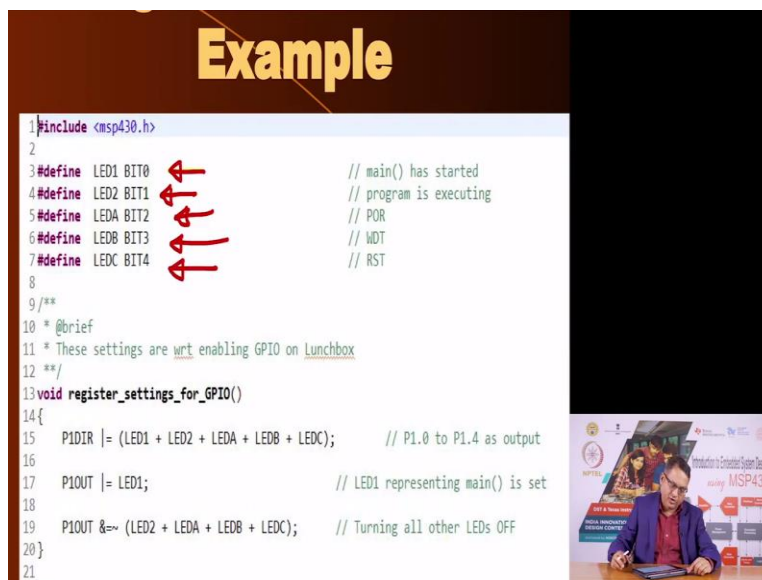


To stop the watchdog timer, you simply write these two values into the watchdog timer control register, these are basically this part of the instruction, this part of the register is supplying the password and this is telling that please reset the, please stop the watchdog timer and we will you will see how it is used in our program example.

(Refer Slide Time: 50:00)

## Example

```
1#include <msp430.h>
2
3#define LED1 BIT0 // main() has started
4#define LED2 BIT1 // program is executing
5#define LEDA BIT2 // POR
6#define LEDB BIT3 // WDT
7#define LEDC BIT4 // RST
8
9/**
10 * @brief
11 * These settings are wrt enabling GPIO on Lunchbox
12 **/
13void register_settings_for_GPIO()
14{
15    P1DIR |= (LED1 + LED2 + LEDA + LEDB + LEDC); // P1.0 to P1.4 as output
16
17    P1OUT |= LED1; // LED1 representing main() is set
18
19    P1OUT &=~ (LED2 + LEDA + LEDB + LEDC); // Turning all other LEDs OFF
20}
21
```



So now what we have done is we have set up a experiment which this is the code for it, and we want you to download this code. Now instead of testing the code on the lunchbox, we want you to just download the code into the lunchbox and then remove the IC and insert it on a breadboard

and connect few switches, well, no switches, a few LEDs to indicate what was the source of reset.

We also want you to connect one switch but that switch will be connected to the RST pin and with this setup, we will be able to identify was the source of reset the watchdog timer overflow or was it power being applied for the first time or was it because the reset pin was pressed. This program allows you to do that.

Now for that, we have used five LEDs, whenever, whatever be the source of reset, the program will start executing. As we have mentioned, the master clock which supplies to the CPU is derived from the DCO at 1.1 megahertz. And so immediately upon whatever be the reason for reset, the microcontroller will start working and this LED will be turned on. This LED is connected to P1.0.

Then when the program actually runs in a loop, this LED which is P1.1 will toggle, will blink. Besides this, the program will try to identify what was the source of reset, if the source of reset was power-on reset that is you had turned the power off and you turned it on again, LED on bit P1.2 will turn on. If on the other hand, the watchdog timer overflow happened, it will turn the LED on P1.3, and the third option is if you press the reset switch, the LED on P1.4 will turn on. And so I recommend that you go through the code. Let us go through this.



