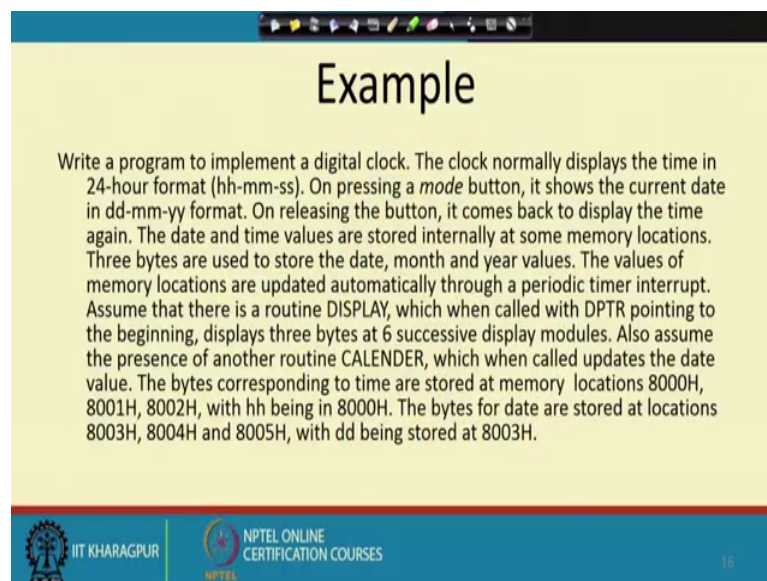


Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 40
8051 Programming Examples (Contd.)



Next we will look into another program where it will try to model one digital clock. So, in digital clock has to be constructed; so, statement of the problem is like this.

(Refer Slide Time: 00:22)



Example

Write a program to implement a digital clock. The clock normally displays the time in 24-hour format (hh-mm-ss). On pressing a *mode* button, it shows the current date in dd-mm-yy format. On releasing the button, it comes back to display the time again. The date and time values are stored internally at some memory locations. Three bytes are used to store the date, month and year values. The values of memory locations are updated automatically through a periodic timer interrupt. Assume that there is a routine DISPLAY, which when called with DPTR pointing to the beginning, displays three bytes at 6 successive display modules. Also assume the presence of another routine CALENDER, which when called updates the date value. The bytes corresponding to time are stored at memory locations 8000H, 8001H, 8002H, with hh being in 8000H. The bytes for date are stored at locations 8003H, 8004H and 8005H, with dd being stored at 8003H.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

So, we have to implement a digital clock; the clock will normally display the time in 24 hour format in hh-mm-ss hour minute second format and there is a mode button. So, if the mode button is pressed; so it will show the current date in dd-mm-yy format.

So hour, time and date if you the button is released. So, it comes back to display that time again; this date and time value; so they will be stored internally at some memory location. Three bytes will be used to store the date, month and year values; so dd mm and yy values and the values of the memory location; similarly for this hour minute second also will three more bytes. So, total 6 bytes will be used for storing this hh-mm-ss; dd-mm-yy these values.

And then this location; the values of these locations will be updated automatically through a periodic timer interrupt. So, 8051 has got the timer interrupt; so, at periodic

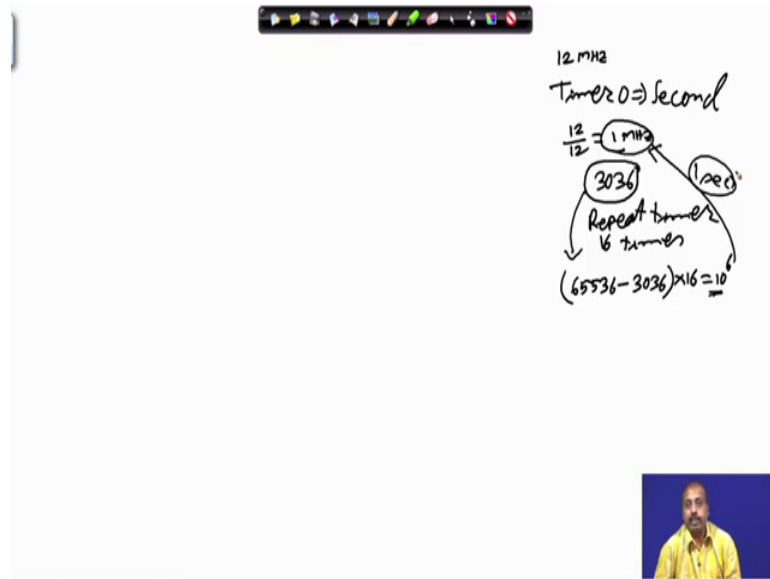
intervals that interrupt should come and it will update this timing. And assume that there is a routine display; so, which when called with DPTR pointing to the beginning displays three bytes at 6 successive display modules.

