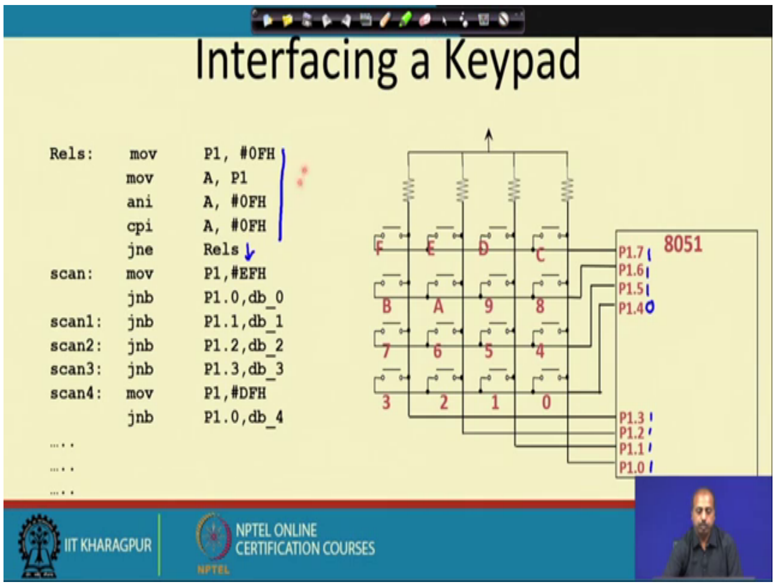
Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 54
Interfacing (Contd.)

(Refer Slide Time: 00:17)



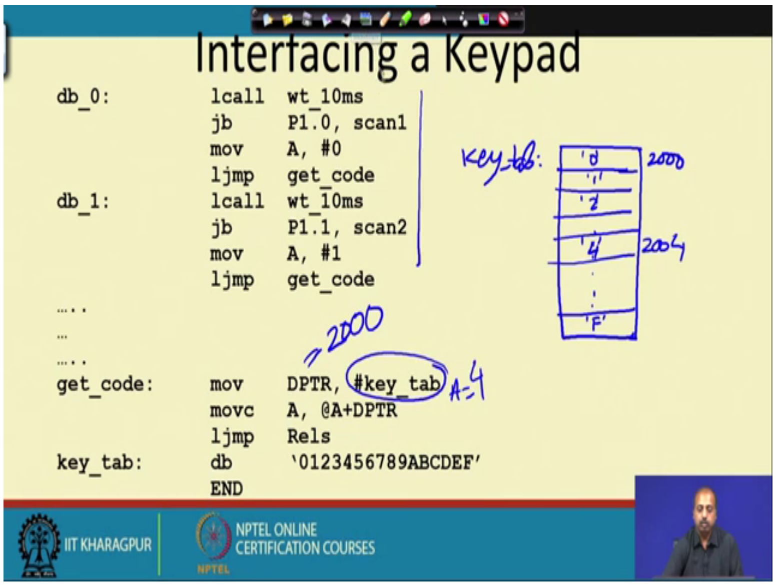
Interfacing a Keypad

```
Rels:   mov     P1, #0FH
        mov     A, P1
        ani     A, #0FH
        cpi     A, #0FH
        jne     Rels
scan:   mov     P1,#EFH
        jnb     P1.0,db_0
scan1:  jnb     P1.1,db_1
scan2:  jnb     P1.2,db_2
scan3:  jnb     P1.3,db_3
scan4:  mov     P1,#DFH
        jnb     P1.0,db_4
.....
.....
.....
```

So, for the keyboard interface we have seen that for the initially we have to check whether all keys are released or not and that is done by the first part of the code, in this region where I am doing that thing. So, it is we are checking that whether all keys are released or not. So, once we have seen that all keys are released. So, this jne has failed. So, you have come to the next point. Now I have to make one of the rows as 0 and check whether any key has been placed on that line.

