Embedded Systems

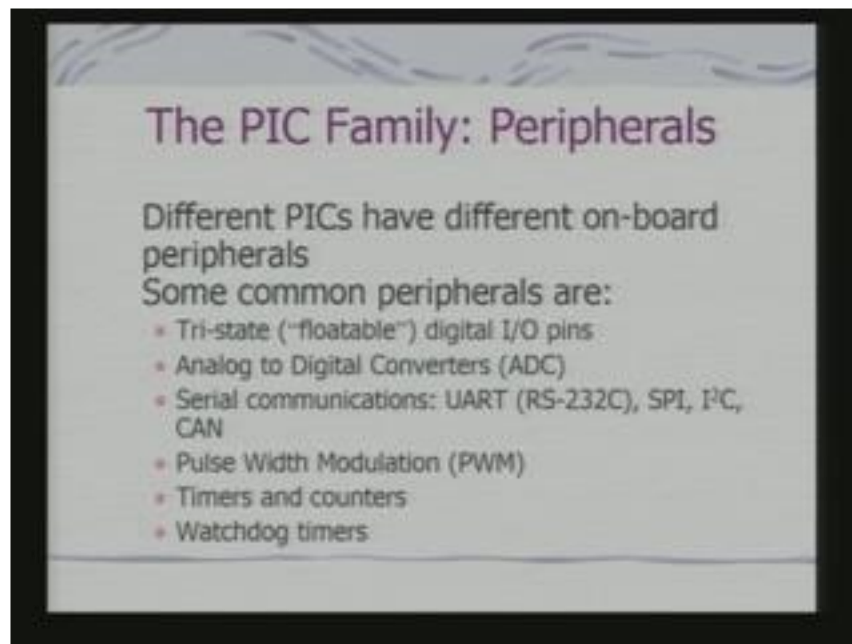
Dr Santanu Chaudhury
Department of Electrical Engineering
IIT Delhi

Lecture 4

PIC Peripherals on chip

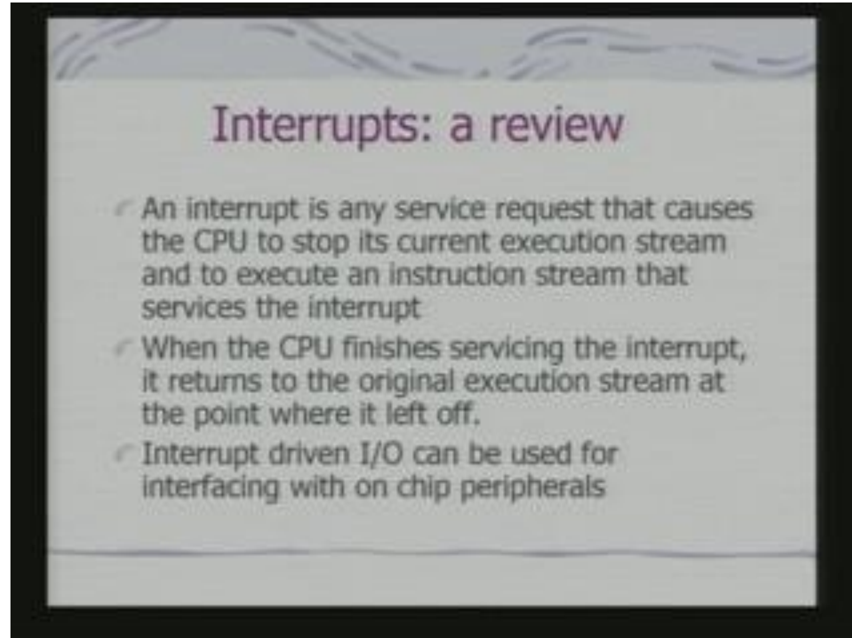
We have already studied the PIC processor, its architecture as well as its instruction set. But PIC micro-controller other than processor has also got on chip peripherals. Today we shall look at these peripherals.

(Refer Slide Time 01:44 min)



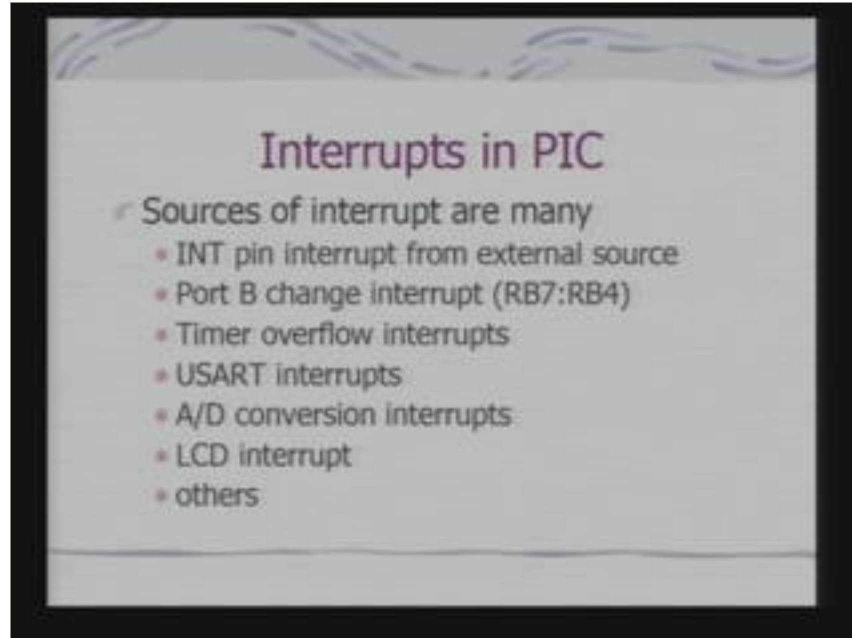
The more common peripherals on this PIC micro-controller family are digital I/O ports, analogue to digital converters, serial communication interfaces like UART SPI, I2C as well as CAN interface, pulse width modulation module, timers and counters and definitely watchdog timer. It is not that all these peripherals present in all PIC micro-controller variants. In fact, there are more peripherals than what are listed here present in some of these micro-controllers. But before going into these peripherals in detail, we shall like to look at interrupt mechanism of PIC processors because these peripherals can be interfaced using this interrupt driven I/O with the PIC processor.

(Refer Slide Time 02:29 min)



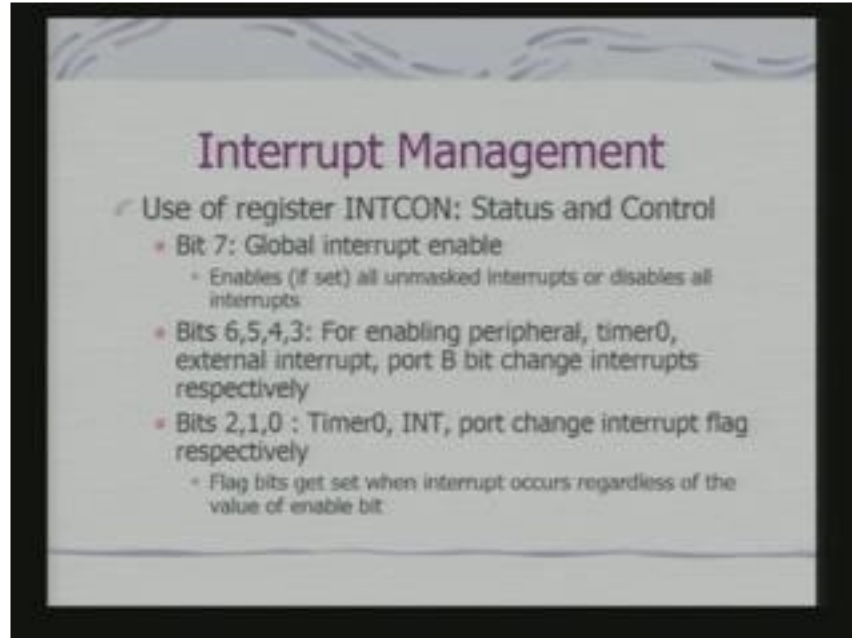
First, we can have a brief review of interrupt. What happens when interrupt occurs? The CPU stops its current execution and it starts execution from a different instruction stream which is targeted for handling the interrupt and when CPU finishes the interrupt, the execution returns to the original program. We shall see how this mechanism is really implemented in PIC. The sources of interrupt in PIC are many; some of these sources are definitely the on chip peripherals but also there is a provision for external signals to interrupt PIC.

(Refer Slide Time 03:29 min)



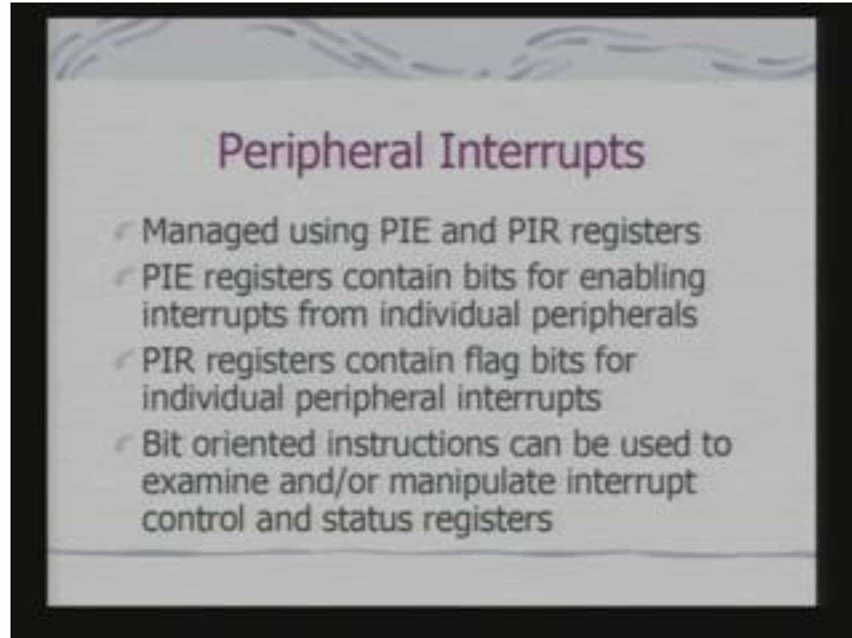
There is an INT pin interrupt from external source, as well as there is another interesting source of interrupt which is called port B change interrupt. If there is a change in these pins, it can interrupt the processor. And rest what I have listed are interrupts generated from on chip peripherals. We are timer overflow interrupts, USART interrupts when there is a character waiting in the buffer to be read, there can be an interrupt AD conversion interrupts when analogue to digital conversion is complete there can be an interrupt. The LCD interface module can generate interrupts and there are others also because as we increase the number of peripherals on chips when we need a mechanism to control the peripherals and they can also be source of interrupt for the PIC processor. And these interrupts, I have managed by use of few registers.