So, there are 6 display modules are there; may be several segment display or something like that. So, this hh-mm-ss will be displayed; so, these are the three successive bytes they will be displayed on this 6 successive display modules. And if the DPTR points to the location containing this dd-mm-yy then it will be display the date part. And if it is a DPTR points to the hour minute second part; then it will be displaying the hour minute second.

Also we assume that there is a routine called calendar which when called updates the date value. So, naturally this calendar routine is also quite complex where it will be updating the; after every second the interrupts will come. So, accordingly this ss-mm-hh they will be updated and this dd-mm-yy those values will also be updated. So, it is a very complex routine; so, we will not go into that. So, you can write that code separately, but our purpose is to show how this interrupt and these display modules they can be integrated into the system.

Finally the bytes corresponding to the time are stored at location 8000, 8001 and 8002. So, hh is an 8000; mm is an 8001, ss is an 8002 and then the bytes corresponding to the date dd-mm-yy; they are at 8003, 4 and 5. So, how to design this particular program? So, let us see how to do this.

(Refer Slide Time: 03:13)



So, we assume that the crystal that is connected to the system is say 12 megahertz crystal; for the ease of calculation. So, we will take that eleven point something the fractional value of that crystal frequency; so, we will take it as 12 megahertz. So, we will be using this timer 0 for counting seconds; so, after every second I want to get an interrupt and then I should be updating the second value.

So, we should get a clock frequency of 12 by 12; so, you know that this for the timer. So, the system clock is divided by 12 to get the clock reaching the timer. So, the clock that reaches the timer is 1 megahertz; so, if we set the value of that timer to 3036 and repeat the timer for 16 times. So, this is the timer value that is set and repeat timer 16 times then you will get about; so every time you set this value. So, you get it time value of 65536 minus 3036 this into 16; so, this is the total after. So, this is part time; so, if you repeat it 16 times it is into 16.

So, this value is about 10 power 6; so, this will be; this is 10 power 6. So, this will be giving me; so, it is matching with this 1 megahertz. So, this delay will be matching to that 1 second that we are looking for. So, for designing the timer part I have to use this timer with the initial value as 3036 and then repeat the timer 16 time to get 1 second delay. So, after the timer interrupt has come 16 times; I should update the second field and then that updation routine will be updating the second field minute field etcetera, the update routine should be called accordingly.

So, this is actually this is the proper way of writing interrupt service routine because in interrupt service routine; so you may be using some registers now you do not know from which point in the system; the interrupt has come. So, that main; the program that was executing there might be using those registers.

So, it is better that whatever register you are going to use in your ISR code; at the beginning you save all those registers in the stack and before coming back from the subroutine you just restore all those registers. So that the program which got interrupted will be able to continue without any problem; so though in our previous interrupt service routine, so we have not done this thing. So, ideally you should do this for all ISR; whatever register you are using, you first save those registers into stack and before coming back; so you restore all those registers from the stack.

Then in the R 1 register; we load the repeat count. So, MOV R 1 comma repeat count. So, that repeat count will be using as a counter for that 16; then increment R 1 and we check whether the value has become 16 or not. So, CJNE R 1 comma hash 16 comma L 1; so, if the count is not 16 then I do not have to do anything, no updation or time will take place. So, I have to skip over the next part of the code; so this part of the code will be valid when we will be coming to updating; this part whatever you are writing now so, they will become useful when this count value has become 16; so, that the second has to be updated.

So, for updating second what we do? So, first we reset this R 1 to 0; MOV R 1 comma hash 0, then MOV; the value 8002 will be obtained in the DPTR because 8002 is holding the second value. So, 8002 is holding the second value; so MOV DPTR comma hash 8002 H so, that this is MOV X; sorry no this is not MOV X, this is MOV and then MOV X; A comma at the rate DPTR; that means, I have got the second value into the A register.