(Refer Slide Time: 52:20)

## Determining the Reset Source: Code Example

```
73
74 /*Brief entry point for the code*/
75 void main(void)
76 {
77     WDTCTL = WDTPW | WDTHOLD; // stop watch dog timer ✓
78     volatile unsigned int i;
79
80     BCSCTL1 |= DIVA_3; // Dividing ACLK by 8, 32768Hz/8 = 4096Hz
81
82     register_settings_for_GPIO();
83
84     /* This loop checks for Oscillator fault flag to reset means
85     it delays execution until external crystal is Power On */
86
87     do
88     {
89         IFG1 &= ~OFIFG; // Clear oscillator fault flag
90         for (i = 10000; i ; i--); // Delay, minimum value of i = 5000
91     } while (IFG1 & OFIFG); // Test osc fault flag
92
93     checking_reset_source();
94
95     while(1)
96     {
97         P1OUT ^= LED2; // Toggle LED
98         _delay_cycles(1000000);
99     }
100 }
101
```



## Determining the Reset Source: Code Example

```
1 #include <msp430.h>
2
3 #define LED1 BIT0 // main() has started
4 #define LED2 BIT1 // program is executing
5 #define LED3 BIT2 // POR
6 #define LED4 BIT3 // WDT
7 #define LED5 BIT4 // WDT
8
9
10 #pragma
11 /* These settings are set enabling GPIO on launchbox
12 */
13 void register_settings_for_GPIO()
14 {
15     P1DIR |= (LED1 + LED2 + LED3 + LED4 + LED5); // P1.0 to P1.4 as output
16
17     P1OUT |= LED1; // LED1 representing main() is set
18
19     P1OUT &= ~(LED2 + LED3 + LED4 + LED5); // turning all other LEDs off
20 }
21
22 #pragma
23 /* These settings are w.r.t check source of reset on Launch Box
24 */
25 void checking_reset_source()
26 {
27     if (IFG1 & PORIFG) // Check for Power on Reset Flag (POR)
28     {
29         P1OUT |= LED5; // Turning LED5 On
30         P1OUT &= ~LED3; // Turning LED3 Off
31         P1OUT &= ~LED4; // Turning LED4 Off
32     }
33
34     /* Settings for watchdog timer register
35     Giving Watchdog Timer Password
36     Clearing Watchdog counter to 0x0000
37     Watchdog Source select -> ACLK
38     Watchdog Timer interval set to generate watchdog reset at 2 secs
39     */
40     WDTCTL = (WDTPW + WDTORCTL + WDTSSEL + WDTEN0);
41
42     IFG1 &= ~PORIFG; // Clearing Power on Reset flag
43 }
44
```



## IFG1: SFR Interrupt Flag Register 1

Address	7	6	5	4	3	2	1	0
02h				NMIFG	RSTIFG	PORIFG	OFFG	WDTIFG
				rw-0	rw-0	rw-1	rw-1	rw-0

WDTIFG Set on watchdog timer overflow (in watchdog mode) or security key violation. Reset on V<sub>CC</sub> power-on or a reset condition at the RST/MI pin in reset mode.

OFFG Flag set on oscillator fault.

NMIFG Power-On Reset interrupt flag. Set on V<sub>CC</sub> power-up.

RSTIFG External reset interrupt flag. Set on a reset condition at RST/MI pin in reset mode. Reset on V<sub>CC</sub> power-up.

NMIFG Set via RST/MI pin.

Here we have first stopped the watchdog, this is the main code, we are going to go back to the part of the code which is, which has some functions. This is first we have turned the watchdog timer off, then we have selected the auxiliary clock to operate at this, using the crystal divide by 8, so that we are operating at 4 kilohertz.

What is it going to do? It is going to feed to the watchdog timer. Why do we want such a low-frequency operation for the watchdog timer? So that you can get some time after, let us say, power-on reset or the user reset.