So, we do this by putting the pattern EFH on to the port P 1. So, EFH means these all these bits are 1. So, they are f and this is also so these bits are all 1 and this bit is 0 so, this bit being 0. So, well be getting the so accordingly so any key pressed on this row if any key is placed on this row. So, accordingly that corresponding bit here will become 0. So, after outputting this so, we are we are getting this and then we are checking the bit P 1.0. So, we are checking this particular bit. So, if this key has been if P 1.0 this bit appear this bit becomes 0, now after doing this output. So, then it is that the key placed is bit 0. So, then we go to this db 0. So, db 0 will be so, db 0 is here.

(Refer Slide Time: 01:48)



Interfacing a Keypad

```
db_0:       lcall   wt_10ms
            jb      P1.0, scan1
            mov     A, #0
            ljmp    get_code
db_1:       lcall   wt_10ms
            jb      P1.1, scan2
            mov     A, #1
            ljmp    get_code
.....
...
.....
get_code:   mov     DPTR, #key_tab
            movc    A, @A+DPTR
            ljmp    Rels
key_tab:    db      '0123456789ABCDEF'
            END
```

So, now we need to check that whether it was an error or not. So, that check is done at this point. So, we call for a delay of 10 millisecond we wait for 10 millisecond and after 10 millisecond, we again check whether bit 1.0. So, if the bit becomes 1 by this time; that means, it was a noise ok. So, that was a flicker so, it is not a valid bit. So, in that case we will be going back to scan 1. So, scan 1 is here so this points. So, again it will be going back to this point and again we will check whether this bit is pressed or not.

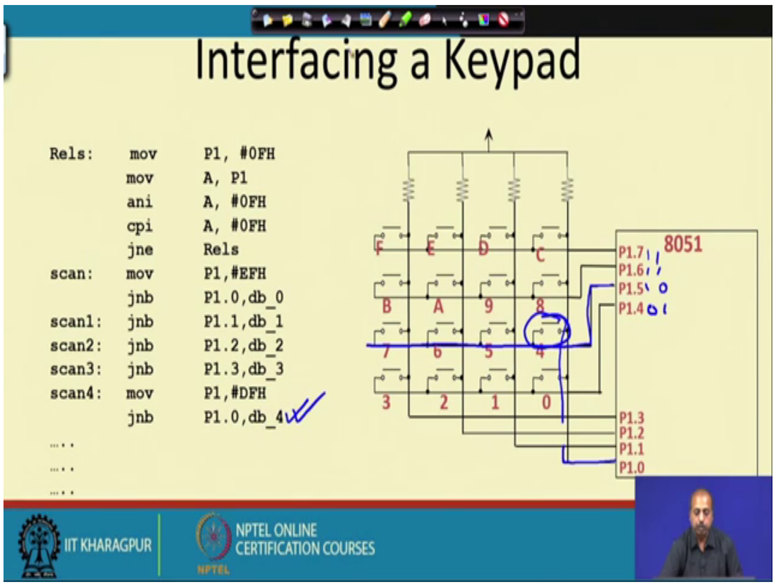
So, in this way it will be going back to the previous point and, then we will be moving so if the bit has been placed if the key has been placed, then this we are moving the value 0 to A and A register and then ljmp gate code. So, ljmp gate code here I will be getting the corresponding ASCII key code. So, in this key table is defined like this so, in individual bytes. So, key table is a memory location from where I will be storing the db value 0 to F. So, those key values are stored and this A value A register value will be used to index into this table.

So, what is done from that DPTR register we load the value of key table, then movc A comma at the rate a plus dp DPTR. So, if the 0 key has been placed, then it will be coming to the first location of this key table. So, you will get the character 0. On the other end if the one key was placed. So, if we just go back, if this is key this key is not placed, then we are checking the next key the scan 1. So, we are checking for whether the next key has been placed.

So, 1 has been placed or not so if 1 is pressed then it will be so, we have to say that we have to put the code of one into the accumulator. So, that is done here in db 1 you see that we wait for 10 milliseconds, then again wait for they, then again read the value of 1.1. So, if 1.1 is 0 so, if this is sorry if this is 0; that means, the key 1 has been pressed if this bit is 1 that means, the key is not pressed. So, it was in noise so we go to the scan 2. So, scan 2 is actually taking us to the next point. So, scan 2 is here I am checking for the next column ok. So, this way this routine continues ok.