(Refer Slide Time 04:44 min)



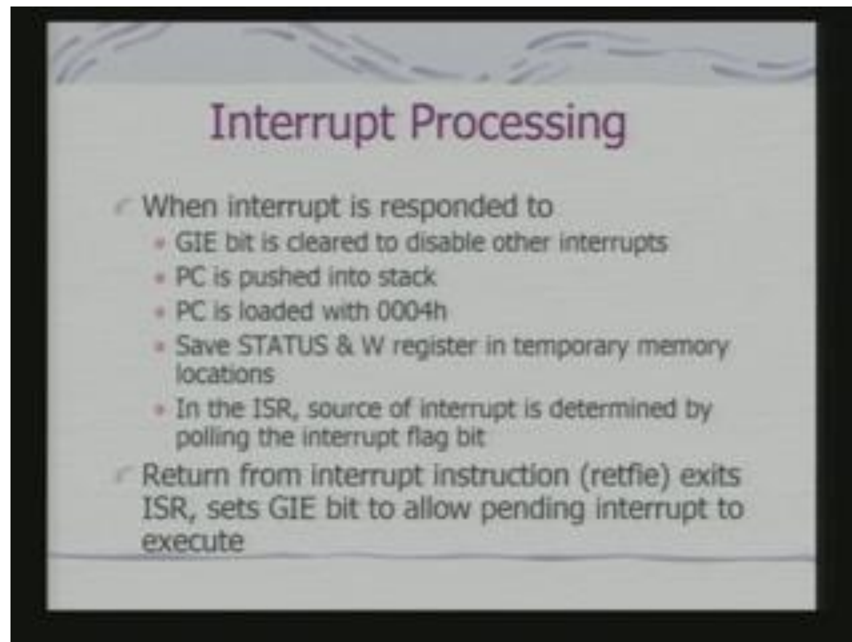
Now, let us look at the basic concept of maskable and non maskable interrupts. If you have a maskable interrupt, it essentially means that I have a mechanism to enable or disable that interrupt. Also another very interesting and important thing is concept of vector interrupt. When a vector interrupt occurs, the processor starts execution of the interrupt service routine from a fixed address. In fact, in PIC we have only vector interrupts. The INTCON register is used basically for managing majority of these interrupts. Also, there are other register's which are equally involved. I have listed the significant of the bits in the INTCON register. The bit 7 is the most significant bit because it is global interrupt enable disable bit
and if you disable this bit, the PIC will not respond to any pending interrupt. Bits 6,5,4,3 are for enabling, say peripheral interrupts,, timer zero interrupts external interrupt and port B bit change interrupts respectively. In fact when I enable peripheral interrupt, these correspond to again another set of interrupts because I can have many peripherals other than those which are listed here. For enabling or disabling those interrupts, I have a different register. The bits 2, 1, 0 in INTCON register are interrupt flag bits. They record the source of interrupt that means a flag bit gets set when interrupt occurs regardless of the value of the enable bit. If the enable bit is set that means the interrupt is enabled, the interrupt is responding to otherwise the interrupt may not be responding to, but the factor interrupt is occurred would be recorded in the flag bit. For peripheral interrupts, what I say that, once I enable peripheral interrupts, the source of peripheral interrupts can be various other peripheral. And I use the register called PIE which contain bits for enabling interrupts from individual peripherals.

(Refer Slide Time 07:29 min)



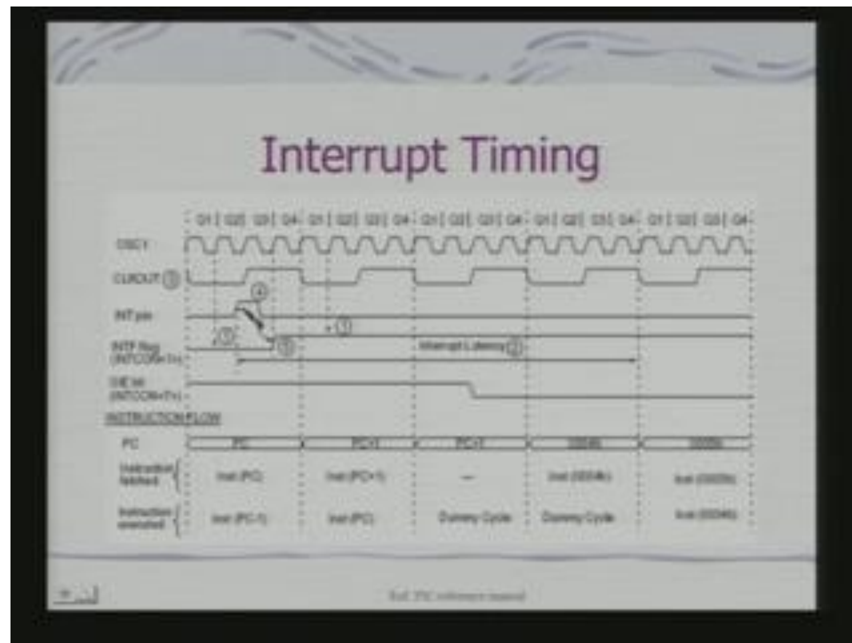
Associated with PIE, there is PIR register which content flag bits for individual peripheral interrupts. And in fact you can understand now again the utility of this bit oriented instructions that we have discussed in the previous class. Using those bit oriented instructions I can enable or disable; also I can figure out the source of interrupt by examining the flag bits of these registers. So, what shall I do to process an interrupt or what or PIC is supposed to do to process an interrupt. When interrupt is responding to, the global interrupt enable bit is cleared to disable other interrupts. PC is pushed on to the stack, this is again a hardware stack and now PC is loaded with a particular address and this is the vector for the interrupt service routine. And what you are supposed to do in your interrupt service routine is to preserve a copy of status, at least the copy of status and W register. Why this is important, because inside the interrupt service routine if you do any processing that can affect W register, that will also affect the status register and when you come back from the interrupt service routine to the main processing thread, you would like the original status and W to be restored back.

(Refer Slide Time 08:14 min)



So, while writing an interrupt service routine, you must safe W and status register if not other register in some temporary locations. Inside the ISR, the source of interrupt is determined by polling the interrupt flag bit, again here by using your bit check instructions. And when you return from the interrupt, you use retfie, few instructions which we had already discussed. Bit, this causes the processor to exit from ISR and set GIE bit that is global interrupt in other bit so that now processor can respond to pending interrupt. And this pending interrupt might have set the flags and should have set the flags and again in the ISR you can check the flag to determine the interrupt. So, actually what you find here is that your interrupt handler routine for specific interrupts should be jumped to or called to based on the flag bit that has been set and that flag bit is checked inside your interrupt service routine.

(Refer Slide Time 11:14 min)

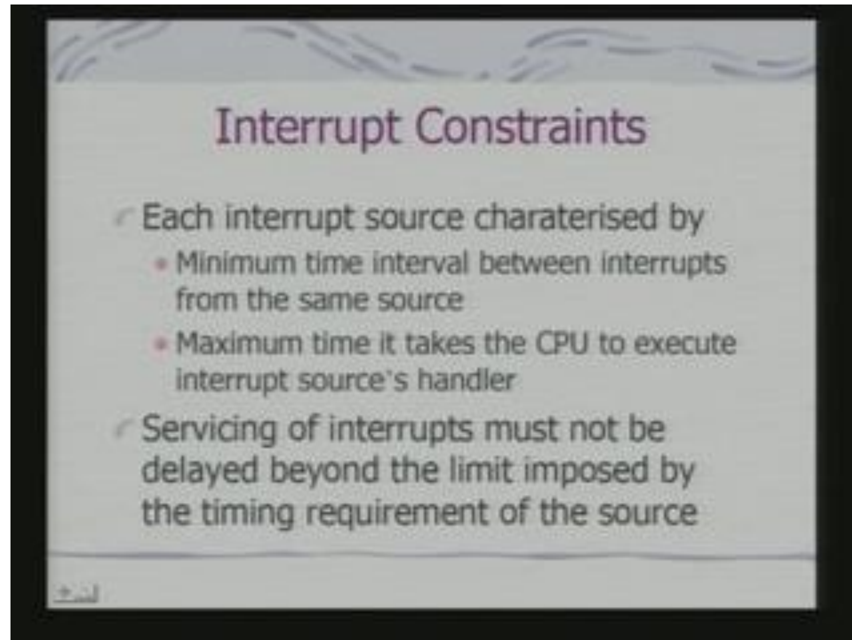


So, let us look at the interrupt timing for PIC. Now, let us say that this is my basic clock and this is your foreface clock which is controlling the entire processing of my PIC processor. We have discussed this clock in the context of instructions cycle also. Now, just consider that here, what we are looking at is the source of external interrupt along INT pin. So here, when the interrupt is occurring and what will happen? There would be a delay and at the delay INDF flag would be set. The flag is indicative of the source of the interrupt and what is assumed here is your global interrupt enable flag is set. That means the processor is enabled to respond to your interrupt. Now, the interesting feature is, that is the point I have got rising age of the INT pin to the point where the first instruction corresponding to ISR gates executed is typically your what is called interrupt latency. This is the hardware latency, okay and this latency is the basic delay which is involved between the raising of the interrupt and that of servicing the interrupt.

Now, what will find here when we look at the instruction flow, at this point I was actually executing the instruction which was pointed to by PC here. And I had already fetched the next instruction because that the basic pipelining in the PIC processor. But since interrupt has occurred, so what will PIC do? It has to respond to the interrupt and start execution of the instructions from ISR. So, you will find effectively a dummy cycle is introduced here which I was referring to as nop nothing happens in that cycle. And in this cycle, what you do is basically the instruction that you have actually fetched is not being used; so instruction fetch is, there is blank over there and the PC is effectively set to the vector address. Now, you fetch the instruction here and here also the execution wise it is still a dummy cycle because you have not got the instruction to execute. After this, at this point use start execution of the instruction from that vector location and you fetch the

instruction from the next location. So, this is basically the interrupt latency, that is the delay behind the servicing, okay. So, the gap, time gap between the interrupt occurrence and the service which can be provided. Now, this is a very critical component, this interrupt latency, because this is a hardware latency and there will be also a software latency. These two things have to be taken into account while designing an application because I need to satisfy interrupt constraints.

