

Introduction to Embedded System Design
Professor. Dhananjay V. Gadre
Netaji Subhas University of Technology, New Delhi
Lecture 22
MSP430 Digital I/O

Hello, and welcome back to a new session. And in this session, we are going to actually connect a piece of hardware, that is our MSP430 lunch box to a laptop, as you can see my setup here has changed apart from the iPad that I used to write and present this lecture. I also have a laptop, on which I have connected the MSP430 lunch box. And I am going to go through the first aspect that you would do when you connect a microcontroller to your system that you would like to run the first program.

And when you learn C programming, the first program that you wrote was 'Hello World' in which you wrote a program, compiled it, and when you executed it on the screen of the laptop or desktop computer it printed 'Hello World'.

Now in our world of embedded systems, our world is dictated by the LED. And so the first program will be 'Hello LED' and I will show you, once I explain how to write a program to manipulate the bits and the pins that we have on our MSP430 microcontroller, how do we write our first program so that we can convey the information on a LED connected to our kit, that we, that is the objective of this lecture. So, let us begin.

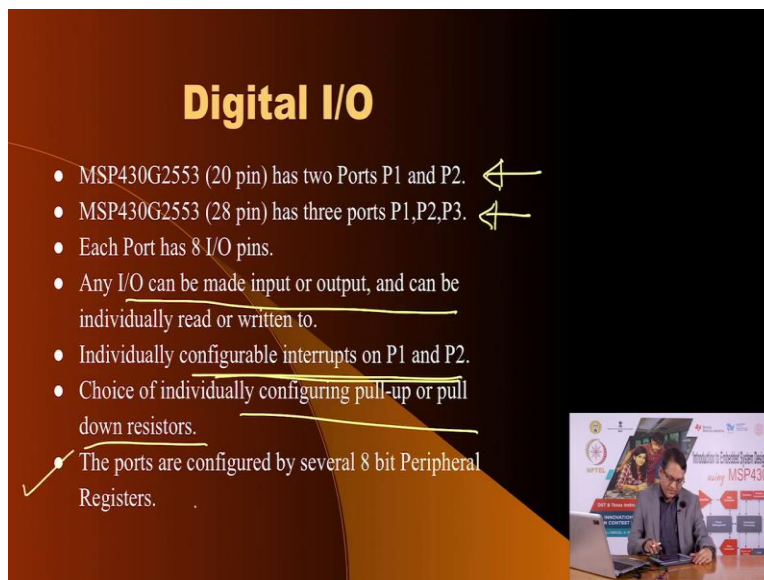
Now, the problem is that in a microcontroller you have so many pins and you have many more functions and the way to address this disparity is that allocate many, many functions on a single pin and leave it to you the programmer, the embedded system designer to write an appropriate program, to select an appropriate function to be routed to that pin. But the basic function of each pin is to serve as either an input pin or an output pin and even that is a big question that, how do you make a pin programmable?

You may recall that in one of the previous lectures I had mentioned, that one of the great features of modern microcontrollers is the ability to decide by writing an appropriate program that of the given number of input output pins how many will be input and how many will be output. And so in this lecture, we are going to first of all solve that conundrum and then we are going to address the issue as to how multiple functions are

routed to these pins, if you choose not to have those pins functions that in, function as input output pins, how do you route additional functions on those pins?

So, we have a slightly complex task at hand and in the beginning it may appear very daunting to you but please you know stay with me and I am sure, I would be able to help you understand this whole complex arrangement of logic which allows a microcontroller to funnel so many functionalities on one side onto a single pin. So since, we are dealing with a MSP430G2553 microcontroller, which itself is available in two footprints, meaning physical devices.

(Refer Slide Time: 03:37)



Digital I/O

- MSP430G2553 (20 pin) has two Ports P1 and P2. ←
- MSP430G2553 (28 pin) has three ports P1,P2,P3. ←
- Each Port has 8 I/O pins.
- Any I/O can be made input or output, and can be individually read or written to.
- Individually configurable interrupts on P1 and P2.
- Choice of individually configuring pull-up or pull down resistors.
- The ports are configured by several 8 bit Peripheral Registers.

The slide features a dark background with a light-colored diagonal shape. A small inset image in the bottom right corner shows a person sitting at a desk with a laptop, with a presentation slide visible on the wall behind them. The presentation slide has the text 'Introduction to Embedded System Design using MSP430'.

The fact is that the first version of the microcontroller, if you choose dip package you will get two ports and those two ports are labeled P1 and P2, and our MSP430 lunch box as a dip IC. In case you choose to implement your system using the SMD version, then you will get 3 ports because that SMD version has 28 pins and those 3 ports are labeled P1, P2 and P3.

Each of these ports has eight input output pins, meaning you can choose whether these eight pins can be input or output or any combination thereof. As I mentioned, each of these input output pins can be input or output and you can read or write to these pins.

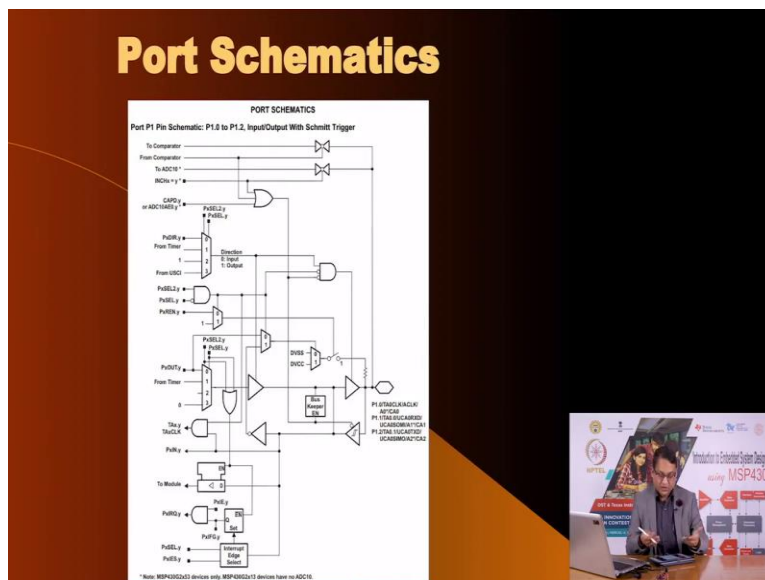
You can also, apart from being input and output or output pins you can also configure these input pins to become a source of interrupts on P1 and P2.

Apart from this functionality if you choose these pins to become input pins, then I had mentioned that when you connect external devices which do not provide any logic voltage then you had to transform the state of those signals as open and closed into 1 and 0, and so you need external resistors, you need pull-up or pull down resistors.

Well, on the microcontroller such as MSP430 you do not have to connect external resistors, you can program the chip so that internal resistors can be pressed into service and you can choose either a pull up resistor or pull-down resistor.

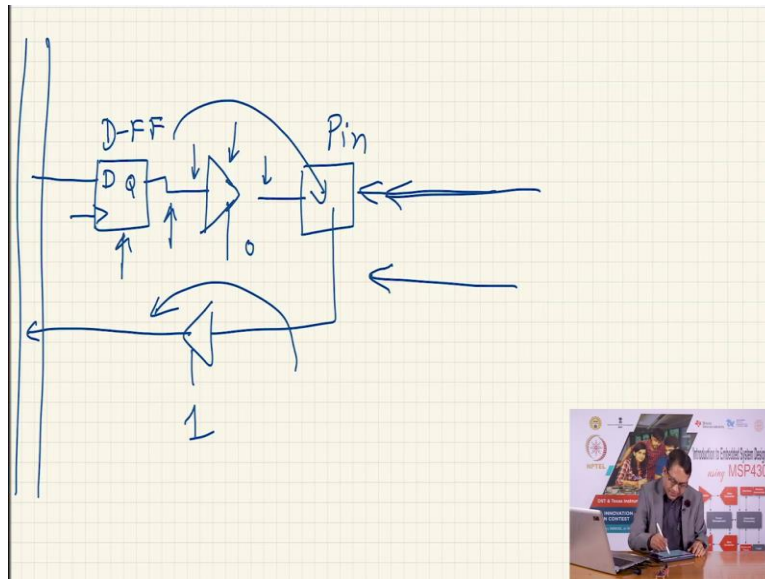
So, you can configure a pull up or pull down resistor and these ports, all these functionalities are programmed through the 8-bit peripheral register interface that I had discussed earlier, in the memory map of the microcontroller there were some registers which were 8-bit access, some which were 16-bit so for the digital input output these are all 8-bit registers. So, this is what we talked about here.