So, now we have got this a register will contain the value of the key that has been placed. So, after all these are done then we will have so then we will have to go back to this point again. So, where will be so will be outputting DFH. So, we have previously we are outputting EFX, where this e 1 this bit was this bit was 0 and rest of the bits were 1. Now I am putting df df means. So, this bit will be 0.

(Refer Slide Time: 04:46)



Interfacing a Keypad

```
Rels:    mov    P1, #0FH
         mov    A, P1
         ani    A, #0FH
         cpi    A, #0FH
         jne    Rels
scan:    mov    P1,#EFH
         jnb    P1.0,db_0
scan1:   jnb    P1.1,db_1
scan2:   jnb    P1.2,db_2
scan3:   jnb    P1.3,db_3
scan4:   mov    P1,#DFH
         jnb    P1.0,db_4
.....
.....
.....
```

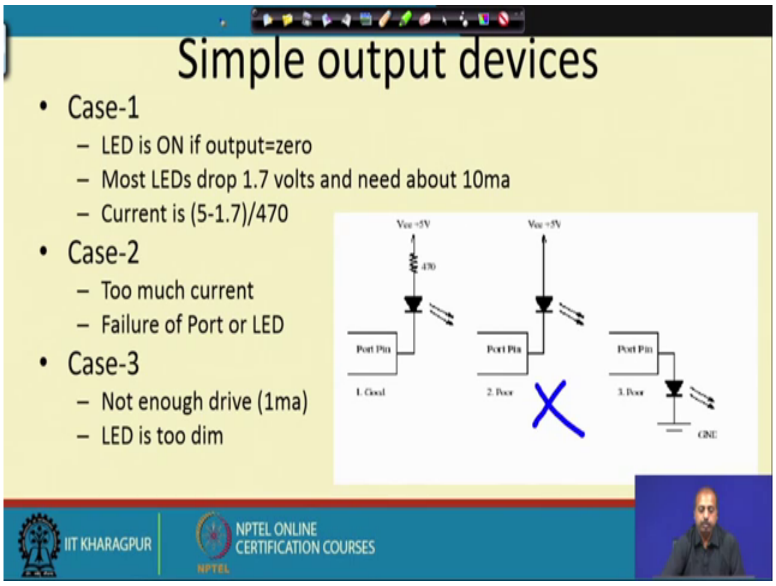
And these are all 1s so now, this 1.5. So, this particular row I am checking and then again the same thing the similar code will repeat, but the numbers will be different. Now if you find that say db 0. So, when you are doing this check 1.0. So, when you are checking this pin. So, if this bit is 0 means this key has been pressed this key has been placed. So, we can jump to the db 4 routine.

So, then we jump to that db 4. So, this way we will have a similar such db 4 routine which is not shown here for the sake of brevity. Now so, this in the key table we are storing all the key values. So, this key table you see that it is something like this in the key table it is a part of the memory where, I have stored all the key values. So, in the first location of this key table we have got the hex key code of 0 the character 0, then the hex key code of character 1 then the hex key code of character 2, and then finally we have got since up to f is there in the hex key code in the hex key board.

So, we will store up to the character f and then so, this is the key table. So, this is the key table. So, this is so success so it will have some address say these addresses say 2000. So, from 2000 onwards these values are stored. So, when I am first in this gate code routine we first getting this hash key tab into DPTR. So, DPTR becomes equal to 2000 and then we are adding with DPTR the value of A. And if you look into this part of the routine you can understand that in the A register we are holding the key index that has been placed. So, that is they are in this a register it is added we DPTR. So, if you have pressed say key number 4 then A is equal to 4. So, DPTR plus a will take us 2 thousand plus 4 it will come to the location 2004 and in 2004 I will have the character 4 there. So, character 4 will be stored there. So, ultimately a will get that character 4.