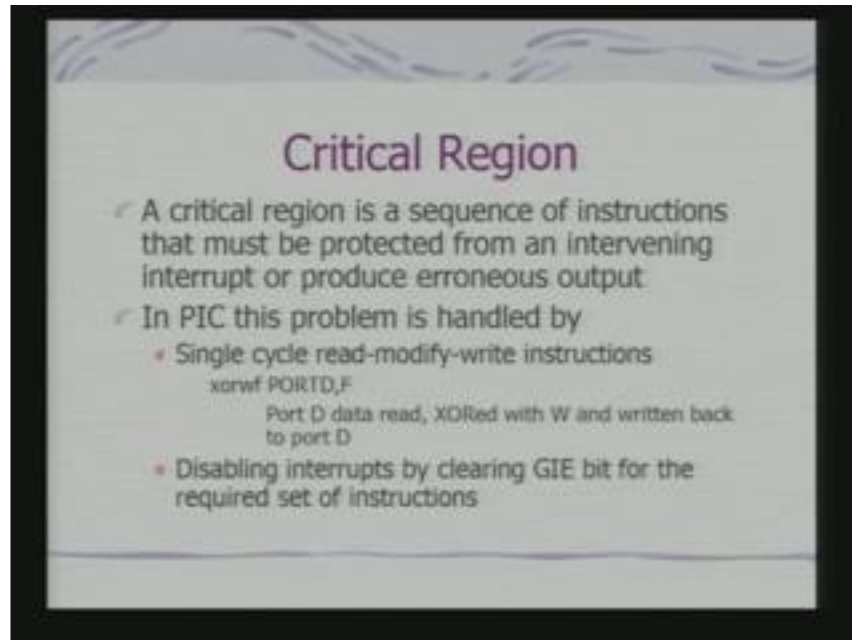
(Refer Slide Time 14:57 min)



What kind of constraints? Because, I am making a system which is situated in the external world and if we assume that there will be interrupts generated from the external source, I need to know the frequency with which the interrupt is expected to be generated. And so, each interrupt source which maybe an external source or for the case of micro-controllers can be peripherals which are on chip, I need to characterize each of the sources by the minimum time interval between interrupts and the maximum time it takes to the CPU to execute interrupt source's handler. And what is interesting to note is that the first instruction in the ISR is not really the instruction of the interrupt sources handler because I have to figure out the source by checking the flag bit and then only jump to the instruction which are supposed to service the source. And when I am inside an interrupt servicing routine, I am actually disabling the interrupt. So, if there are other interrupts occurring, they have to wait. So, when I am designing interrupt service routine, I have to be very careful to check what the maximum time that an interrupt service routine can take dealing other pending interrupts. And by delaying other pending interrupt I should not miss a deadline imposed by the external world and this is the key issue which goes into designing appropriate interrupt service routine. So, what is the basic bottom line? Servicing of interrupts must not be delayed beyond the limit imposed

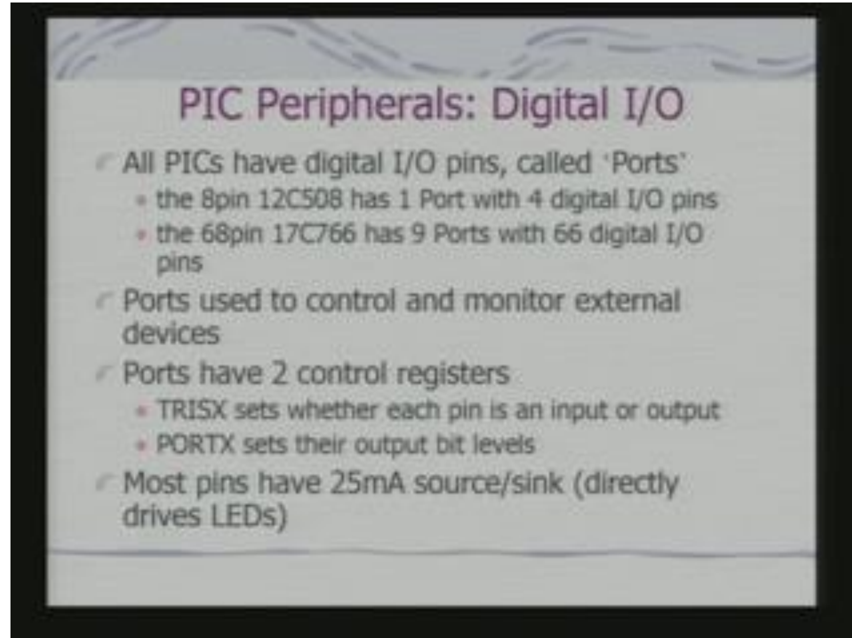
by the timing requirement of the source. So, when you are using an application and it is a dedicated application, so you know what are the resources of interrupt, what are the frequency and accordingly you should design your code so that you do not miss the deadline. And these deadlines is imposed by properties of the external source and you have to design code to meet the deadline taking account the instruction execution frequency as well as hardware latency. Next, we come to the concept of critical region.

(Refer Slide Time 16:58 min)



A critical region is a sequence of instructions, we have refer to that in the last class as well; is the sequence of instruction that must be protected from an intervening interrupt or otherwise it may produce erroneous outputs. And in the PIC this problem is handled in two ways. One is, we have got a single cycle read, modify, write instructions. So, in this case you read that means you your reading it as well as checking and modifying it in one cycle. So, after read there cannot be an interrupt, okay before you really modify there is change in the status. So, read, modify, write instructions help you to implement critical regions particularly with reference to your external world and external inputs. Other thing is obvious that if I disable my GIE bit over range of code, I shall ensure a critical region, but at the same time in this case and increasing the interrupt delay. So, that constraint I have to keep in mind. Also I have told you that we can also use a SWAPF instruction to implement this kind of synchronizing primitives to ensure correctness of critical regions. So, with this background on interrupts we shall now look at the different peripherals because many of the peripherals would be interface or can be interface using this intra processing structure of PIC. The most common peripheral is digital I/O.

(Refer Slide Time 18:43 min)

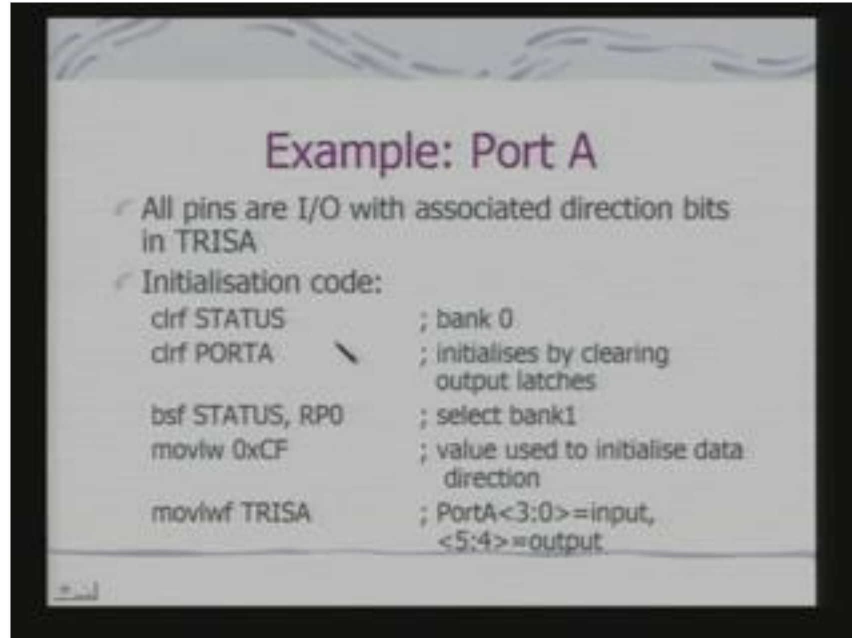


All PICs have digital I/O pins and we call them as ports and in fact there are external pins and these pins are associated with registers. So, whenever we want to do any manipulation on the port, one of the way of doing it is to read, write this registers. So ports typically are used to control and monitor external devices and in fact to read input or write output on to the external devices. And each port as I told you have to control registers. The TRISX is the port identity, if I have port A B C D, X will be replace by A, B, C, D sets whether each pin is an input or output pin that is the direction of input output. PORTX sets their output bit levels. Most pins have typically 25 million source or sink so that it can directly drives the LEDs.

(Refer Slide Time 20:13 min)

So, this is the typical configuration of port A, okay. So, this is essentially your data latch, I am showing it for a single bit. So, when we look at it for 8 bits or multiple bits, there will be multiple such latches and this is your TRIS latch which actually records the status that you set. So, what we say the setting of TRISA bit because I am referring to port A, put output drivers in high impedance state. And clearing a bit in TRISA, puts contents of output latch on the pin. When the output drivers are in high impedance state I can use that pin as an input pin, okay. And many cases even this digital I/O pins, there are some cases where you can even configure to get even analogue inputs for some other purpose because these pins are multiplexed for serving requirements of other peripherals as well. So, this is the basic structure you can see that these are the two latches which actually record the data as well as the TRIS bit enabling the direction of operation.

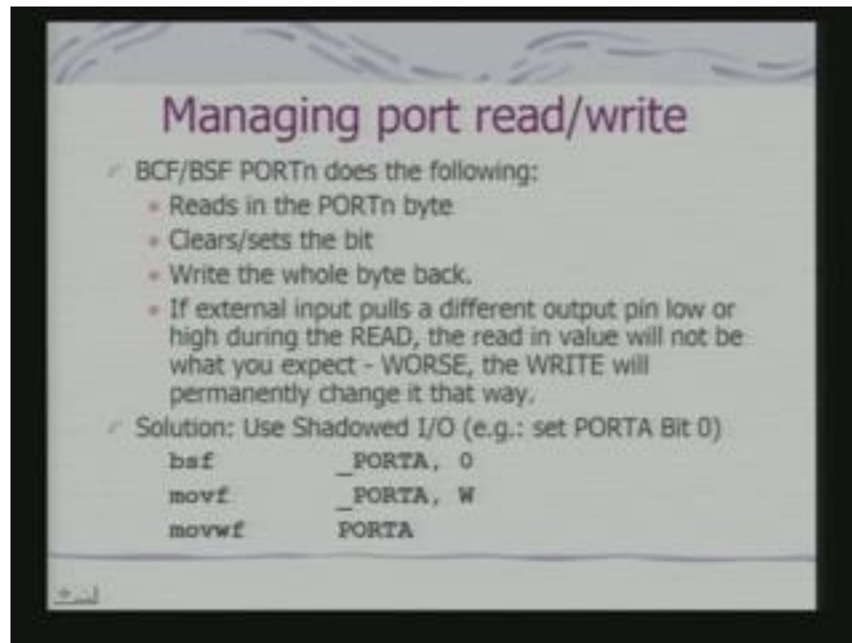
(Refer Slide Time 21:13 min)



Let us look at the code. So, what, what is happening here? This is an example of a code for initialization of a port. So typically, we clear the status; now, in this case obviously I am selecting the bank zero because these ports are also located on banks, memory banks, so I have to select the bank. Then I am clearing the port A. This is the corresponding data latch and then you are selecting the bank one, okay. And using the bank one you are writing what, you are basically writing on to this TRISA. TRISA is a control port which sets a direction of the pins on PORTA, this is clear. So, once we write the appropriate data, so these has been now programmed as that bit 0 to 3 are programmed as input and 5, 4 as output. So, these can be done by writing on to TRISA and this is, I am doing by selecting the appropriate port address along with the appropriate memory bank.