(Refer Slide Time: 05:42)



Now here is a schematic of one of the pin, you see this is such a complex circuit, it is very much possible that you as a beginner will get, you know confused with such a circuit but you do not have to worry, I am here to simplify things for you.

(Refer Slide Time: 06:04)



So, let me draw a simple diagram to illustrate this idea. The idea is that, I have a physical pin, this is my pin, let me label it a pin, this is a pin of the microcontroller and you want a mechanism, you want some circuit which will convert this pin as an output pin, which means here is my microprocessor CPU bus, data bus and I would like to make this pin output when I want to.

But when I reprogram the hardware related to this pin, I should be able to convert this pin into a input pin. Of course, MSP430 offers you even more options but let us first see how very basic digital circuit components can be pressed into service to provide that functionality of being able to change the pin as input pin or as output pin. And one of the biggest beauties of digital electronics is this, tri-state buffer.

So imagine, that I have a tri-state buffer, here is my flip-flop, a D type flip-flop, I call it D-FF, into which I can write any value and here is a clock so that at the clock edge that I decide a particular bit is stored in this flip-flop, the output of this flip-flop feeds the input of this tri-state buffer, now it becomes a tri-state buffer. If the tri-state buffer control signal is 1, then this output gets the value as input.

If on the other hand it is 0, then this tri-state buffer is disconnected from the pin and this is how a typical microcontroller, a modern microcontroller such as MSP430 uses this

kind of hardware to select the functionality and give the control of that functionality to you, the system designer or the programmer so that you can choose.

Do you want this pin to be an output, yes, if you want to this pin to be an output you will write through appropriate registers, a logic one here that will enable this tri-state buffer after that whatever you write into this flip-flop that value will appear here and subsequently it will appear on the pin.

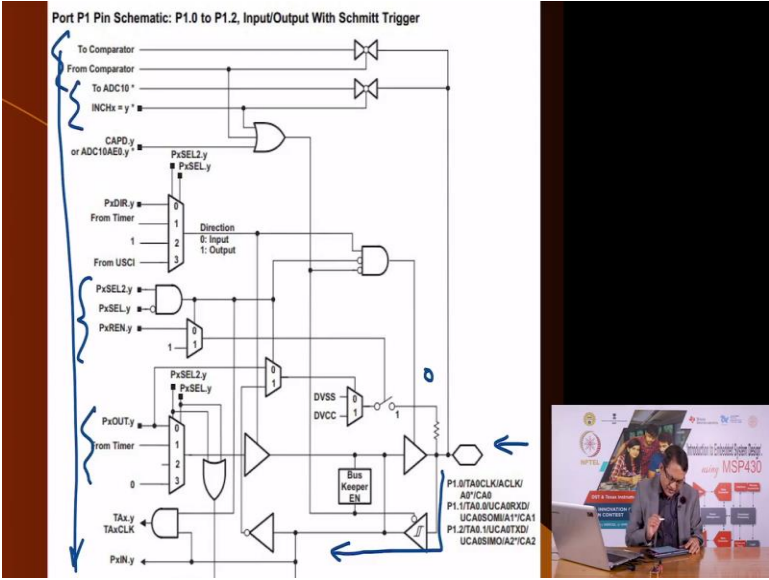
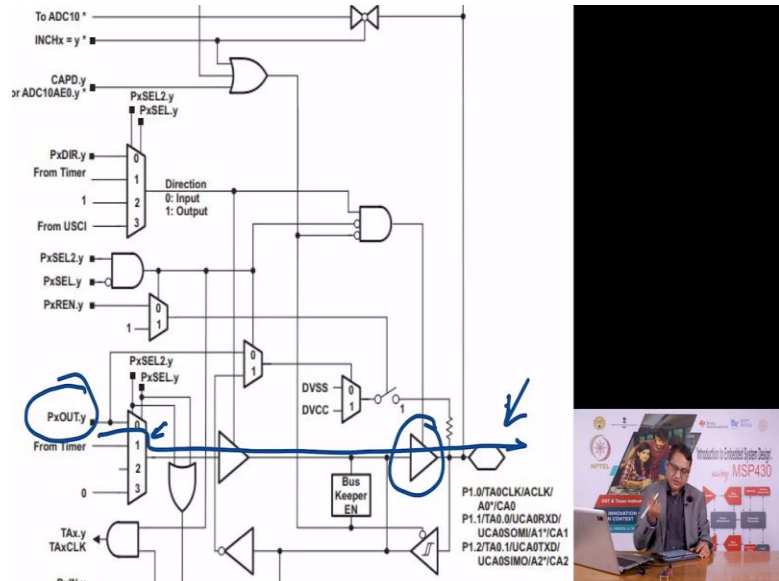
And so when I want to make a pin an output pin, I, my micro controller would have suitable circuitry especially with the tri-state buffer which will be enabled through writing bits into appropriate control registers and this tri-state buffer will be enabled, thereafter I can keep on writing any bit sequence into this flip-flop and that value will appear on this pin.

Now suppose, you want to convert this pin into an input pin. Obviously, you expect an external voltage to be applied to this pin, this voltage could come from a external device or it could be a switch, then you do not want this tri-state buffer to be enabled. Because otherwise, this logic will come from here, and some logic will come from here, and if these two logic values do not agree with each other it will lead to what we call as bus contention.

And the simplest way to avoid it is to write, is to write a logic 0 here, if I write a 0 on this control pin, then it as if disconnects this part. So, I am just left with a hanging pin from the microcontroller side, and now I can use an external device to apply voltage.

And of course, this goes through a buffer and goes into the data bus, so that now, and again this also needs a tri-state buffer so that on demand when the microcontroller program reads, makes this bit a logic 1, here it will read the pin value and put it on the data bus and eventually it will go into your register so, that you can process that information. So, this is the basic idea behind any pin with the functionality that it can be made input pin or output pin. And this is actually what is happening on this pin.

(Refer Slide Time: 10:19)



Here is my pin, if you see here, here is the pin. And you see, this is the tri-state buffer that I was talking about, this tri-state buffer is controlled by this gate which is getting information from some of these registers here and I am going to simplify that information for you.

If this is enabled, this information is coming from this multiplexer and one of the multiplexer inputs is this register called PxAOUT. And you writing information here can be routed into this and it can appear at this pin, if you want to make this pin as output, if

you want to make this pin as input then obviously, you will have to ensure that this tri-state buffer here becomes 0, and then you can apply external voltage it will go through this buffer, tri-state buffer and go into here and then can be read by the microcontroller.

So, as you see here, there are several, you know value is written here, some of them here are related to the port, some of them are talking about timer, there is something here about comparator, and ADC, and so on and so forth. So, this is to just indicate that MSP430 microcontroller pin is multifunctional and it is up to you as the programmer to decide which function this particular pin should behave and so we are going to look at that.