Now I increment A because 1 second has passed; so, I increment A; so, that is I need to check whether the A value has become 60 or not. So, if it is 60; that means, I have to update the minute part. So, CJNE 60 comma L 1; so if it is less than 60 nothing is done; so the updation is over, but if it is 60 then it will come to this point a clear A; the accumulator is cleared and then I need to save this new value of this accumulator into the second location; the location corresponding to the second part.

So, MOV X at the rate DPTR comma A; so it will put 0 on to the second part. Then I have to look into the minute part; so, for getting the minute part I have to MOV A comma sorry this is not A; MOV DPTR comma hash 8001 and then I should get. So, this now if I do a MOV X; MOV X A comma at the rate DPTR. So, I will get the minute part into the accumulator.

And since the value; the second was over, so that 60 was over; so, I need to increment the minute part. So, the accumulator is incremented now again I need to check whether these value has becomes 60 or not. So, if it has become 60 then the hour value has to be updated otherwise this is fine. So, again we compare CJNE A comma hash 60 then if it is not 60 then nothing need needs to be done, but if it is equal to 60; then I have to clear A then store this pattern onto the memory location for this minute and then I have to do the update.

So, that is I need to do a MOV X this at the rate DPTR comma A; then I have to update the hour part. So, hour part is available in location to 8000 H; so MOV DPTR comma hash 8000 H fine then MOV X; A comma at the rate DPTR. So, this will take the hour part into A register and now I need to increment A.

So, if I do an increment A; that means, the A value will be updated; so this hour part is updated. So, now, I need to check whether the hour part has become 24 or not. So, that CJNE A comma hash 24 comma L 1; if it has become equal to 24, in that case I need to clear the accumulator and MOV this 0 value to the hour memory location.

So, that will be done using this MOV X instruction; MOV X at the rate DPTR comma A. So, that register is cleared and also that location is cleared; now I have to update the calendar. So, I said that there is a routine called calendar which we will be able to do that; so we are not writing that routine. So, we are giving ACALL to the calendar routine for doing that; so, ACALL calendar will call that routine and it will be updating the calendar part.

So, done now that L 1 onwards that portion has to be written where I do not need to do anything. So, there actually the value; the second value or the minute value or hour value is updation has been over. So, I do not need to do anything; so what I do? So, MOV X at the rate DPTR comma A. So, A register was holding the hour value or minute value or

second value, the updated part and that is moved to the DPTR and this repeat count should; this R 1 has been incremented to hold the repeat count, so that has to be restored.

So we MOV this R 1 to the repeat count memory location; so MOV repeat count comma R 1; so, R 1's value will be moved to the repeat count memory location then I am done. So, now I have to restore all those registers, so it should be in the reverse order; in which we have saved them like POP 01 H; then I have to POP; the accumulator register POP 0E0 H; then that 83 H; POP 83 H; then pop 82 H; so, those locations are popped out.

Now, this TL and TH; so they are to be again loaded with the values, so now actually; so this part where I have to again load that pattern, that 30 that 3603 or that value 3036; that value 3036 has to be loaded into this the timer registers TL0 and TH0; so that has to be done.

So, we do it like this; that MOV TL0 comma hash 0DCH and this TH0; MOV TH0 comma hash 0BH. So, if you do that; so, this number 0BDC; this is actually that 3036 H; 3036 value and then I have to set the timer SETB TR0. So, that the next the timer is enabled; so that it will again be interrupting the system, when the timeout occurs.

So, this is the end of the interrupt service routine; now the main routine that I need to write. So, the main routine will be something like this; first this timer has to be programmed, the mode has to be set. So, you have to MOV it the TMOD register has to be programmed with the pattern 00000001 in binary; then TL and TH, they are to be having that this time values; the initial values MOV TL comma hash 0DC H; then MOV TH 0 comma hash 0BX.