So, this is a typical code to be used for initialization of a port. Now, there is another problem associated with the code. Let us look at this read, modify, write instructions enable as to implement the critical region, but there also some issue involved here. Just imagine that you have READ and then there is an external input which pulls a different output pin low or high during the READ, because READ takes some finite clock time. So, reading value will not be what you expect.

(Refer Slide Time 22:58 min)

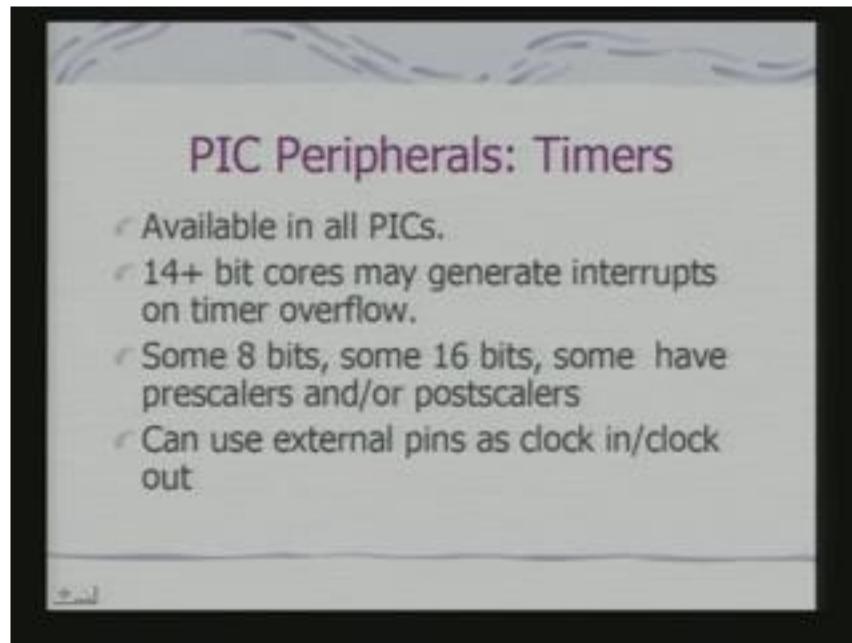


WORSE, the Write will permanently change it that way, okay. Because after write, the port, these bit value is to be changed and it could be changed permanently. So, these also can be source of problem and source of error when you are interfacing the external devices via port. So, one of the solution is that you use what is called shadowed I/O. That means you clear in this case, basically, when I am showing underscore PORTA it is not that I am writing on to exactly the PORTA memory location but a copy of PORTA memory location which is in general purpose set of registers which PIC provides for. And then what you do? You write on to W and from W you write on to PORTA. So, here what is happening is you are actually reading the data and whatever manipulations you are doing, you are doing on a shadow register. So, it is not been directly done on the PORTA register; only thing is that I am writing final result on to the PORTA register, okay. So, intermediate result will not effect the operation that you are trying to do.

Next, we look at timers. Timers are very important peripherals for any micro-controllers because timers are used for synchronization with other devices, taking care of various timing requirements for control application and there are a number of timers, in fact typically 3 timers on mid-range PIC microcontrollers.

And what we say the 14 plus bit cores means what? When you instruction length is 14 bits, these cores enable the timers to generate interrupts on overflow. Some timers are 8 bits, some are 16 bits and they have prescalers as well as postscalers. In fact, not all of them have both, some of them have one of, one of it and they can have even both. And they can use external pins as clock in as well as clock out. So, it is not that all these timers have been operated by the internal phase clock- the processor clock.

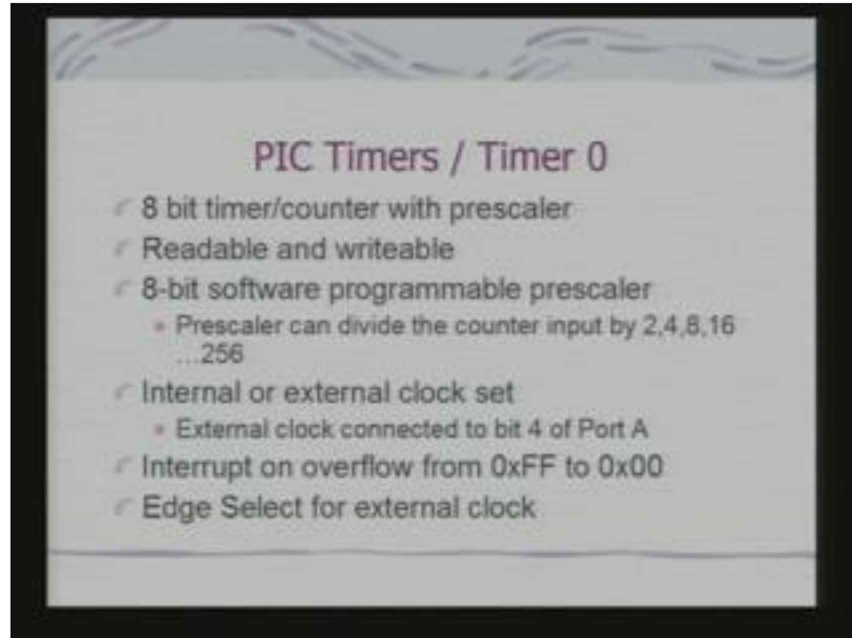
(Refer Slide Time 24:28 min)



So, let us look at timer zero, this is an 8 bit timer with prescaler. What is really a prescaler? Prescaler is basically an option for dividing the input that is the input which is coming to the counter by various factors. So, I can set the factors um maybe 2, 4, 6, 8; so I have an 8 bit programmable prescaler I can set a value to the prescaler. So, that would mean that input wave form is divided by that factor before its being fed to the timer. So, effectively what I am doing, I am increasing the range over which and the time period over which the timer can really count.

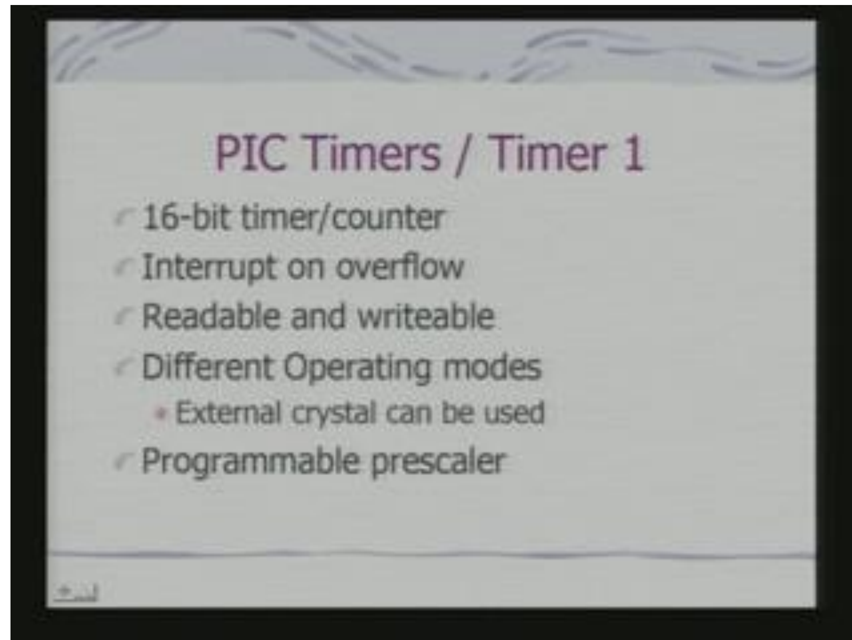
These timers are readable and writable, so I can read the current count as well as I can write to modified. And it can be set to internal or external clock and in fact, you can see that external clock is connected to bit 4 of port A.

(Refer Slide Time 25:13 min)



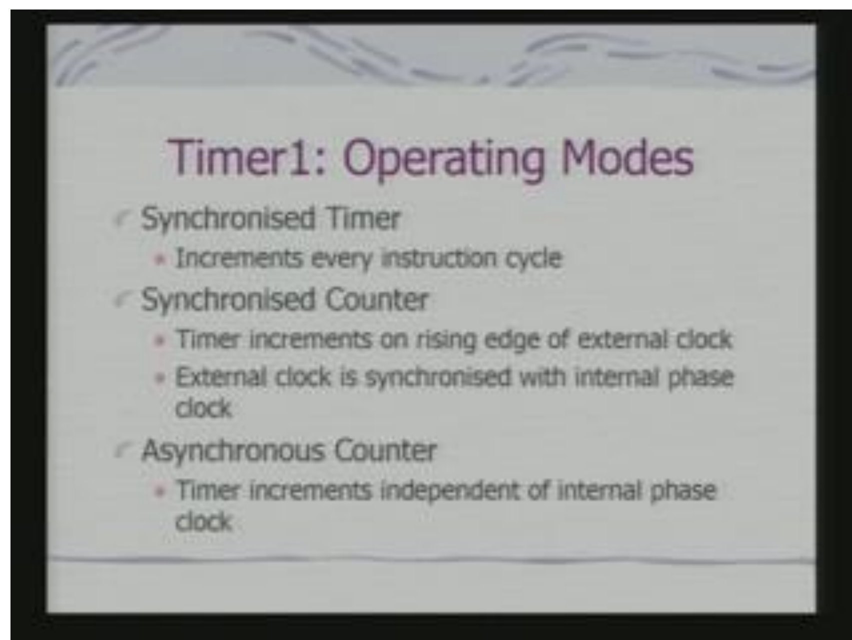
So, this is an example what I had already told you that these bits, I/O bits are multiplexed for various tasks with peripherals. This is an example where bit 4 of port A can be used as an external clock for the timer zero and it generates interrupt on overflow. Then, we have got another timer; now this is timer 1. This is a 16 bit timer, okay. Now, this 16 bit timer counter can also generates interrupts and overflow, just like it is other timer readable and writable it has got a programmable prescaler.