(Refer Slide Time: 11:52)

Table 16. Port P1 (P1.0 to P1.2) Pin Functions

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS ⁽¹⁾				CAPD.y
			P1DIR.x	P1SEL.x	P1SEL2.x	ADC10AE.x INCH.x+1 ⁽²⁾	
P1.0/ TA0CLK/	0	P1.x (I/O)	t: 0; O: 1	0	0	0	0
ACLK/		TA0.TACLK	0	1	0	0	0
A0 ⁽²⁾ /		CA0	X	X	X	1 (y = 0)	0
Pin Osc		Capacitive sensing	X	0	1	0	0
P1.1/ TA0.0/	1	P1.x (I/O)	t: 0; O: 1	0	0	0	0
UCA0RXD/		TA0.0	1	1	0	0	0
UCA0SOMI/		TA0.CCIDA	0	1	0	0	0
A1 ⁽²⁾ /		UCA0RXD	from USCI	1	1	0	0
CA1/		UCA0SOMI	from USCI	1	1	0	0
Pin Osc	Capacitive sensing	X	0	1	0	0	
P1.2/ TA0.1/	2	P1.x (I/O)	t: 0; O: 1	0	0	0	0
UCA0TXD/		TA0.1	1	1	0	0	0
UCA0SIMO/		TA0.CC1A	0	1	0	0	0
A2 ⁽²⁾ /		UCA0TXD	from USCI	1	1	0	0
CA2/		UCA0SIMO	from USCI	1	1	0	0
Pin Osc	Capacitive sensing	X	0	1	0	0	

(1) X = don't care
(2) MSP430G2x53 devices only




Table 16. Port P1 (P1.0 to P1.2) Pin Functions

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS ⁽¹⁾				
			P1DIR.x	P1SEL.x	P1SEL2.x	ADC10AE.x INCH.x=1 ⁽²⁾	CAPD.y
P1.0/	0	P1.x (I/O)	t; 0; 0	0	0	0	0
TA0CLK/		TA0.TACLK	0	1	0	0	0
ACLK/		ACLK	1	1	0	0	0
A0 ⁽²⁾ /		A0	X	X	X	1 (y = 0)	0
CA0/		CA0	X	X	X	0	1 (y = 0)
Pin Osc		Capacitive sensing	X	0	1	0	0
P1.1/	1	P1.x (I/O)	t; 0; 0; 1	0	0	0	0
TA0.0/		TA0.0	1	1	0	0	0
TA0.CCIDA		0	1	0	0	0	0
UCA0RXD/		from USCI	1	1	0	0	0
UCA0SOMI/		from USCI	1	1	0	0	0
A1 ⁽²⁾ /		A1	X	X	X	1 (y = 1)	0
CA1/	CA1	X	X	X	0	1 (y = 1)	
Pin Osc	Capacitive sensing	X	0	1	0	0	
P1.2/	2	P1.x (I/O)	t; 0; 0; 1	0	0	0	0
TA0.1/		TA0.1	1	1	0	0	0
TA0.CC1A		0	1	0	0	0	0
UCA0TXD/		from USCI	1	1	0	0	0
UCA0SIMO/		from USCI	1	1	0	0	0
A2 ⁽²⁾ /		A2	X	X	X	1 (y = 2)	0
CA2/	CA2	X	X	X	0	1 (y = 2)	
Pin Osc	Capacitive sensing	X	0	1	0	0	

(1) X = don't care
(2) MSP430G2x53 devices only

Table 16. Port P1 (P1.0 to P1.2) Pin Functions

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS ⁽¹⁾				
			P1DIR.x	P1SEL.x	P1SEL2.x	ADC10AE.x INCH.x=1 ⁽²⁾	CAPD.y
P1.0/	0	P1.x (I/O)	t; 0; 0; 1	0	0	0	0
TA0CLK/		TA0.TACLK	0	1	0	0	0
ACLK/		ACLK	1	1	0	0	0
A0 ⁽²⁾ /		A0	X	X	X	1 (y = 0)	0
CA0/		CA0	X	X	X	0	1 (y = 0)
Pin Osc		Capacitive sensing	X	0	1	0	0
P1.1/	1	P1.x (I/O)	t; 0; 0; 1	0	0	0	0
TA0.0/		TA0.0	1	1	0	0	0
TA0.CCIDA		0	1	0	0	0	0
UCA0RXD/		from USCI	1	1	0	0	0
UCA0SOMI/		from USCI	1	1	0	0	0
A1 ⁽²⁾ /		A1	X	X	X	1 (y = 1)	0
CA1/	CA1	X	X	X	0	1 (y = 1)	
Pin Osc	Capacitive sensing	X	0	1	0	0	
P1.2/	2	P1.x (I/O)	t; 0; 0; 1	0	0	0	0
TA0.1/		TA0.1	1	1	0	0	0
TA0.CC1A		0	1	0	0	0	0
UCA0TXD/		from USCI	1	1	0	0	0
UCA0SIMO/		from USCI	1	1	0	0	0
A2 ⁽²⁾ /		A2	X	X	X	1 (y = 2)	0
CA2/	CA2	X	X	X	0	1 (y = 2)	
Pin Osc	Capacitive sensing	X	0	1	0	0	

(1) X = don't care
(2) MSP430G2x53 devices only

Now, you see we are going to look at one example, let us see that, let us say that our pin is P1.0. Now P1.0 pin whatever physical pin it maps on to has several functionalities, one of the functionality is that it will act as a port pin 1.0, the other is that it can act as input to one of the timers here TA 0 clock.

The other could be that it can be used to route an external information, in this case the auxiliary clock value can be outputted on this pin. The fourth function that this can serve is it can become the ADC input for with bit 0, the third is that it is a CA 0 input that is the input for a comparator, and the last function that it can serve is that it can be used for capacitive sensing.

So, as you can see each of these port pins has 1, 2, 3, 4, 5, 6 functions and how do you select which function will be used to make this pin work in, these are all the registers we have. So we have one register called P1DIR, that DIR stands for direction, then we have two registers called P1SEL and P1SEL2 so these are select registers.

So let us simplify it, say select register and select two register associated with each port, which means P1 would have a select register, P1 would also have select two register, P1 would have direction register, P1 would also have registers associated with the ADC, and P1 would have register associated with the capacitive sense functionality.

So, let us see, if we want to use the pin as an input output pin then what should be the value of the register called direction register, what should be the value of the two registers called select, and select two registers and the rest of the registers. Now you see here, I can write a 0 or a 1 into the direction register, this will decide whether it becomes input or output. But, to select it as input output, whether it is input or output will be determined by the direction register.

But for it to become an input output pin you have to write a 0 and into the select register and another 0 into the select two registers in the appropriate bit, remember each port has eight pins, each pin is associated with the port bit. So, if I am talking about bit 0, that means I am saying P1DIR 0, I am talking of P1SEL 0, I am talking of P1SEL2.0, and so on.

So, this is very important that you have this functionality associated with each and every pin. Now let us say, that no, you do not want to use this pin as a input output pin, you in fact wanted to become a clock for the timer. That means, the value of P1DIR register for bit 0, let us talk about 0, should be 0 and the select and select two pins, select and select two bits should be 1 and 0 and ADC and the capacitive register can have 0, 0.

If it has to be used to route the internal auxiliary clock signal on to this pin, now you see the direction becomes that it becomes an output. And that is why, you have written the 1 you have to write a 1 into the direction register. And select value is 1 and 0, select two 0 and ADC is 0, CAPD 0. If you want it to become ADC input, then it does not matter what

is the direction value, it does not matter what is the select bit value, select two bit value, X means do not care, you overwrite that function by writing a bit 1 into this register ADC10AE and since you do not want it to be the capacitive sense you write a 0.