Then it will again jump to so, we have got the character 4 here. So, we can do some processing with that character which is not shown here explicitly, but as far as keyboard routine is concerned. So, it will go back and it will again check whether all keys are released or not. So, it will do that. So, like here it is t is jumping back to this Rels so, Rels is here. So, it is jumping back to this point to see whether all keys are released or not. So, this way we can very easily interface a keyboard here the hexadecimal keyboard with any microprocessor or microcontroller based system.

(Refer Slide Time: 07:42)



Simple output devices

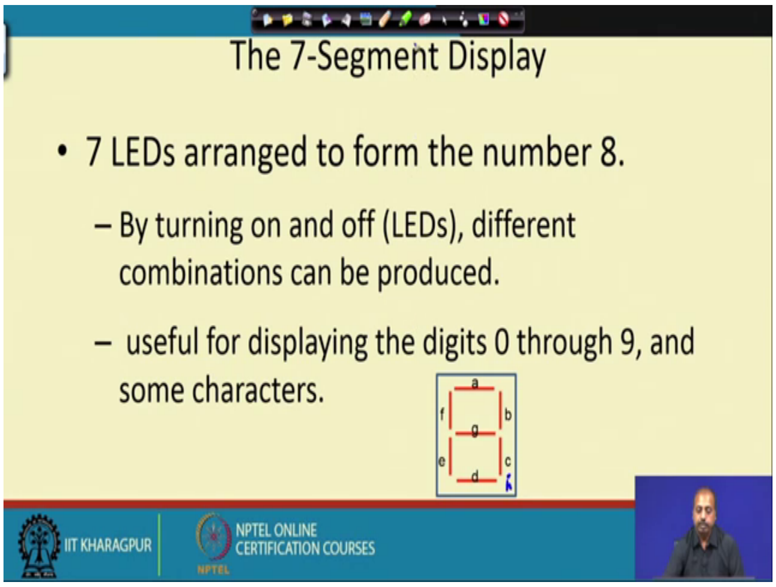
- Case-1
  - LED is ON if output=zero
  - Most LEDs drop 1.7 volts and need about 10ma
  - Current is (5-1.7)/470
- Case-2
  - Too much current
  - Failure of Port or LED
- Case-3
  - Not enough drive (1ma)
  - LED is too dim

Next we look into output device. So, we have seen the keyboard as an input device. So, next we look into output device. So, output device we can connect some LED which is the most common output device. So, this may be one type of connection. So, here we have put a current limiting resistance here, then that is led. So, if you to glow the LED to glow the LED you should put A 0 you should get A 0 at this point. So, if we put A 0 here then there is a current flowing path and the LED will glow. So, that is one type of connection. Second type of connection is like this where this limiting resistance is missing, but if you do this then there it is dangerous because a high amount of current will flow through this line, when this be this bit is 0 on port bit is 0.

So, this will you will get a high range of high value of current flowing. So, LED may burn and the port pin whether the port of the microcontroller or microprocessor may also get damaged. So, this is never recommended ok. So, this is not recommended. And the third connection where I have got this port pin giving a value of 1 and that is actually making this LED on ok. So, this connection this is this may be fine, but this port pin they normally drive very low current. So, they do not have high current. So, the LED may not be glowing very brightly ok.

So, we have got 3 different cases in case one LED is on if output is 0, most of the LEDs they will drop 1.7 volts. So, you can find out so for glow for a LED to glow completely you need about 10 milliampere of current. So, you can find out 5 minus 1.7 by 470. So, that is a nearly 10 milliampere current you will get here, if you put a 470 ohm resistance at this point case 2 is not advisable case 3 normally we do not get more than one milliampere current this port pins they do not drive high load. So, the current values are not that high. So, the LED will be dim so, the visibility may be a problem. So, thus the first connection is better ok.