(Refer Slide Time 26:43 min)



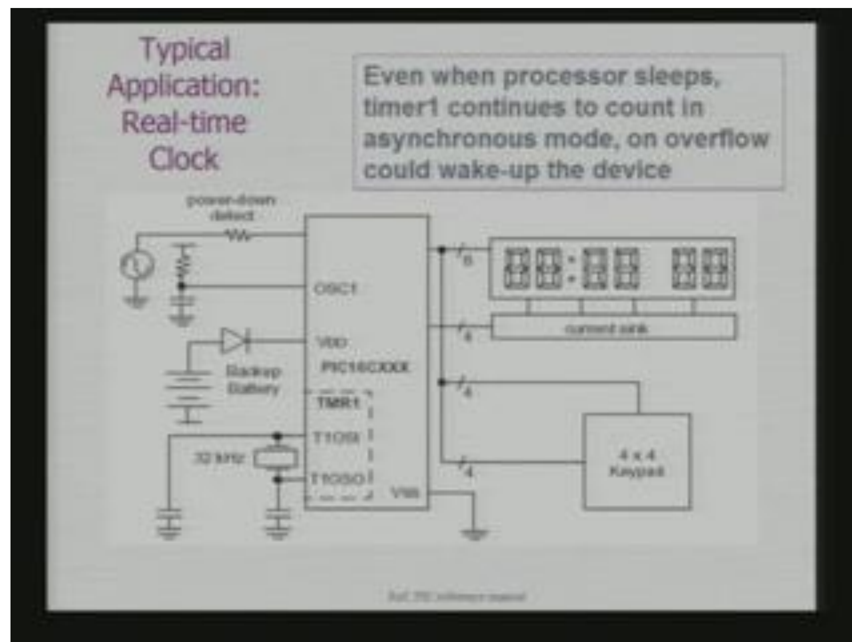
But what is the interesting is, it can operate in different operating modes. It can operate as a synchronized timer, it can operate as a synchronized counter, it can operate as a asynchronous counter. Now, there is a basic difference between them and these differences are interesting. Asynchronous counter, when it works as a synchronized timer, not counter it increments every instruction cycle.

(Refer Slide Time 26:58 min)



That way it is perfectly synchronized with the processor clock and you can count the number of instructions that is getting executed using this timer. It can also be used as a synchronized counter and in this case the timer increments on rising edge of external clock. But this external clock is synchronized with internal phase clock. So, what does that mean? It means that if we if we put some how the internal place phase clock off, then it will no longer count because it is synchronized with the internal phase clock. So, if the phase clock is off it cannot count. The other mode is asynchronous counter, the timer increments independent of the internal phase clock. So, I can feed it from an external source and it will continue to increment independent of the external phase clock.

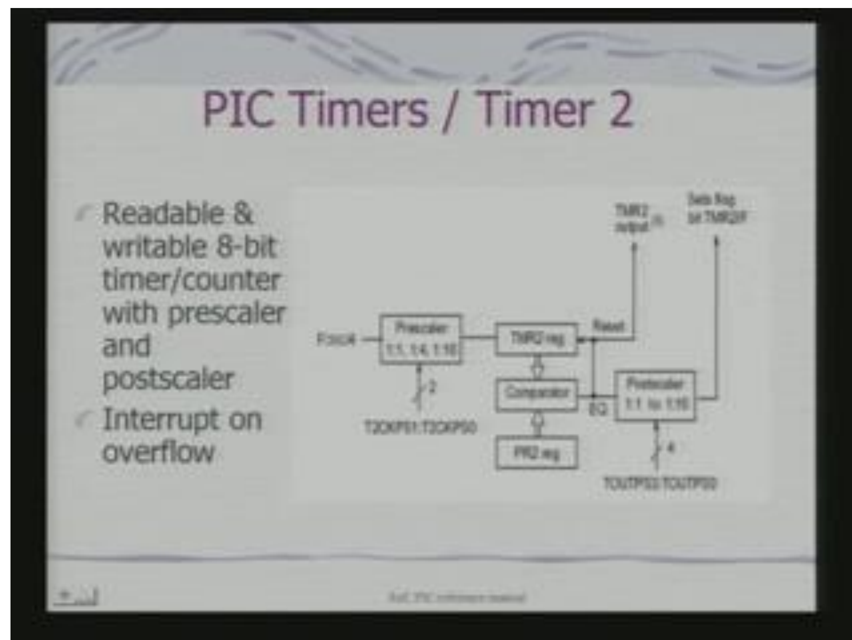
(Refer Slide Time 28:13 min)



Now, this is a typical application of a timer. I can build a real-time clock using this timer and this timer operating in a asynchronous mode. Because in an asynchronous mode, just look at it, in an asynchronous mode the timer needs to count and I can actually put my processor to sleep if my processor is not doing anything; if you have designed just a digital clock to display the time and do some other task but that is an occasional task and you may use a processor to do user interaction. So, in the majority of the time I can put the processor to sleep, okay. The moment I am putting the processor to sleep, I am saving

on NH okay and since it is an asynchronous clock it can operate from an external source of external clock signal, okay. So, timer continues to count and when it overflows it wakes up the processor. Once it wakes up the processor, the processor can do what; can update the display, okay. This is an LCD display and I told you that many of these PIC micro-controllers have LCD interface module. So, using that LCD interface module I can connect an LCD and I can update the display on the LCD.
So, the processor is not doing anything normally it just wakes up on getting the interrupt, okay on timer overflow and updates the display. And the timer works on its own from an external clock. So, I can connect external crystal as well to this timer; in fact what I have shown here is that we have connected on external crystal. These are the inputs to that timer one, okay. And these can be the keyboard, keypad interface for an user interaction with the clock. In fact, this you can consider to be a very simple embedded system built around your PIC. There is another timer, it is called timer 2 and timer 2 is readable and writable 8 bit counter with both prescaler and postscaler.

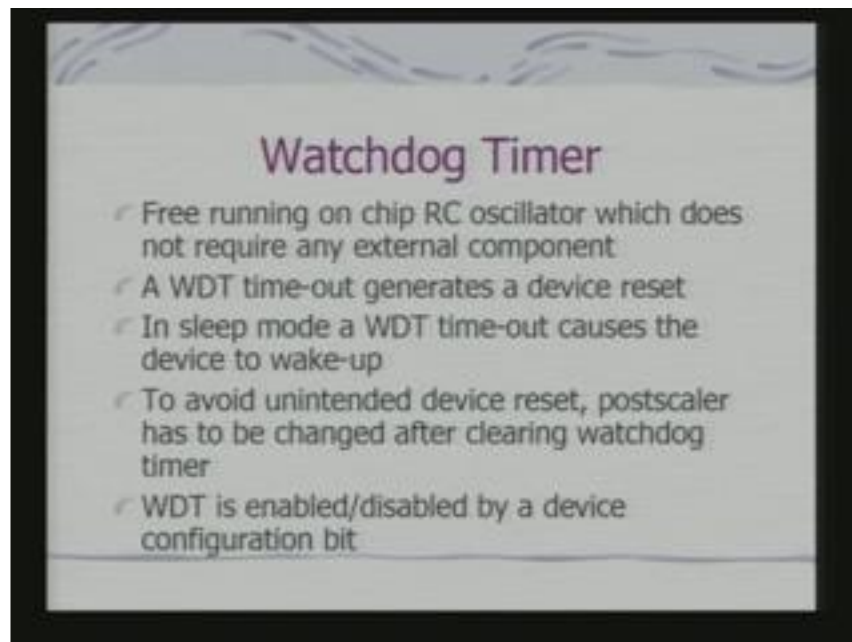
(Refer Slide Time 30:28 min)



Now, here we have shown its configuration. In fact, whenever we are using prescaler or postscaler this is the basic structure. So, what you find that the input here, I can set these different values to the prescaler. So, that means this will divide the input when it is being fed to the timer and a postscaler means what, so place postscaler also can be set to different values for a post scaling the output of the timer 2 and these bits that we are showing here, these are some of this control bits which can be set to set the different values to the prescaler as well as the postscaler. And in fact, this timer has got away the

interesting property we shall see soon and we have just shown that property here, a glimpse of the property but not exactly how it is used in the complete context. See, we have shown a PR2 register that is some register we can set. So, if I set the value to this register, this is a comparator which can actually compare the timer value with that of the preset value and if it is equal I can possibly pres reset the timer, okay. And I can change the value depending on the requirements and I can reset the timer on the fly, okay.So you can see this kind of a functionality is provided with the timer 2.
So, I can realize that although I am discussing 3 timers, a very basic question can come. Why 3 timers, why not one timer. You can see that these 3 timers perform and play different roles because of the different functionality, okay. So, and these timers that is why have been provided on chip on some of these micro-controllers that is PIC micro-controller to take care of different requirements.

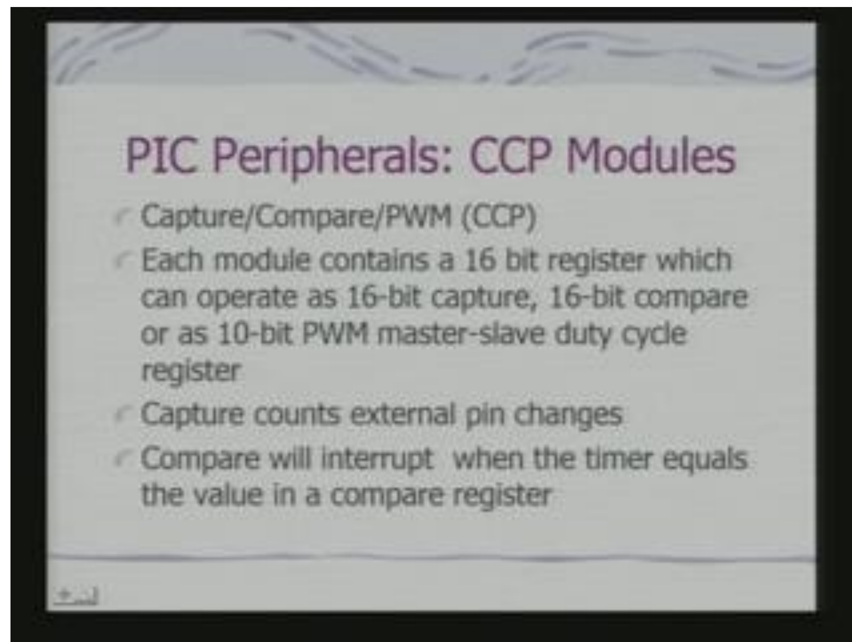
(Refer Slide Timer 32:57 min)



Along with the timer there this is watchdog timer which I told you is a very important component for majority of micro-controllers. Why it is important? Because when the micro-controller goes into some kind of an undefined state because of a software error or a hardware error, you need some mechanism to come out of that error state. And this watchdog timer provides such facility. So, it is a free running on chip RC oscillator provides the input for this timer and it does not require really require an external component as a clock source and time out generate a device reset. So, the device is reset

to the initial state. In sleep mode, a WDT time out causes the device to wake-up that when the device is sleeping if this time out causes the device will wake-up and it will start processing. And what you have to do is to avoid unintended device reset, we can have we need to clear the watchdog timer.
So, I have already talked about the clearing watchdog timer instructions and I can also associate the post scalar. By using the postscaler I can vary this time period, okay because after what will be the time period, after which watchdog timer should generate a time out that can be programmed through your, this postscaler. And, when you, by setting the postscaler you can change the period. So, by clearing the watchdog timer and setting the postscaler we can avoid unintended devise reset. Another interesting feature is I talked about if you remember there is a configuration port on the processor. And in the context stuff this assembler, I said that you have to also specify the configuration property so that when you doing the software development, the system knows what is the configuration in which the system is, the processor is set. So, one of the bit in the configuration port, we shall not have time to going to details of it, one of the port is, one of the bit in the, in that register is targeted for enabling or disabling the watchdog timer. So, if I disable the watchdog timer then all these things are of no consequence.