Then I have to clear the TF 0 bit and then I have to enable the interrupt for the timer. So, you can check that this interrupts setting will lead to the value 82 H for enabling this timer 0 interrupt, it will be 82 H. Then this will be SETB; this is SETB TR 0; this is say SETB TR 0; so, the timer will be enabled and then I can just wait here. So, this is we can wait for this; it said that now I have to read the mode button is pressed or not; so, that has to be checked.

So, jump on bit P 1.0 comma L 2; then if the mode button is not pressed then this bit is 0. So, we should display the date part; so in that case the DPTR register should be loaded with the value 8000 and then I can just jump over to L 3; then in L 2; L 2, we came

because the mode button has been pressed fine. So, this DPTR has to be loaded with a different value; so MOV DPTR with the address of this date part that is 8003 H.

And then in L3; I have to call the display routine ACALL display and then SJMP L 4. So, this is the main routine; so, main routine what are we doing? First setting the timer mode, setting the time count, clearing the timer interrupt flag TF 0; then we are enabling the interrupt; the timer interrupt is enabled then we are starting the timer this much is done.

Now, we are; so this is done only once up to this setting is done only once. Now I am reading port once bit number 0 and if that bit is set; that is the mode button. So, if it is if the mode button is pressed then this bit is 1, so we are coming to L 2 and in the DPTR we are putting the address of the date and the display routine is called; so display routine will display the value; we have got 6 displace and there that hh; that dd-mm-yy those will be displayed. And then SJMP L 4; it will again go back there check whether the mode button is pressed or not.

So, if the mode button is not placed in that case it will display the time part. So, it will come to 8000 H; so 8000 H will go to DPTR and then this value; so it will be jumping to L 3 and there it will be displaying part and this is DPTR points to the time part; so, it will be displaying the time part properly.

So, this way I can do this thing; only one thing one small thing is missing. So, at this point I think we need to have another clearing of this TF 0 bit; otherwise it will be taken as another interrupt immediately. So, the clear TF 0; so this line should be there. So, in this way we can design a timer based routine that can be used for designing a digital clock.

(Refer Slide Time: 23:24)

Example

Assume that the oscillator connected to 8051 chip is operating at 12MHz. Write a program to generate 4KHz square wave on pin P1.4 using timer 0.

$(65536 - \frac{125}{1}) = 65411$
↓
C1FF H

$\frac{12}{12} = 1\text{MHz}$
Time period = $\frac{1}{4\text{kHz}} = 250 \mu\text{s}$
ON = ~~250~~ OFF = 125 μs

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

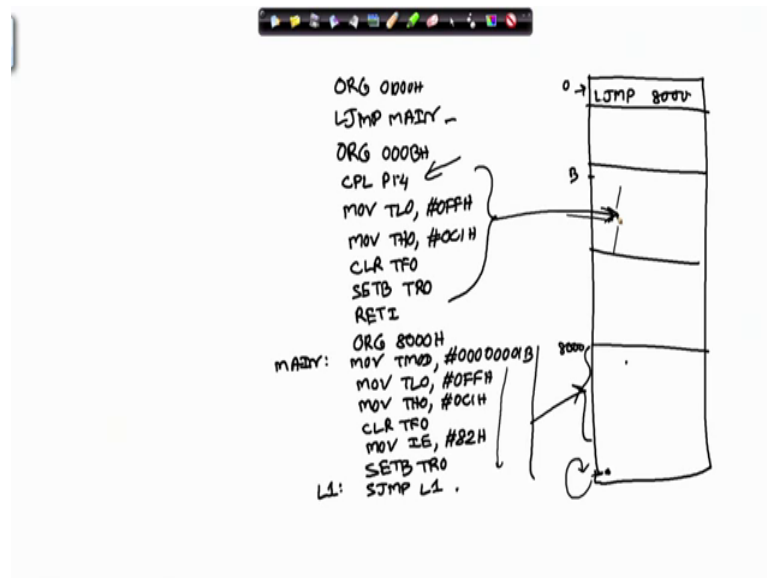
Next we look into another program; so, suppose we have got an oscillator connected to that 8051 that is operating at 12 megahertz. And we want to generate a 4 kilohertz square wave on the pin 1.4 using this timer 0. So how to do this? So, for this matter again; so, this timer clock that we have is that is a 12 by 12. So, it is 12 megahertz crystal divided by 12; so the timer clock is 1 megahertz, but the waveform that we generate for 4 kilo hertz.