(Refer Slide Time: 09:54)



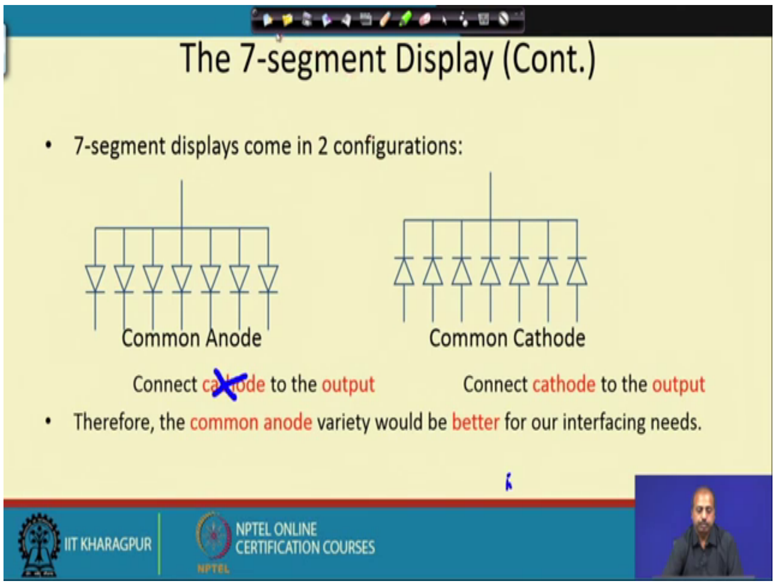
The 7-Segment Display

- 7 LEDs arranged to form the number 8.
  - By turning on and off (LEDs), different combinations can be produced.
  - useful for displaying the digits 0 through 9, and some characters.

The next display that we will have is a 7 segment display. So, this is also very common, but like say digital watches and all that of course, many a time now we have got LCD displays, but LED displays are less costly and that way we have got 7 LEDs arranged to form the number 8. So, this is the structure. So, we have got this LEDs a b c d e f g and, sometimes we have got another LED.

So, instead of 7 segments it will be called 8 segments so, where we have got another LED here so, another LED at this point. So, which is the dot so, you have got a dot here, so that is often called the bit h, but so that can display the more variety of digits and letters some of the characters can be displaced displayed not all. So, what can we do if we want to display some pattern we can turn on the corresponding top portions accordingly we can get a digit or a character displayed.

(Refer Slide Time: 11:00)



The 7-segment Display (Cont.)

- 7-segment displays come in 2 configurations:

Connect ~~cathode~~ to the output

Connect cathode to the output

- Therefore, the common anode variety would be better for our interfacing needs.

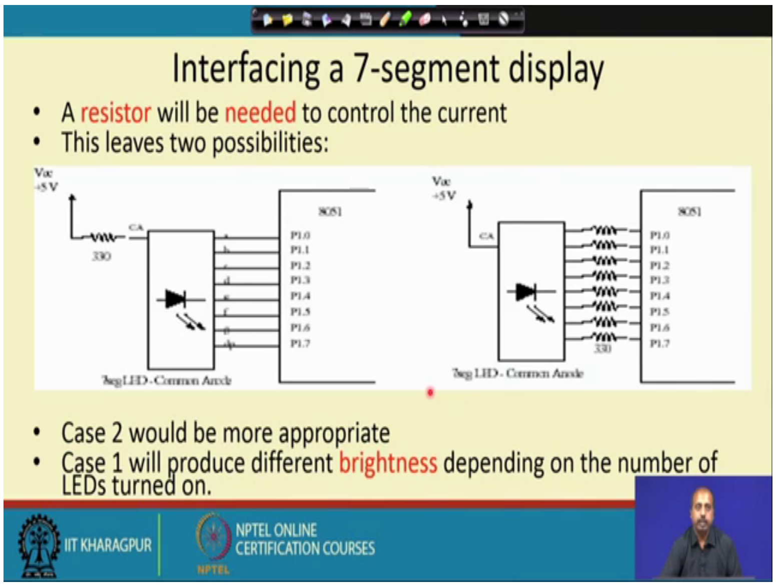
So, there are 2 types of 7 segment displays one is called common anode another is called common cathode. So, this common anode means all the anodes they are tied together all the 7 LEDs that anodes are tied together and, they are driven by a single line and, then all these cathode. So, if you want to glow over a particular segment, you have to give a high value here and you have to ground the corresponding LED corresponding diode ok.