On the other hand, if you want it to become a comparator input again this is X , X, X, this is 0 and CAPD becomes 1. And if you want it to become capacitive sense then, the value here is X but SEL1 value has to be 0 and 1 and these two are again 0, 0 this will select this particular pin to become capacitive input.

And this as you see, this kind of functionality is repeated for rest of the pins, the actual functionality will change from pin to pin but the selection process is the same. So, let us see, how many resistors are we going to deal with when we are going to connect, decide to use these pins as input or output.

(Refer Slide Time: 17:15)

Digital I/O Registers

The MSP430 communicates with the Digital I/O Peripherals through a set of 8 bit Peripheral Registers:

- PxDIR ✓
- PxIN
- PxOUT
- PxREN
- PxSEL and PxSEL2

P2DIR

Introduction to Embedded System Design using MSP430

Digital I/O Registers

The MSP430 communicates with the Digital I/O Peripherals through a set of 8 bit Peripheral Registers:

- P_xDIR ✓
- P_xIN ✓
- P_xOUT ← ←
- P_xREN ←
- P_xSEL and P_xSEL2



So, for using the input output pins as I/O registers we have at least 1 the direction register. X here refers to which port are we talking of, for example, if you are talking about port 1 this will become P1DIR, if we are talking of port 2 it will become P2DIR, and so on. So, I will not repeat this. So, we have P_xDIR means the export, the direction register, then we have the information register called P_xIN, this is the register that you will read if you want to make this pin or this port as input, if you want to make this pin of this port as output you will write to the P_xOUT register.

Now, if you choose to make this pin of this register as of this port as input you will of course read this, but you may choose that you want a pull-up resistor or pull down resistor, for that you have to enable the resistors and you do that by writing into this. Whether it will be pull-up or pull down will depend on what you write into the P_xOUT register.

And then, is this pin going to function as input output pin or is it going to take the alternative functions associated with this pin will be determined by writing appropriate bits into SEL and SEL2 bits. So, this is the sequence of operation that we have to perform and let us see how we do that.

(Refer Slide Time: 18:48)

The slide features a dark background with yellow and white text. At the top, 'PxDIR' is written in yellow with two equals signs below it, and 'P1DIR' is written in white with a handwritten arrow pointing to the '1'. Below this is a bulleted list of five points. At the bottom left, a diagram shows an 8-bit register with bits 7 to 0, where bit 0 contains '0' and bit 1 contains '1'. At the bottom right, there is a small inset image of a person presenting.

PxDIR
==
P1DIR

- 'P' stands for port
- 'x' stands for port number (Upto 3 ports available on MSP430G2553, based on package selected)
- It is an eight bit register. It is used to define whether a pin is to be used as an input or as an output.
- If value of a bit in PxDIR register is '0', it is set as input.
- If value of a bit in PxDIR register is '1', it is set as output.
- Default value of PxDIR register for all ports is '0'.

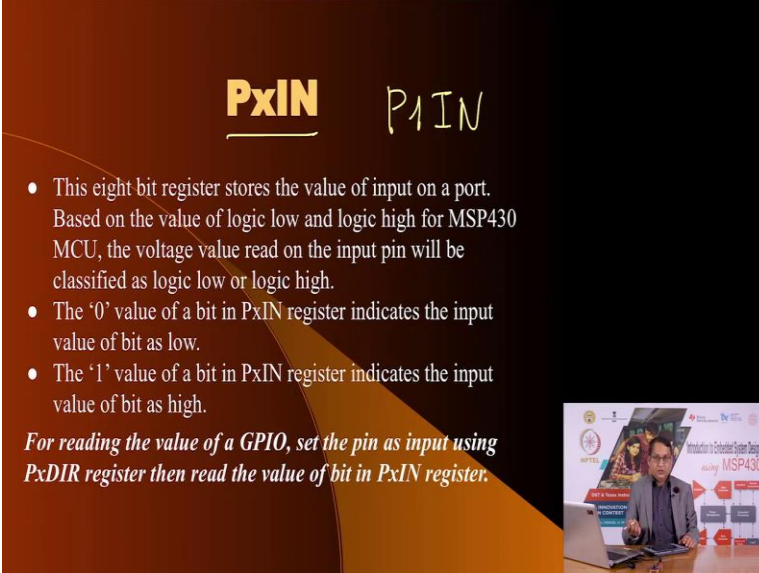
7 ← 0
P1DIR

So, we already gone through that, here X refers to the particular port, P refers to the port. So, if I say, P1DIR that means, I am talking of the direction register associated with port 1. And if we choose the 28 pin package you will have port 1, port 2, port 3 it is a 8 bit register. A value if you write in a particular bit, so you have 8 bits in a port, so let me show that 8 bits to you, this is how I make 8 bits, this is bit 0 and going up to here bit 7.

Now, in this if I write a 0, so this is my P1DIR register. If in this I write 0 at bit 0 that means the pin connected to bit 0 of port 1 will be configured as an input, that is the meaning. If next pin if I write 1, that means bit 1 of port 1 whichever pin it is connected to will become an output pin and I will write appropriate values in this register to decide the basic functionality of the port, whether I want them as input or output. I hope it is clear.

Now, if I choose by writing in the P1 direction register to be a 0 to make it as input, where do I read the input from? This is just going to determine the direction input or output. Now, you have to read different registers if you have made this pin as input or you have made this pin as output.

(Refer Slide Time: 20:33)



PxIN P1IN

- This eight bit register stores the value of input on a port. Based on the value of logic low and logic high for MSP430 MCU, the voltage value read on the input pin will be classified as logic low or logic high.
- The '0' value of a bit in PxIN register indicates the input value of bit as low.
- The '1' value of a bit in PxIN register indicates the input value of bit as high.

For reading the value of a GPIO, set the pin as input using PxDIR register then read the value of bit in PxIN register.

The slide also features a small inset image of a presenter at a desk with a laptop and a banner in the background that reads 'Introduction to Embedded System Design using MSP430'.

If you want to make this pin as input, you have to read a register called PxIN, so here for P1 it will become P1IN. Now of course, you may choose that for the 8 bits some of them are inputs some of them are output, how do you read the value, you will read the entire register, you will read P1IN and then isolate those bits which you have configured as input and identify their values, a 1 means the pin is 1 is held at logic 1, if it this register reads a 0 in a particular bit location, that means that pin was externally set to 0 that is the meaning of this entire slide here and you can go through it to verify what I mentioned.

On the other hand, if you have decided that for a particular bit you want it to be output you are going to write a 1 in that bit location.

(Refer Slide Time: 21:37)

PxOUT

P1OUT

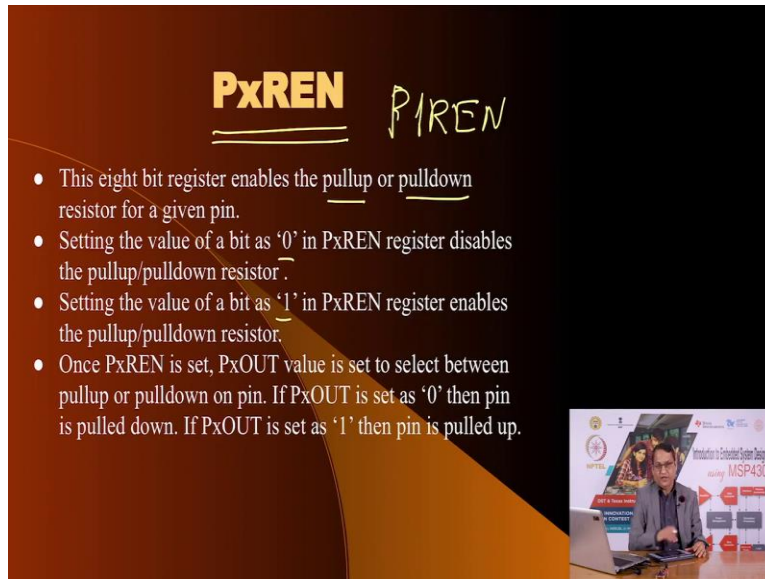
- This eight bit register sets the digital value on a port.
- Setting the value of a bit as '0' in PxOUT register sets the output value of bit as low. _____
- Setting the value of a bit as '1' in PxOUT register sets the output value of bit as high. _____

For setting the value of a GPIO, set the pin as output using PxDIR register then set the value of bit in PxOUT register.