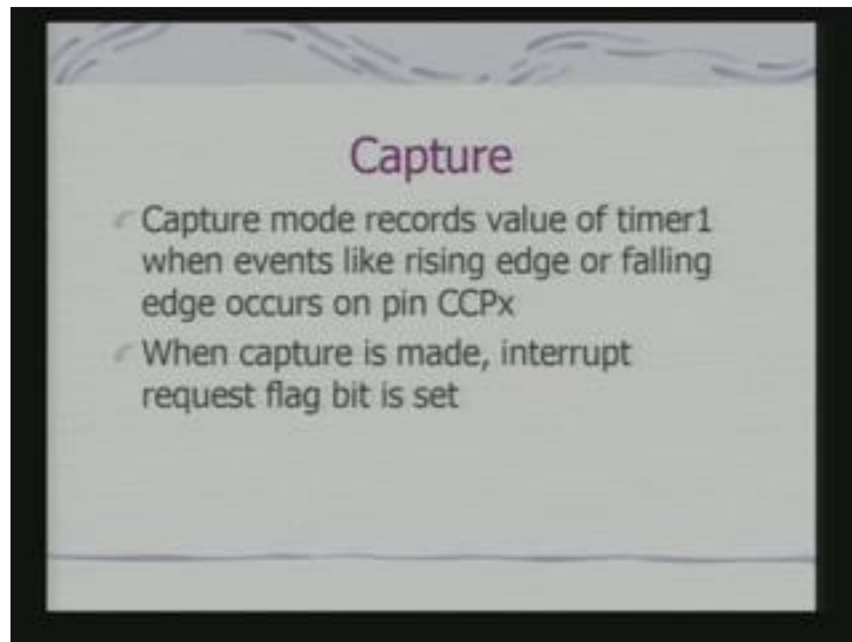
(Refer Slide Time 34:57 min)



Next, we shall look at another module which is called CCP module- capture, compare, PWM module. There maybe one or more such modules on this PIC micro-controller and this is again targeted for various kinds of control applications. Each of these module contains a 16 bit register which can operate as what we call 16 bit capture register, 16 bit compare register or as 10 bit PWM master-slave duty cycle register. And capture counts

external pin changes and compare will interrupts when the timer equals the value of a compare register. Now, what is PWM? Pulse Width Modulation and we shall see also its application, how it happens. First let us look at the capture. Capture mode records value of timer 1 when events like rising edge or falling edge occurs on pin CCPx. CCPx means depending on the number of CCP modules you have, it will be CCP1, CCP2 etc etc.

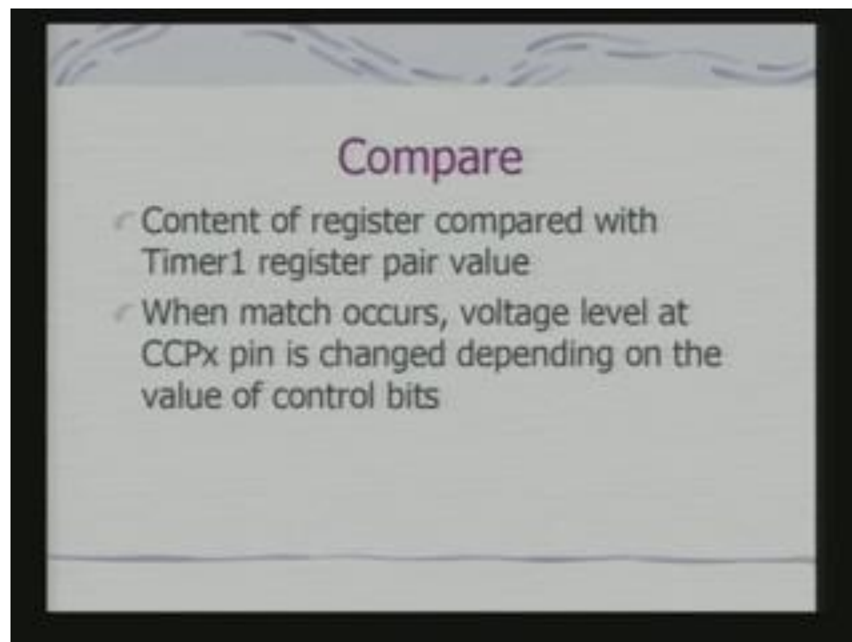
(Refer Slide Time 35:57 min)



When capture is made what happens, interrupt request flag is set; in fact it can lead to generate, this is one of the source of peripheral interrupt. Now, this is a very interesting feature what you are doing, that you are actually recording events and you are recording events with respect to time. This kind of feature is not commonly available in other processors, okay. You have not found, you have not, must not have encountered them

when you look at a general purpose processor like 8085 or 8086 and it is provided on this because it is supposed to take care of various external events interfaced with external world. So, it can actually keep track not only of the event but also the time, in a sense relative time based on the timer count when that event has occurred and that value is recorded in the capture register. Then, we have got a compare, the content of register is compared with timer 1 register pair value. When match occurs the voltage level at CCPx pin is changed depending on the value of control bits. In fact it can be, I can have a rising edge I can have a falling edge there may not be any change. So, I can program what happens when the comparison takes place, okay and when comparison is giving you equality as a value, okay.

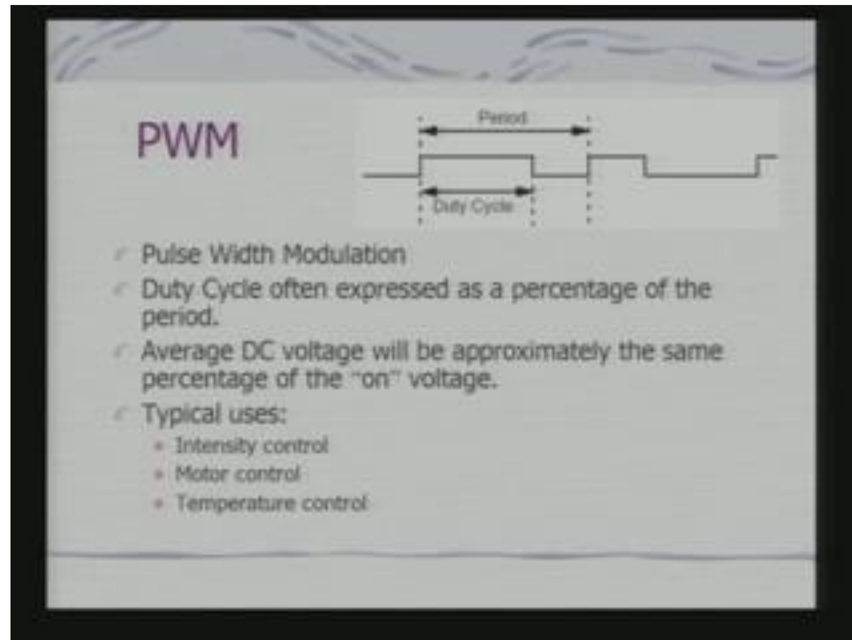
(Refer Slide Time 37:27 min)



So, the interesting feature here you can realize that you can generate also to this process external signal okay not only just I can possibly match and reset my timer. But not only that I can also generate an external signal to control an external device by taking care of this feature. So, that CCP module in that way is distinct from other kind of peripherals

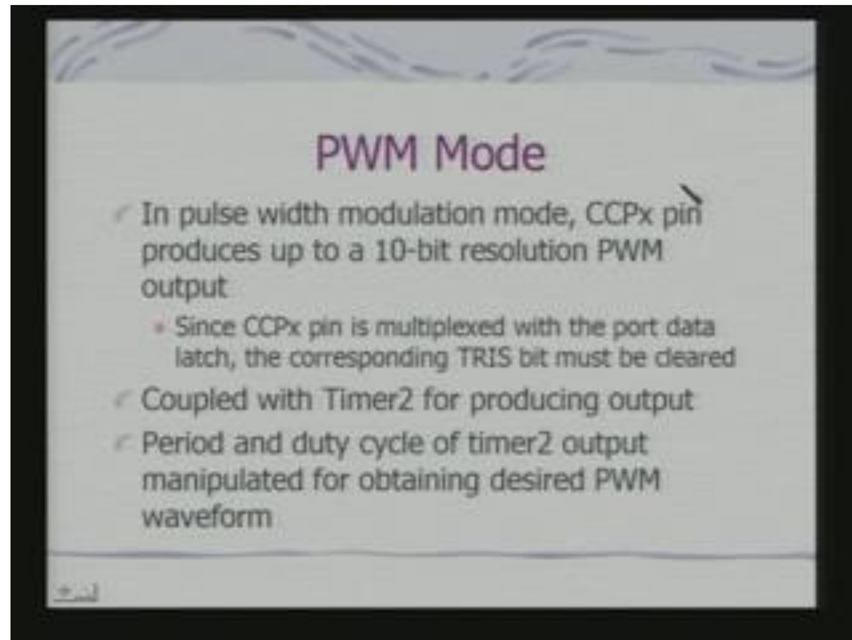
which you come only encounter. Next, we see look at the P part of it that is your pulse width modulation um feature of the CCP module.

(Refer Slide Time 38:12 min)



Now, what is pulse width modulation? All of us know that I can change in a very simple fashion, I can change the duty cycle of a square wave and that is precisely what we are doing when we say that we are modulating the pulse width and this duty cycle is expressed typically as a percentage of the total period. And if you change this duty cycle what will happen? The average DC voltage would change so what we can look at it and thought of as, average DC voltage will be approximately the same percentage of the on voltage and I can do more elaborate and actual calculations but conceptually the picture is like this that average DC voltage will be approximately the same percentage of the on voltage. So, now I can use these concept by, for intensity control. By changing the DC voltage I can control the intensity of a light source, I can do a DC motor control, I can do a temperature control that is controlling and hitting element and what I shall do in that process I shall just change the duty cycle and I need therefore on chip in hardware facility to control the duty cycle as well as period of this PWM. And that is exactly what, this is P module provides for. So, what we say that in pulse width modulation mode the CCPx pin produces up to a 10 bit resolution PWM output.