So, the required time period for this clock signal; time period needed is 1 upon 4 kilohertz. So, that is about 250 micro second; so this is 250 microsecond that we want. So, if we assume that it is a 50 percent duty cycle square wave. So, on period is on period an off period; so, both of them are same and they are going to be equal to 125 microsecond.

So, what we need to do from that timer is that we need to have a delay of 125 microsecond. So, for getting that; so I have to see what is the value that should be loaded? So, the value to be loaded is equal to 65536 minus this 125 by 1; so, 125 microsecond and this is 1 megahertz; so I can divide it by that. So, this quantity; so it is basically 65411 in decimal; so, that converts to C1FF in the hexa-decimal notation.

So, this should be the value to be loaded and accordingly we can have this square wave generated. So, we will write that program; so, how to write it?.

(Refer Slide Time: 25:40)



So, we can again take help of the timer interrupt; ORG 0000 H; you can write then LJMP main; so, which will take this assembler at this program at the memory location 0, it will put the instruction LJMP main. Then the timer interrupt is that location 000B; so, ORG 000B H. So, that is the timer location and the timer routine is simple; so, it just complements the bit P 1.4. When the timer interrupt comes, the action that I need to take is just a complement the port bit 1.4.

And then the timer has to be restarted; so, for that purpose. So, I have to MOV this TL 0 comma hash 0 FFX and I have to MOV TH0 comma hash 0C1 H. And then if that interrupt comes, then we have to clear this TF 0 and we have to SETB TR 0; then RETI; so, this clear bit will be clearing the bit, so that it is not the interrupt has been serviced. So, it should be cleared; otherwise it will be taken as another interrupt and this SETB TR 0. So, it will be setting the timer so that the timer will start again so, that the interrupt can come.

Then maybe my main program is from location 8000 H; so, I put another ORG here. So, ORG 8000 H and the main program is starting from there; so here the thing that I need to do is initialize that TMOD register. So, I can say like MOV TMOD comma; based on the requirement hash 00000001B then MOV TLO comma hash 0FF H; then MOV TH 0 comma hash 0 C1 H and then clear the TF 0 bit; set the interrupt enable register for

getting the interrupts hash 82 H. So, this will enable the timer interrupt 0 and then I have to start the timer; so, that is by SETB TR 0.

And now the main program can wait in a loop because there is nothing more to do. So, it is SJMP L 1; so, it will be looping there. So, now what the main program is doing? It is setting the timers setting interrupt and all that and then it is looping in an infinite loop because sitting idle actually. And whenever the interrupt comes, the timer interrupt comes; so it will be coming to this location 000B. And if you look into this memory layout for this program that I have written; so, it is like this. So, at location 0 we have the instruction LJMP main and main is 8000. So, it has got this one LJMP main; so that is 8000.

And then; so, this is there then at 000B at location B; you have got this code; this interrupt service routine code; so this will be here. Then at 8000; from location 8000 you will have this piece of code, so this piece of code will be loaded here. Now how the system executes; the whole thing like when the when the processor is reset the program counter is loaded with 0000 H; so, it will come to this particular location; it will get this instruction LJMP 8000. So, it will jump to this location 8000; start executing these instructions. So, when it starts executing these instructions; so it is the timer is set and all those things, so that is done; then the program is looping at this point.

In between some time the timer interrupt will come; timer 0 will overflow, timer interrupt will come. So, and the processor; the hardware will automatically put the program counter to come to this 000B; when the interrupt occurs. And then it will execute this interrupt service routine and after some after executing this interrupt service routine, it will again come back to this endpoint; where it was SJMP L 1. So, it will be coming back to this point; it will be executing there, it will be looping there continually; it will be executing there.

So, this way again when the interrupt comes; again this program will be invoked and it will be this interrupt service routine will be invoked and it will go like this. So, in this way we can use these interrupt service routines for doing many important jobs; so which are to be done irrespective of the main functionality that we have in the system.

So, that is for timer 0; so we have also seen for the serial transmission; we have also seen for these external interrupts. So, that way interrupt programming is very important if you

are trying to design an embedded system, where you have got a number of devices or number of activities to be carried on at some specific intervals of time.