How do you actually write the value, how do you actually send the value to the port pin, you are going to do that by writing into the register called PxOUT, in this case that value becomes P1OUT. So you will write values in this register, those bits of port 1 which are configured as output they will take value that you write into P1OUT and will decide whether that pin becomes 1 or 0, you will of course write it into the entire register but only those ones will be affected which you have made as output, please understand this.

You, when you write to a register you are writing all the 8 bits but maybe you have only one bit we have configured as output so the other seven bits will not affect the functionality on those pins because they are not output, only the one which was configured as output will get the value. And if you want to output a 0, you write a 0 into the PxOUT register, if you want to output a 1 you are going to write a bit 1 into that bit.

(Refer Slide Time: 22:37)



The slide features a dark background with a large orange 'L' shape on the left. At the top, the text 'PxREN' is written in a bold, yellow, sans-serif font and underlined. To its right, 'PIREN' is written in a white, handwritten-style font. Below the titles, there is a bulleted list of four points. In the bottom right corner, there is a small inset image of a man in a grey suit sitting at a desk with a laptop, with a presentation slide visible behind him.

- This eight bit register enables the pullup or pulldown resistor for a given pin.
- Setting the value of a bit as '0' in PxREN register disables the pullup/pulldown resistor.
- Setting the value of a bit as '1' in PxREN register enables the pullup/pulldown resistor.
- Once PxREN is set, PxOUT value is set to select between pullup or pulldown on pin. If PxOUT is set as '0' then pin is pulled down. If PxOUT is set as '1' then pin is pulled up.

And then as I mentioned, if you want to configure a port as input, you had to decide do you want internal pull-up resistors, if you do, then you control and program this register called PxREN, which means for port 1 this will become P1REN, port 1 resistor enable and this enables either the pull-up or pull down.

Now, if you write a 0 in the PxREN register at an appropriate bit location it will disable the function of pull-up pull down, if you write a 1 it will enable it, if you write a 1, question is will it become pull-up or pull down? The answer is provided by writing an appropriate value into the corresponding bit of the P1OUT register, by writing 1 into P1OUT at that bit location it will make a pull-up function, if you write a 0 it will become pull down function that is the meaning of this.

(Refer Slide Time: 23:48)

P1SEL P1SEL2
PxSEL and PxSEL2

- Multiple functions are available on a single pin, PxSEL and PxSEL2 register are used to select function of a given pin.

PxSEL2	PxSEL	Pin Function
0	0	I/O function is selected.
0	1	Primary peripheral module function is selected.
1	0	Reserved. See device-specific data sheet.
1	1	Secondary peripheral module function is selected.

Now, we also had to deal with the select bits, there are two registers; one is called P1SEL and P1SEL2, so this will become P1SEL for port 1, and this will become P1SEL2. And what are the options we have? If the value is 0, 0 written into these two registers at any bit location that enables that pin to function as input output, if it is 0, 1 it is the primary peripheral module if it is 1, 0 it is reserved you are not supposed to use this combination and if you write a 1, 1 into corresponding bit locations of these two registers a secondary peripheral function is selected.

And we have already gone through the various combinations of PSEL and PSEL2, how the other functions are selected. So now, we know that we have so many registers to deal, with we have a direction register, we have an input register, we have an output register, we have a pull-up enable register, and we have select and select two registers these many registers you have to write.

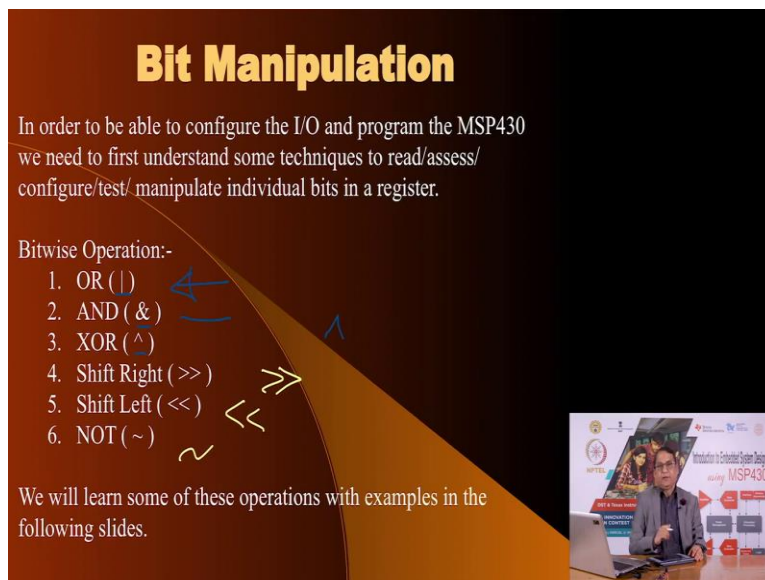
Now thankfully, many of the registers have friendly default values, that is values which are at reset are helping you that these are configured as port and they are usually configured as input, so you do not have to worry.

So, the only thing you have to do is, you have to decide whether you want the port to become input or output by writing into the direction register and if you have chosen them

to become output to write values into the output register and that is it. Only when you want to invoke the additional functionality, the secondary functionality you have to worry about writing into the PSEL and PSEL2 registers.

Now since, in this first exercise we are going to deal with writing values, converting these pins as output and writing values of ones and zeros we are, our program is going to be very simple. But we still need to go through how we manipulate bits, so that I can make a 1 or 0.

(Refer Slide Time: 26:17)



Bit Manipulation

In order to be able to configure the I/O and program the MSP430 we need to first understand some techniques to read/assess/ configure/test/ manipulate individual bits in a register.

Bitwise Operation:-

1. OR (|)
2. AND (&)
3. XOR (^)
4. Shift Right (>>)
5. Shift Left (<<)
6. NOT (~)

We will learn some of these operations with examples in the following slides.

The slide features a dark brown background with a light brown curved shape on the right side. Blue arrows point from the operation symbols to their corresponding list items. A small inset video in the bottom right corner shows a man in a suit presenting at a desk with a laptop and a screen displaying 'Introduction to Embedded System Design using MSP430'.

So as we know, we involve bitwise operations and we can invoke or function, that is or two bits in corresponding locations and two bits for this we use a symbol this, for or we use, this XOR we use this here, for shift right we use this, for left we use this, and for inverting we use this tilde. Let us see some examples to deal with this.

(Refer Slide Time: 26:40)

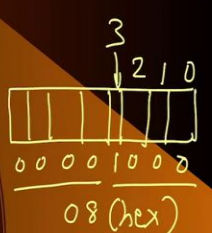
Setting outputs

The I/Os are independently configurable!

Example: For Port P1, suppose there are 8 LEDs connected on each pin. Suppose we have to turn on the LED on P1.3, without changing any configuration on the rest of the port. How will we do it?

Based on the logic that $x \mid 0 = x$ and $x \mid 1 = 1$, there are a number of ways to set a bit:

1. $P1OUT = P1OUT \mid BIT3;$ ←
2. $P1OUT = BIT3;$
3. $P1OUT = P1OUT \mid 0x08;$



The diagram shows an 8-bit register with bits 7 down to 0. Bit 3 is highlighted with a vertical line and the number 3 above it. Below the register, the binary value 00001000 is written, which is equivalent to 08 in hexadecimal.