(Refer Slide Time 39:42 min)



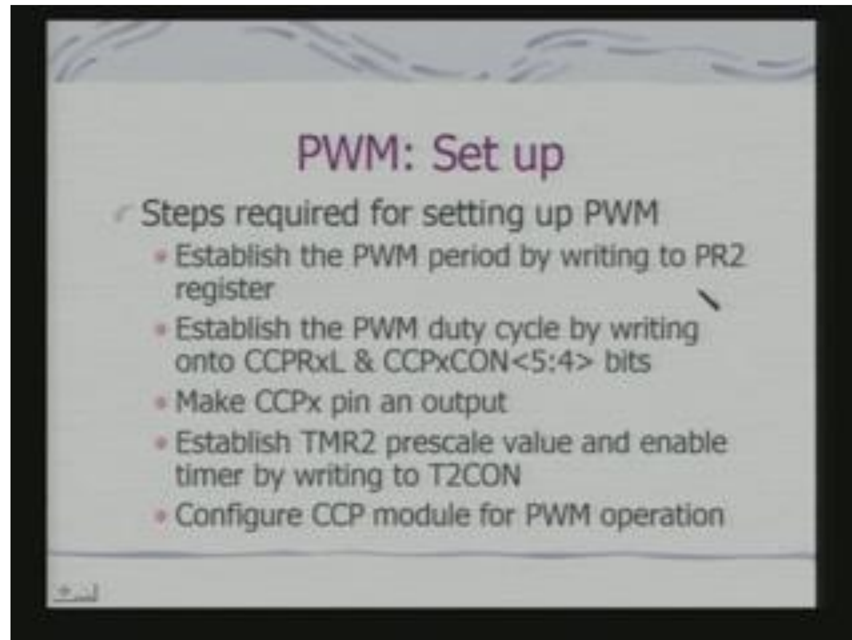
Since CCPx is multiplexed with the port data latch the corresponding TRIS bit must be cleared. This is again another point that of a multiplexing of the pins I talked about. And typically your PWM module is coupled with timer 2 and period and duty cycle of timer 2 output is manipulated for obtaining the desired PWM waveform. And this is again another important role the timer 2 plays in PIC. So, how it happens? Let us look at this Now, what you have got? This is your timer, okay and this is your timer 2 and this timer 2 can be compared with this PR2 that is at by that I can control the period, okay.

(Refer Slide Time 40:27 min)

And what happens is, here using this comparator I can also, I can control basically the duty cycle, okay. So, what we have talking about here is a duty cycle register. So, I can control the duty cycle, okay by through this part of the circuitry and I can control the period also to this part of this circuitry. And using this, what happens is I provide the output and I provide the output on the, one of this I/O pins and effectively I have to set my TRIS bit appropriately so that I get the output on that pin. So, what are the steps involved there for setting up PWM.

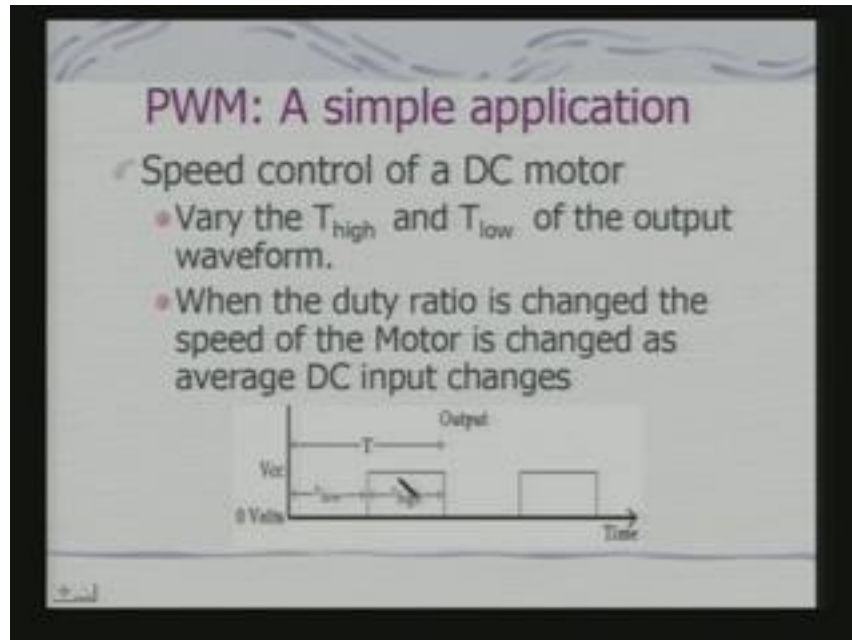
The steps required is establish the PWM period by writing to PR2 register because I have shown you that when I do the comparison that comparison would take care of the period requirement of the PWM waveform. Establish the PWM duty cycle by writing onto this 2 bits. We have to make CCPx pin as an output. We can, we need to establish the TMR2 prescale value and enable the timer by writing on to T2CON. In fact this we are talking about a timer 2; in fact all the timers have got a enable bit which we can set by one of these bits which have to be set to enable the timer and configure you CCP module for PWM operation.

(Refer Slide Time 41:27 min)



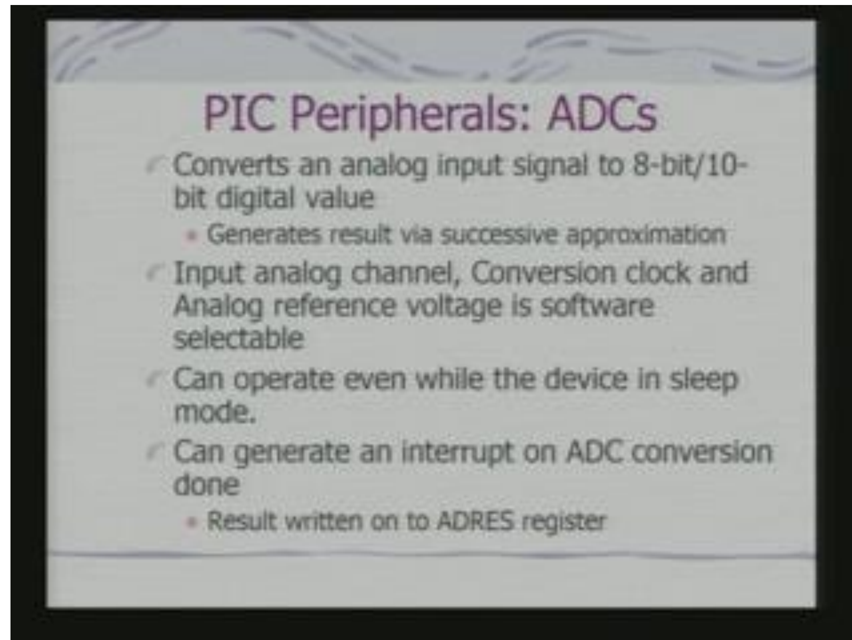
So, if you go back to the previous slide, we can see that what we are talking about, we are talking about setting the duty cycle, setting the PR2 for determining the period and what we have not show in the prescaler; so, we have to set the prescaler and to get the PWM in the waveform. So, once we set it up we get the PWM in the correct form. So, what does that mean effectively. If we now look at this application of controlling a DC motor, effectively the speed of the motor would depend on this ratio of T high and T low. So, that means when you want to change the speed, what you need to change? I can change if I want to keep my period unchanged and change the duty cycle, I have to change the corresponding bits. If I change my duty cycle, my speed can also change because average DC voltage would actually be more. This is a very simple way by which I can directly use a PIC micro-controller for a DC motor speed control applications.

(Refer Slide Time 42:57 min)



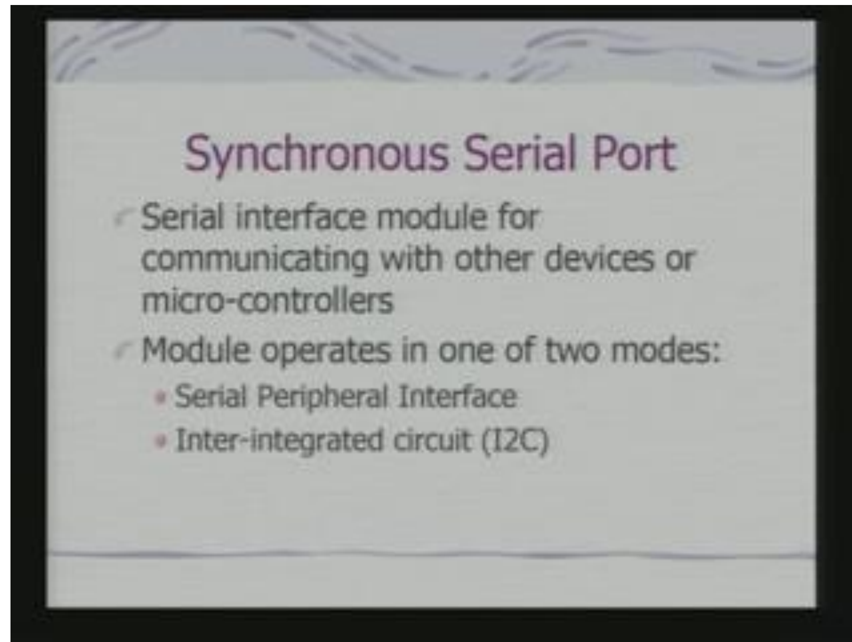
It can be use a variety of other applications as I already told you, intensity control of a light source, temperature control of a hitting element so on and so forth. In fact one of the applications, major application for which PIC was designed was control applications and that is one of the reasons why you expect and you have got the CCP module on chip of PIC. There are other PIC peripherals, very common PIC peripheral is analogue to digital converters; so which converts your analogue input to 8 bit or 10 bit digital value. And here, this analogue to digital converters at typically successive approximation ADCs.

(Refer Slide Time 43:57 min)



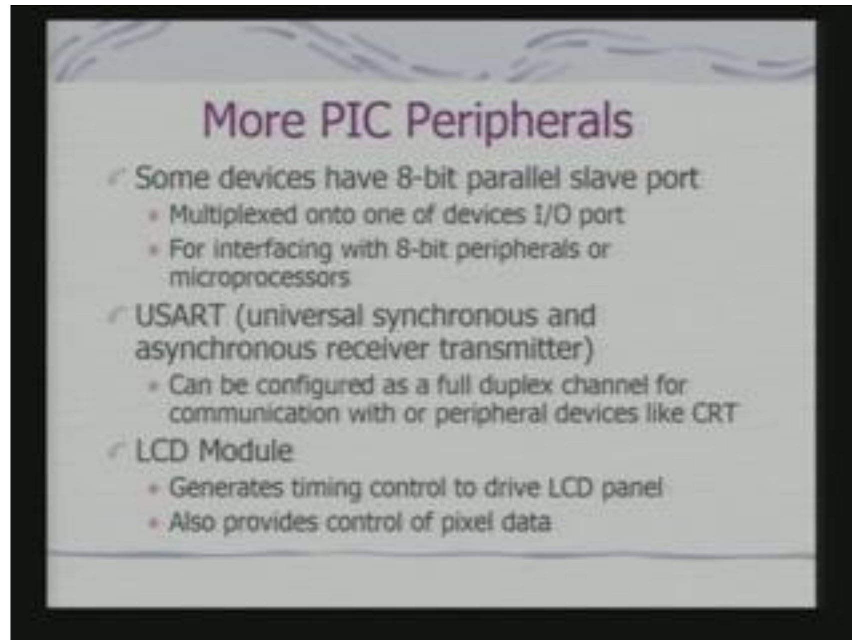
You can select your input analogue channel, your conversion clock as well as analogue reference voltage in a software selectable fashion. Now, interesting feature is this ADC can operate even when device is in sleep mode and once the conversion is over the ADC generates an interrupt and it writes a result on to a special register which is called ADRES register, not address register, ADRES register where the result is putting. So, this gives you a kind of a flexibility on it. Typically, it was 8 bit as well as your 10 bit ADCs are available. Now, the reason for providing ADCs are interfacing with external analogue inputs, many of your external sensor inputs would be analogue. Once you get the analogue input, we actually convert into digital form for doing any further processoing. There is another module which is called synchronous serial port module.
Now what is this? This module is primarily targeted for, this is SPI module; in fact this can be used for setting up your SPI as well as inter integrated circuit interfaces.