So, for example, if you want to glow say this LED only. So, you have to give you one at this point and you have to give 0 here whereas, for the rest of the lines here on this side. So, you will have to you will put a 1. So, that there these LEDs do not glow on the other hand this common cathode configuration. So, here this side the cathode part is common ok. So, there this can be connected to some ground pin to some ground line and, wherever whichever LED you want to turn on. So, you have to put a 1 there so that way. So, that light will be 1.

So, these are the 2 different configurations common anode and common cathode in case of common anode the cathode lines are connected to the output and in case of common cathode anode lines a common cathode, common cathode is a common anode is connected to the output. So, here also the sorry here also the common a common cathode is connected to the output here, the contact cathode will be connected to the output and here, the contact anode will be connected to that called in this case contact anodes will be connected to the output. So, this is wrong so this is wrong this should be contact anode.

So, this con this contact anodes will be connected and this is in this case contact cathodes will be connected, contact and the cathodes will be connected so this 1. So, this variety is the better for interfacing purpose this common anode variety is better because here, you can put a 1 and here you can put a put the value 0 whereas, in this case you have to put. So, many resistances for doing that for interfacing purpose this one will be better.

(Refer Slide Time: 13:26)



Interfacing a 7-segment display

- A resistor will be needed to control the current
- This leaves two possibilities:

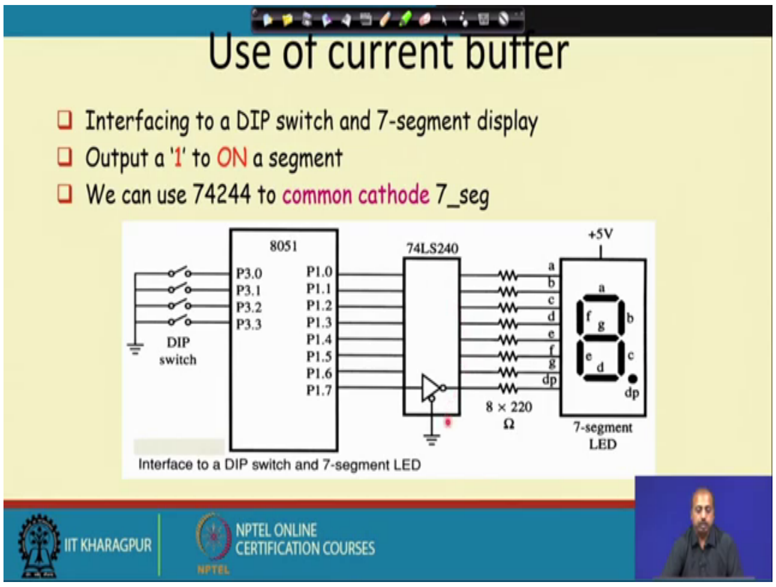
- Case 2 would be more appropriate
- Case 1 will produce different brightness depending on the number of LEDs turned on.

So, a resistor will be needed to control the current. So, this is in common anode configuration. So, ill connect this resistance a single resistance will be sufficient and, they will be giving current to all the 7 segments of the despair this is in the 8051 port, I will put the corresponding bit to 0. So, that the corresponding LED will glow, on the on the other hand in the common anode configuration we will see that well give the supply volt and now individual LEDs so they are having their own registers.

So, in this case the current that will come. So, the current that will be there so, that will be limited by this resistance and that current will be divided among the LEDs that will be on. So, all for all the bits that are turned on the current will get divided between them. So, as a result this the all the LEDs may not glow very brightly because, the current getting divided so, the brightness may be a problem, when if only one LED is on others are off that LED will be glowing very with a with a good brightness whereas, if say all the LEDs are turned on then all of them will be glowing, but the output may be dim.

On the other hand in this case we do not have that problem because, these LEDs currents are controlled separately we have got the separate resistances. So, the currents are controlled separately. So, we have got this brightness will be better in this common anode configuration. So, that is why we said that the common anode configuration is better for connecting the LEDs.