Now, since the input output pins are independently configurable, it is very important that we understand what information are we writing into any direction register, then outputting into the output register, reading from the input register, selecting appropriate pull-up values by enabling or disabling the REN register, and so on. So here let us say, that we execute an instruction $P1OUT$ is equal to $P1OUT$ or $BIT3$. Now what is this $BIT3$, $BIT3$ are value stored in the header file associated with this, we will show that with an example.

And this allows us, these are mask bits, now if I say $BIT3$, now let me show where is $BIT3$. In 8-bit register this is bit 0, 1, 2 and this is bit 3. Now we are saying, $P1OUT$ is equal to $P1OUT$ or with bit 3, so bit 3 is 0 0 0 0 1 0 0 0. Essentially, this is nothing but if you see this this becomes these are four bits become 0, these 4 becomes 8, so this is 0 8 in hexadecimal.

So, these 3 are doing the same thing, here we are explicitly saying that $P1OUT$ is equal to $P1OUT$ original value ored with $BIT3$ which is 8. In the second statement we are saying it in a different, way we are saying $P1OUT$ is equal to, repeat this operation $P1OUT$ with odd with $BIT3$. And in this one we are explicitly saying, that or $P1OUT$ with this information which is $0x08$. So, this is the way you can manipulate the bits using the or

operator. Now we have seen, how to make a bit 1 or 0 on any of the port pins. Let us look at how do we turn that LED on.

(Refer Slide Time: 28:53)

Setting outputs

Now we have to turn off the LED on P1.3, without changing the output on the rest of the port. How will we do it?

Based on the logic that $x \& 0 = 0$ and $x \& 1 = x$, there are a number of ways to clear a bit:

1. $P1OUT = P1OUT \& (\sim BIT3);$
2. $P1OUT \& = \sim BIT3;$
3. $P1OUT = P1OUT \& 0xF7;$

Similarly for toggling the output on any pin:
 $P1OUT = P1OUT \wedge BIT3;$

Handwritten notes: 08 hex for BIT3, 00001000 for its invert, and $11110111 = F7$.

Now you see here, the basic assumption is that on our port in this case P1.3 our LED is connected like this, which means if the bit is 1 the LED will glow, if the P1.3 bit is 0 it will become, it will turn off. Now we want to turn off the LED, we do not know what is the previous value, we do not know whether the LED was on or off, how do you ensure that we turn off that LED? Now there is no way to directly address this LED, why, because this pin is part of eight pins and so if you write to this register it is likely to change the information on the other bits.

So what we must do is, we must isolate this bit and only make it 0 and so this is what it does, that P1OUT is equal to the original value of P1OUT which means the value that exists right now on this port, I want to and it with the invert of BIT3, I already mentioned what is BIT3, BIT3 is 08hex, which means 00001000.

Now what is the invert of that, it becomes 11110111 which is equal to F7. So in this, without explicitly mentioning F7x, I have inverted that bit and involved in that operation it is going to take the original value of port 1 out register and it with this F7 which will make the BIT3 equal to 0 and I am going to send that information to that port, what will it

do, it will retain the information on the rest of the seven pins and only make port 1 BIT3 equal to 0.

Another way of doing that is this. And the third way is to explicitly mention, that you want to and P1OUT with F7 which of course, you got using inversion of BIT3 also. So, this is the way to turn a particular pin of the previous one showed how to turn it on.

(Refer Slide Time: 31:06)

Reading inputs

Suppose there is a switch connected to P1.5 and we need to read the value of the pin.

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

```
if ((P1IN & BIT5) == 0)
    // to be done when P1.5 = 0
else
    // to be done when P1.5 = 1
```

The diagram shows a 8-bit register with bits 7, 6, 5, 4, 3, 2, 1, 0. Bit 5 is highlighted with a bracket and a '5' below it.

Production & Controller System Design using MSP430

Now in case you want to make a particular port pin as input, then you must read the value on the register called P1IN. And here we have taken an example, that suppose we have a switch connected to P1.5, port1 bit 5 then you must and the BIT5, now what is BIT5? Here, this is 0 1 2 3 4 and this is 5 6 and 7, so you want to know what is this value so you will read it, isolate this bit and if, if the anding operation is equal to 0 that means this bit is 0 you can choose to do something, if else you can do something else, that is the meaning of this structure of the program.

So, you can read entire register P1IN, isolate a particular bit that you are interested in using bit operations and the result will be 1 or 0, which will allow you to do one or the other thing that you want to do.

So we have now seen how to manipulate the registers to program the MSP430 appropriately, so that we can control the pins whether they are input or output, right now

we are going to deal with the port pins as output and so we are ready to write our first program and to write it, compile it, and then download it into our MSP430 to control the LED which is connected on the appropriate pin of the MSP430 lunch box, which I will mention in the next part of this where I am going to switch over to the laptop.

So, what I have here is my laptop and I have opened the 'Hello LED' program you would have downloaded it from the git repository and you would see a program called 'Hello LED' I will go through that.

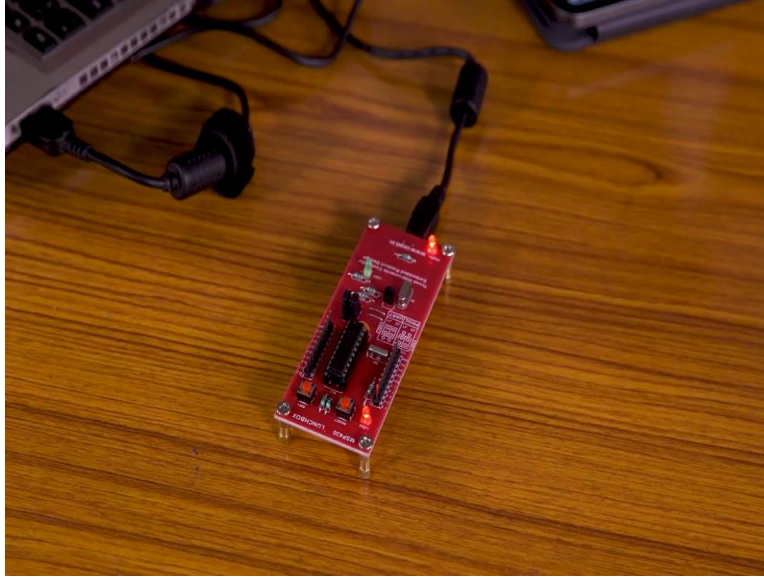
(Refer Slide Time: 33:09)

The image displays two screenshots of an IDE (likely Keil uVision) showing the compilation and execution of a program for an MSP430 microcontroller. The code in the main() function is as follows:

```
int main(void) {
    MCTL = 0x7F; // Stop watchdog (Not recommended for code in production and devices working in field)
    // P1DIR |= 0x01; // Set P1.7 to output direction
    P1DIR |= 0x01; // Set P1.7 to output direction
    P1OUT |= 0x01; // Set P1.7 to HIGH voltage
    // P1DIR |= 0x01; // Set P1.7 to output direction
    P1OUT |= 0x01; // Set P1.7 to HIGH voltage
    // P1DIR |= 0x01; // Set P1.7 to output direction
    P1OUT |= 0x01; // Set P1.7 to HIGH voltage
    return 0;
}
```

The top screenshot shows the compilation process. The console output indicates that the program was successfully compiled and downloaded to the device. The bottom screenshot shows the program running on the device, with the console output indicating that the program is running successfully.