(Refer Slide Time 45:27 min)



In fact these are, these two are specifications of buses by which you connect other devices as well as other micro-controllers to your micro-controller. So, in a sense these are bus interface modules and we shall not discuss much about it here. we shall discuss it later on when we talk about interfacing devices to the micro-controllers because what we have discussed really is till now is the peripherals which are on chip and how this peripherals on chip peripherals can be interface and used by the processor. And these module, this again a non chip module which provides you a mechanism for connecting other peripherals and other processor with your micro-controller. There are other PIC peripherals, okay, what we have discussed so far. Now, one is 8 bit parallel slave port, okay. Now if you look at your digital I/O pins or the ports that we have discussed, what is the difference with them, difference is they are actually, we have considered and we have used them as a single bit as a single bit I/O ports, okay. You can actually manipulate, I can set individual bits to be either in input or output mode separately and I can read them.

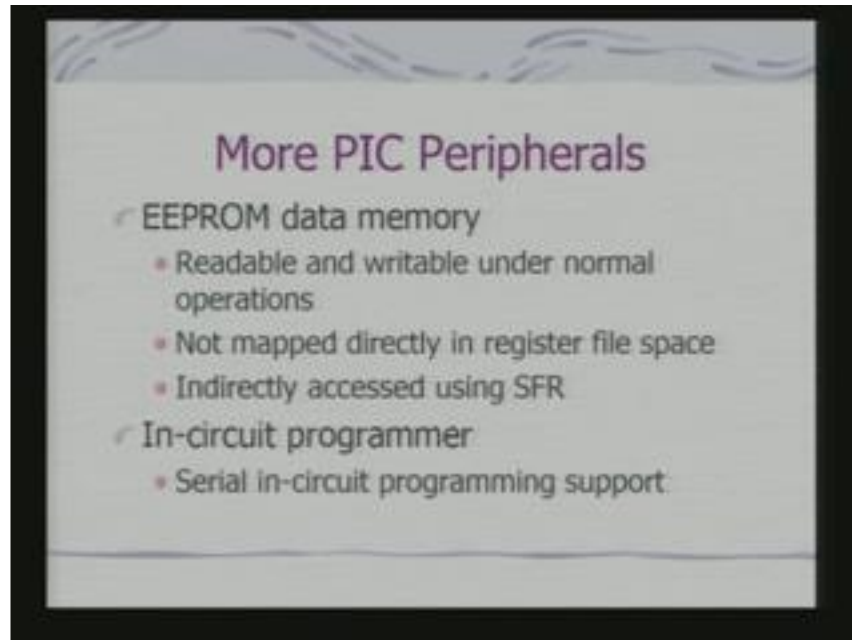
(Refer Slide Time 46:57 min)



Now, what we are talking about here is a eight bit parallel port and typically for interfacing with 8 bit peripherals or microprocessors. And they are, these ports are typically multiplexed onto one of the devices I/O ports. Now, why this kind of a port is at all required; simply because there are lots of peripheral chips which are available in the market which you have targeted for interfacing with 8 bit microprocessors and the PIC wanted to have a facility to use them as well; so this 8 bit multiplexer parallel port provides for that. There is also an USART module which is universal synchronous and asynchronous receiver transmitter module.

It can be configured as a full duplex channel for communication with peripheral devices like CRT. In fact CRT or PC you can communicate through USART. So, in fact this USART is essentially implements your RS232C protocol which is typically you will find with most of your serial interfaces on the PC or other devices. It also has got as its LCD module. The LCD module requires what, a mechanism to time, to do a timing control for the purpose of display. So, it provides a interface for the timing control and also provides the control of actual pixel data because whatever data which is being written on to the pixel how it is to be written onto and this pixel data would mapped to a memory area. So, the LCD module has provision for this timing control as well as management of pixel data. So, in fact the example that I had already shown you the clock that means the real time clock you can implement, but picking of a simple PIC and interfacing a LCD module and putting in a crystal and obviously by writing an appropriate software and loading that appropriate software on to the PIC. You really do not need any other components.

(Refer Slide Time 49:56 min)



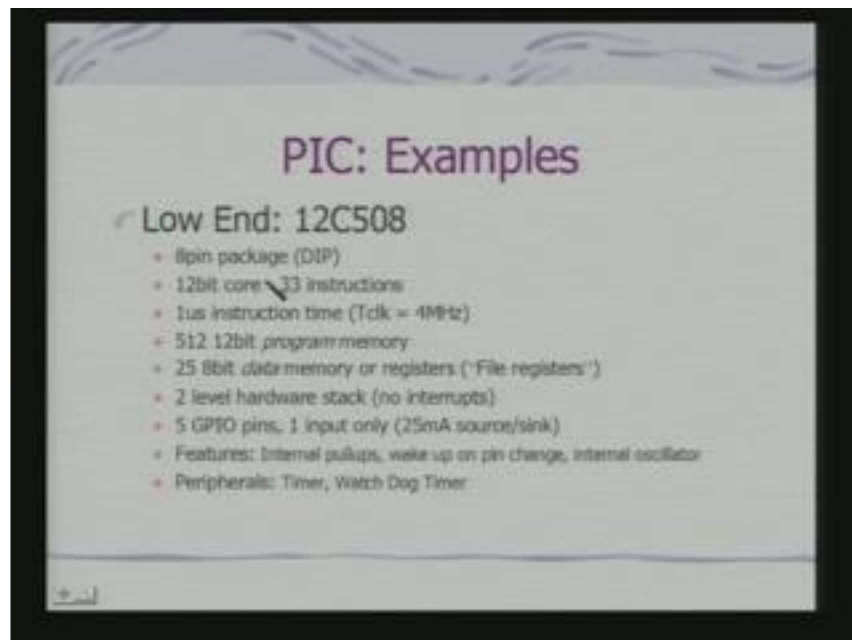
The, another interesting PIC peripheral is that of EEPROM that is electrically erasable PROM data memory, okay. Now, so for we have seen data memory and discussed data memory as registers which were RAM area. Now, we are talking about electrically erasable, in fact this provides you facility for storing the data even when the power is out or in some cases you have not operated the system and you want to record the status for subsequently bugging so on and so forth. Now, these registers are readable and writable under normal conditions. What you really mean by normal conditions? Normal conditions means under normal supply voltage, DC voltage conditions you can access this locations.

You do not require any other supply voltage for accessing these locations, but these locations are not mapped directly in register file space. And what you need, you actually would be using indirect addressing their would be a specific register, okay onto which you will be writing that address and that register is again a special function register. If you remember I said that because got both general purpose registers as well as special function registers. So, this will be one such special function registers and you will be using indirect addressing to access these memory locations. So if you want to records some data, you will be using these locations to record and store that data. Also the higher version of your PIC micro-controllers have got in-circuit programmer.

Now, serial in-circuit programming support. Now, this is interesting and important because when you are developing software, okay. You are developing a software and we can assume that you will be developing a software either in assembly language or even in C high level language and you translate that to assembly and finally to the machine code. And your program area is either EPROM, okay or it is a flash. Now, you need a mechanism to load your program onto the, the program memory area. So, this in-circuit serial in-circuit key programmer provides you that facility; it provides a serial interface by which once you have tested the program in assimilation environment maybe on your

host platform you can load your application onto PIC and you can execute that application on PIC itself.
So, obviously it reduces that wants I have in-circuit programmer facility on the chip itself, it reduces what, what we call time to market. It is not that I have to develop the software and give it to the manufacturer to preprogram it on to the processor. I can get the chips onto my site, I know I have got the software tested and I shall just program it on a production line and get the output program.
So, that reduces the time to market, reduces, also increases the flexibility of usage of these PIC micro-controllers. In fact I have told you that when I use flash version that is also another advantage. So, now I told you, what I have told you so for is a general description about the PIC processors, its instruction set and the peripherals.

(Refer Slide Time 53:56 min)



Now, we shall look at some specific examples of PIC micro-controllers. A low end example is 12C508, okay and you see that there is a restricted set of features that we have so far discussed. Not all the features are present here because it is targeted for low end applications and it will have low cost. So, it has got a 512 only 12 bit program memory; it operates at 4 megahertz. It has not got 35 instructions but 33 instructions. It is typically an 8 pin package. It has got 5 GPIO pins and interesting only two level hardware stack, no interrupts, okay and it has got peripherals, simple timer and watchdog timer. Watchdog timer should be there to recover and the timer for a simple application. So, for a very simple application we can use this PIC, okay. So, you can, the interesting feature is this using this basic architecture feature you can get a variety of processors, okay. Conceptually all these processors support similar instruction set, similar architectural feature and similar way of interfacing with the peripherals. Only the set of peripherals as

well as the reported instructions the data size data memory size as well as the program memory size changes.

The other one is the mid-range which is a bigger one and in fact our discussions had been primarily centered around this mid-range PIC processor, okay. So, it has got operating frequency 20 megahertz, it has got 35 instructions, it has got a 14 bit instruction set, it is also flash so that program can be updated on the fly. It has got 8 level hardware stack, it has got electrically erasable PROM, that is data resister that I had already talked about. It has got 10 bit ADCs. It has got 22 general purpose I/O lines, port A, port B I/O lines that I was discussing. It has got USART, it has got SPI, I2C all these peripheral interfacing, serial interfacing mechanism that I talked about. It has got both 16 bit and 8 bit timers as well as it has got brown out detect and in-circuit debugger. What is brown out detect? When there is a voltage level fluctuation, one is power out, when is power out I got the reset say, brown out detect is that when there is a voltage level fluctuations I can detect that and I can go to a predetermined state. So, this finishes our discussions of PIC family or processors. We have discussed architecture, instructions as well as peripherals. And PIC processors are well suited for low end and mid-range applications. In the next class, we shall look at 32 bit processor where, which are targeted for primarily high end applications.

(Refer Slide Time 56:26 min)