(Refer Slide Time: 15:07)



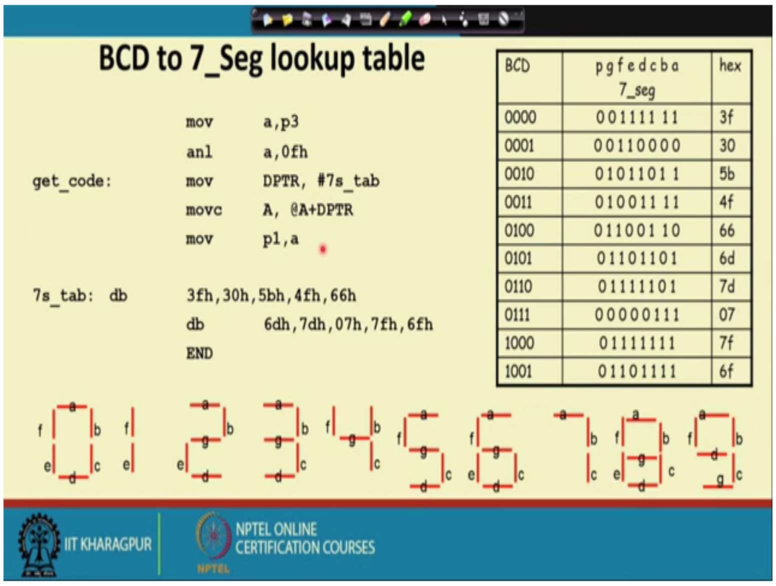
Use of current buffer

- Interfacing to a DIP switch and 7-segment display
- Output a '1' to ON a segment
- We can use 74244 to common cathode 7_seg

So, how do we do this thing, how do we do this connections interfacing 1 dip switch and 1 7 segment display to the LEDs or to the 8051 so, may be with the port 3 we have connected 4 dip switches and with this and their so, 3.0, we have got this active pullup resistance. So, that way the pullup resistances are there. So, you if you keep it open then the bit port bit is automatically 1. So, we can use that philosophy and we can if we want to put a 0. So, I have to close the switch accordingly that bit will become 0 and this particular display.

So, this 1 point so this P 1.0201 0.7 these bits. So, they can drive this 74240 so that chip. So, this is and this is a driver for this 7 segment display and you can put the value here accordingly, this the corresponding segments will be glowing this a b c d e f. So, this actually outputs is this, a b c d e f g and this dot point dp. So, this driver can be used for that purpose.

(Refer Slide Time: 16:22)



BCD to 7_Seg lookup table

```
get_code:       mov     a,p3
                anl     a,0fh
                mov     DPTR, #7s_tab
                movc    A, @A+DPTR
                mov     p1,a

7s_tab:  db     3fh,30h,5bh,4fh,66h
                db      6dh,7dh,07h,7fh,6fh
                END
```

BCD	p g f e d c b a 7_seg	hex
0000	00111111	3f
0001	00110000	30
0010	01011011	5b
0011	01001111	4f
0100	01100110	66
0101	01101101	6d
0110	01111101	7d
0111	00000111	07
1000	01111111	7f
1001	01101111	6f