Description	Resource	Path	Location
Optimization Advice (3 items)			
Power (ADP) Advice (3 items)			



```

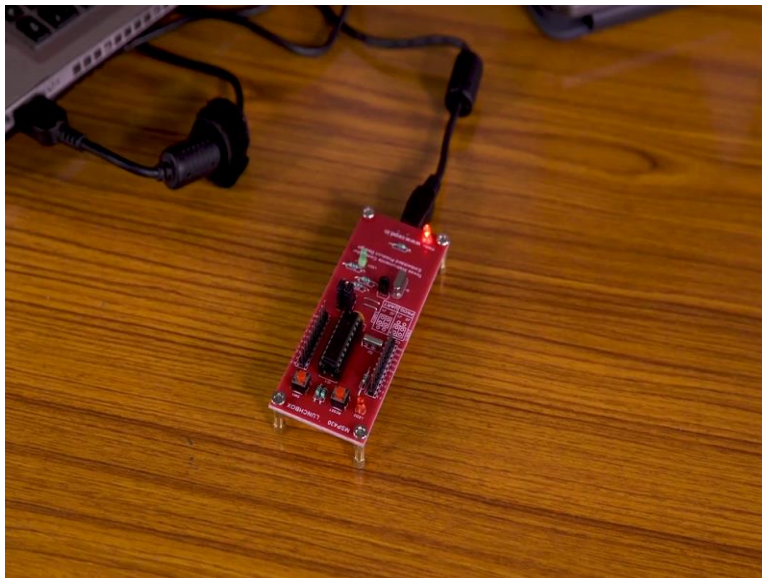
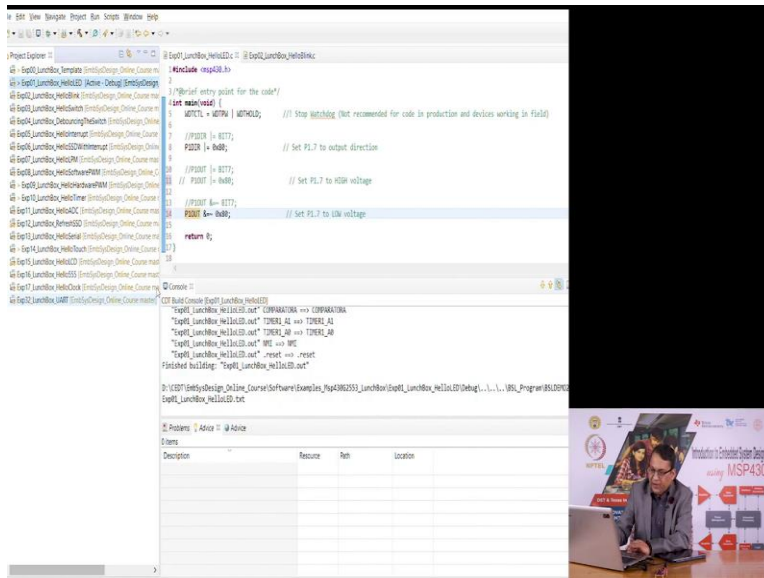
1 #include <msp430.h>
2
3 //Brief entry point for the code!
4 int main(void) {
5     WDTCTL = WDTPW | WDTHOLD; // Stop watchdog (Not recommended for code in production and devices working in Field)
6
7     //P1DIR |= BIT7;
8     P1DIR |= BIT0; // Set P1.7 to output direction
9
10    //P1OUT |= BIT7;
11    P1OUT |= BIT0; // Set P1.7 to 0V and voltage
12
13    //P1DIR |= BIT7;
14    P1OUT |= BIT0; // Set P1.7 to 0V voltage
15
16    return 0;
17 }

```

```

*** Clean only build of configuration Debug for project Debug_LaunchBox_WallID ***
"C:\I:\code\utils\bin\gmake" -j 4 clean -o

```

So, the first part of the program is talking about including MSP430H, this is a header file which has useful information, such as the bit mask and so on. Then you have the main code, this main code is like an infinite loop where you are doing something. And the reset, when you reset a MSP430 micro controller you remember when we were talking about reset it turns on the watchdog timer, and if the watchdog timer is turned on it is going to reset you very frequently.

So, we do not want the watchdog timer to interrupt what we are doing or to stop what we are doing and therefore, the first instruction is to stop the watchdog timer from generating reset information. Then we come to dealing with the deciding the direction of our port,

here we have LED connected to bit seven. And so what we do is, we make P1DIR which is the direction register for port 1 we make it 1 and we do it without affecting the functionality on the other pins.

So we have an instruction called P1DIR is equal to P1DIR ored with 0x80, so that means the most significant bit being equal to 1 the rest being 0 is added, is ored with the rest of the, with the original value of direction register and that is sent again to the direction register, this configures P1.7 pin to become an output pin. Now, in the current form we have, we are writing a 1 into that by saying P1OUT is equal to the original value of P1OUT ored with 0x80, so we are turning it on and I am going to do this.

So, I select the 'Hello LED' program and on that I press the rebuild project and you see what it does, it is going to compile the program and eventually it will download the code into the microcontroller bit and lo and behold the user LED has turned on. Now let me, modify this program, so as to turn it off. I will comment this part of the code here, I am going to comment this and uncomment this part where I want to make it 0 and so I remove it.

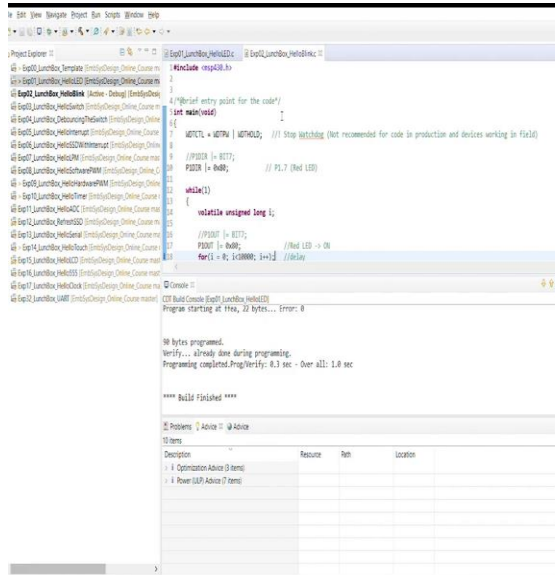
And now I will recompile this, rebuild this, again it is going to, you know compile it and then download it and you see the LED has been turned off. So, this is the first part of the exercise of writing a program and downloading it into the microcontroller kit.

What have we seen in this, that our first part of the instruction is to disable the watchdog timer, then select, write into the direction register so as to select the direction of the bit that we are dealing with, in this case it is P1.7, and then since we want it as an output we write appropriate values on the output register that is P1OUT on bit seven because we do not want to affect the output pins of other bits, so we only isolate P1.7 and write whether we want to write a 0 will turn the LED off, if we write a 1 will turn the LED on, and that is the first part of this exercise.

Now this may look very trivial, but to be able to come to this point gives you great confidence for many things, one that your compile process is working, that you have the ability to download code into your working board and that the board is also working.

Now let us go to the second part of this program, which is to basically blink the same LED so instead of just turning it on or off we are going to add a certain more code into that initial this program and make it blink.