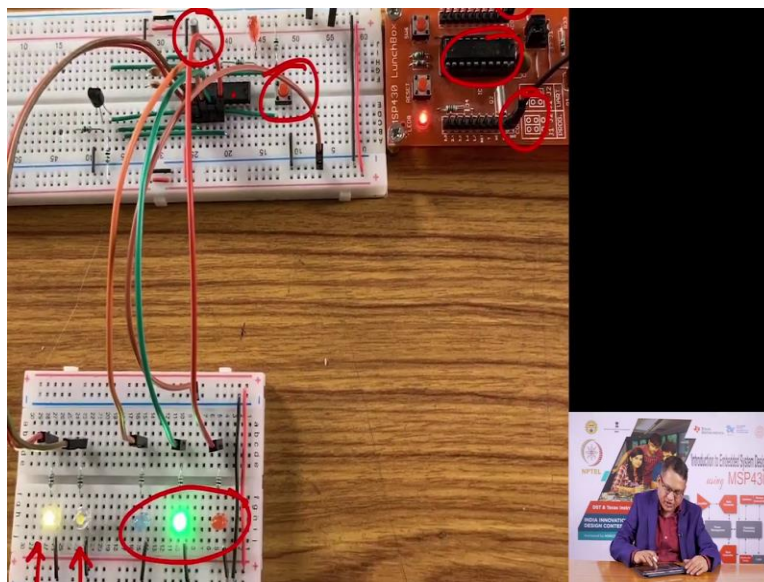
If the watchdog timer is fed with this low-frequency clock, it will give you time before the watchdog overflows and you can see that initially the source of reset was say power-on reset or the user reset, and then when the watchdog timer kicks in, you will see that the system resets again. But now the LED corresponding to the watchdog timer will turn on. And that is why we wanted to give time. And so we have chosen a very low-frequency crystal oscillator.

Here is a code which allows you to set the bits appropriately, we want those port 1 bits to be outputs and now you are waiting for that oscillator to stabilize, and then you check the reset source. Now, also in this we will go and see the code in this, the moment you, the first program that runs is you, of course, turn the watchdog timer off, then you enable the auxiliary clock and then you run off to execute this register settings for GPIO part of the code. Let us see what it does.

Here it simply makes these bits as output, PDIR, and then it simply turn LED1 on and it turns all the other LEDs off. So LED1 is on, the moment you see LED1 on meaning your system is working. Now, what will happen is you go back to the main part of the code. Now you are waiting for the oscillator to stabilize and this may take some time. After that, you go and check the source of reset. Let us see that code here. So in the reset, how do you check the source of reset, you have to look at this register here.

This register, by identifying various bits, you will know whether the source was watchdog reset or whether it was power-on reset or whether it was user reset button. This code basically looks for that turns appropriate LED on and goes back to the main program and in the main program, once you have checked the source of reset, it simply goes in a infinite loop of toggling LED2.

(Refer Slide Time: 55:24)



So here is the how the circuit has been connected. Although I am using a lunchbox and it appears to have a microcontroller here, I am not using this microcontroller. I am using the lunchbox only to drive, derive the power supply from here. This is the MSP430 which you could have taken away out from your MSP430 lunchbox inserted here, as you see here, here is the crystal. So you need to connect a 32 kilowatts crystal for that oscillator that I mentioned and the rest of the connections.

And then here is the, this is LED1, this is LED2 which is going to blink, this will turn on the moment power is applied. And these are the three LEDs which indicate the source of reset

whether it is power-on reset, whether it is the user reset, this is the user reset pin switch and the third source could be the watchdog timer.

So I recommend that once you have programmed your microcontroller in the lunchbox with this code, take it out gently, insert it into the breadboard, using the crystal oscillator and other stuff, connect these LEDs and see how the microcontroller is able to find out the source of reset.

So this is what we have for you in this lecture on sources of clock as well as various ways of various ways in which MSP30 microcontroller can be reset. I will see you very soon with a new lecture. Thank you.