(Refer Slide Time: 37:52)

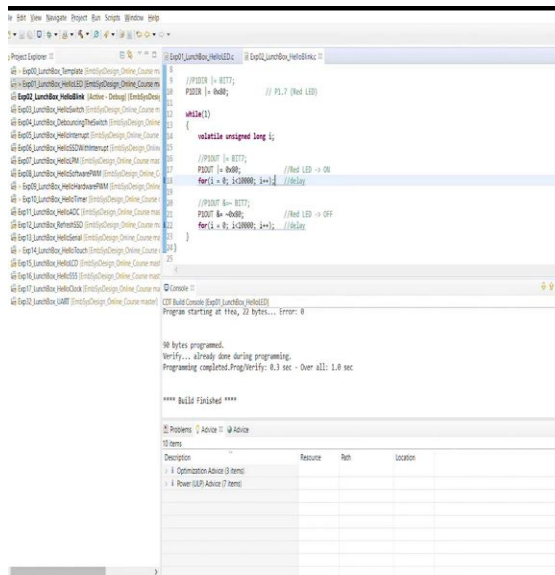


```
1 #include <msp430.h>
2
3
4 /*Brief entry point for the code*/
5 int main(void)
6 {
7     WDTCTL = WDTPW | WDTHOLD; // Stop Watchdog (Not recommended for code in production and devices working in field)
8
9     //P1DIR |= BIT7;
10    P1DIR |= BIT7; // P1.7 (Red LED)
11
12    while(1)
13    {
14        volatile unsigned long i;
15
16        //P1OUT |= BIT7;
17        P1OUT |= BIT7; //Red LED -> ON
18        for(i = 0; i<50000; i++) //50ms
19        {
20
21        }
22
23        //P1OUT ^= BIT7;
24        P1OUT ^= BIT7; //Red LED -> OFF
25
26        for(i = 0; i<50000; i++) //50ms
27        {
28
29        }
30
31    }
32 }
```

38 bytes programmed.
Verify... already done during programming.
Programming completed.Prog/Verify: 8.3 sec - Over all: 1.8 sec

**** Build Finished ****

Description	Resource	Run	Location
Optimization Advice (3 items)			
Power LSP Advice (7 items)			



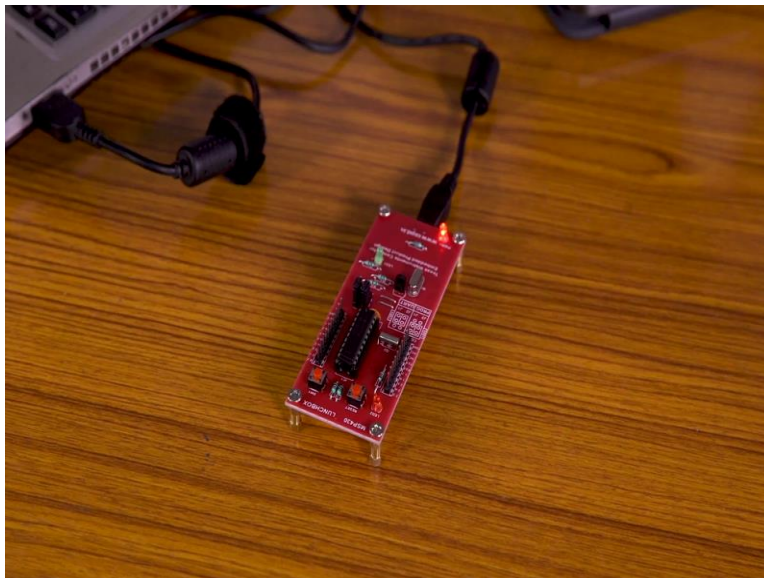
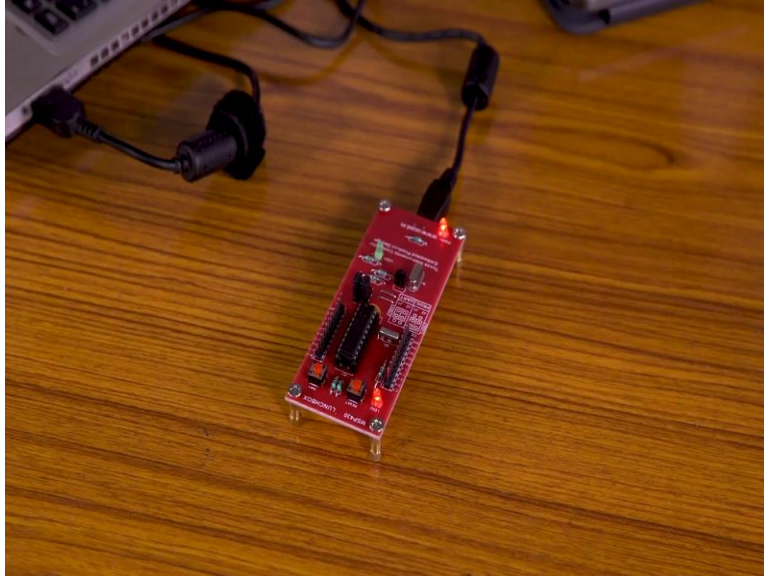
```
1 #include <msp430.h>
2
3
4 /*Brief entry point for the code*/
5 int main(void)
6 {
7     WDTCTL = WDTPW | WDTHOLD; // Stop Watchdog (Not recommended for code in production and devices working in field)
8
9     //P1DIR |= BIT7;
10    P1DIR |= BIT7; // P1.7 (Red LED)
11
12    while(1)
13    {
14        volatile unsigned long i;
15
16        //P1OUT |= BIT7;
17        P1OUT |= BIT7; //Red LED -> ON
18        for(i = 0; i<50000; i++) //50ms
19        {
20
21        }
22
23        //P1OUT ^= BIT7;
24        P1OUT ^= BIT7; //Red LED -> OFF
25
26        for(i = 0; i<50000; i++) //50ms
27        {
28
29        }
30
31    }
32 }
```

38 bytes programmed.
Verify... already done during programming.
Programming completed.Prog/Verify: 8.3 sec - Over all: 1.8 sec

**** Build Finished ****

Description	Resource	Run	Location
Optimization Advice (3 items)			
Power LSP Advice (7 items)			





So, this we call as the ‘Hello Blink’ you would again, by downloading it from the git you would have access to this. Here again we have the include instruction and then we have the main code loop which is an infinite loop in that we, on line number seven or whatever line you have on your version you would find that we have disabled the watchdog timer. And then we write the direction, we make the direction of pin 1.7 port 1 bit seven, we make it output by writing a 1 into it and then we go into an infinite loop.

See in the earlier case we just wanted to write 1 or 0, now we want to continuously write 1 with some delay we write a 0, then with some delay we write 1 again and so on so forth, so that the LED can blink. Now it will appear to be blinking only when the delay

between, delay between the turn on and turn off is sufficient to be observed by human eye and so we have written a delay, small delay function which delays, it ensures that the state of the LED is held to a value of 0 or 1 long enough for it to be observable by human eye.

And so, what we have here is mechanism to do that, in the first part we are turning the LED on by writing a 1 into that register, then we have a line where we say for i equal to 0, i less than ten-thousand essentially this is a delay. We can also replace with some built-in delay functions which we will look at later on. But right now, this is like a delay created using a loop which runs for certain amount of time, once you get out of that loop you again you toggle the value.

Now here we have done and the value of the original value with the inverted value of 80, that is what we have done here. And this toggles, this makes the LED 0 and then again we delay, so we have another same repeat of delay function and then this is the complete while loops, so it is going to go through that writing 1, delay, writing 0, delay, and it is going to do this infinitely.

Now when you do compile this, when you build this code, so I have choose rebuild project, now you see it is going to compile, and download and lo and behold that LED is now blinking at a comfortable rate which is observable by the human eye. I hope you are able to get a similar experience with your kit that you are able to turn the LED on and off and then you are also able to blink it, if you have reached this stage I must extend my congratulations to you because that means that the entire ecosystem is working as good to you with you as it is to us.

And we are ready to jump into the next part of our programming exercises, where we will show how to read inputs from the pins, how to output values based on the inputs that we read and many other things including interrupts, and so on and so forth. So, I hope you are with us in this activity, I thank you and I will see you very soon. Bye.