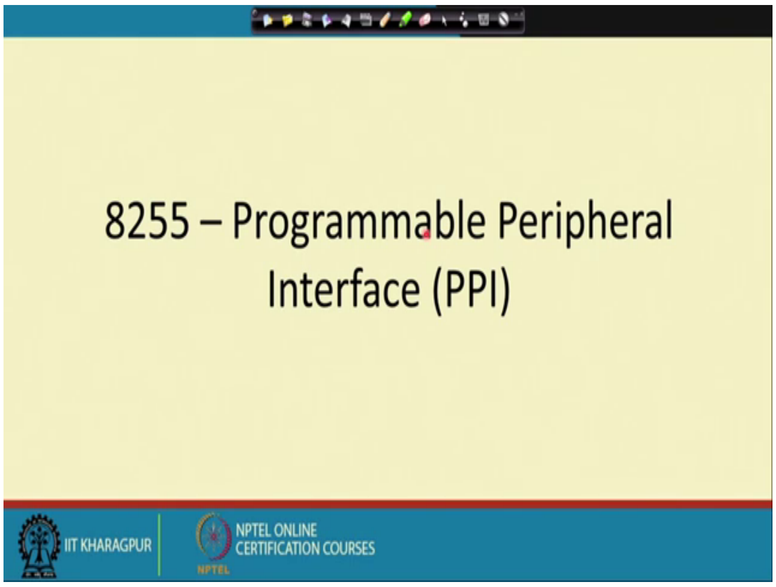
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we can we can have a lookup table like this, this if you want to display that digit 0, then the segments that should glow are a b c d e and f. So, this a b c d these will be on g and h g and p. So, those are off so, accordingly the hex code is 3 f. Similarly if it is 1, then only the segments f and e they will be turned on rest are 0. So, accordingly the code is 3 0. So, this way you can find out the hexadecimal codes for all the characters that you want to display on your system fine.

So, this way the program is now very simple. So, from the port P 3 we get the value into a register and, then we and logical a with 0fh. So, that way we can get the content from port 3 like here. So, we are we can we can just get these 4 port bits into the a register and, then since this 0 1 2 3 is only there. So, we have to mask out the higher order bits. So, that is done here by doing this 0 fh this higher order bits will be masked off, then this move DPTR comma hash 7 s table so 7 s table will have the 7 segment things like all these numbers 3 f 3 0 5 b so, whatever patterns are there. So, they are stored from this 7 s table onwards. So, that value will be move to so, DPTR will get the value of the key that is placed; so 0 1 2 3 so though those keys will be there.

So, that will be added to this DPTR and here we have got this the number that I want to display, that should be entered through port P 3 the 1 single digit. So, though it is up the number so, you can enter 0 to f that way only. So, the number will be entered on port P 3 as a binary decimal binary coded decimal digit and, then the well get that corresponding code from the 7 s table and that will be moved to the port p p 1 and on port p 1 you can get that the corresponding display. So, this way you can interface 7 segment display with the this 8051 chip.

(Refer Slide Time: 18:42)



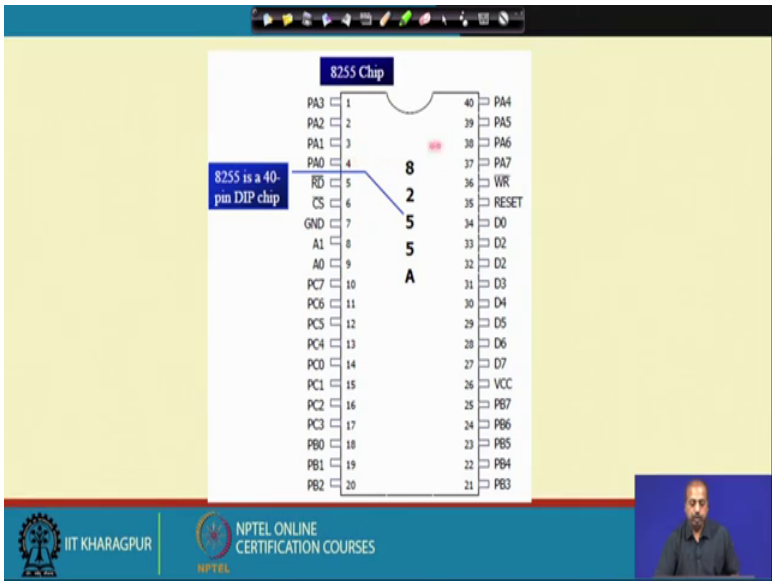
8255 – Programmable Peripheral Interface (PPI)

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Next we look into another very well-known interface chip, which is known as programmable peripheral interface or PPI chip, which is 8255. So, this is needed because your 8051, if you see carefully then 8051 has got 4 ports P 0 P 1 P 2 P 3 out of that only port P 1 is totally free P 0 and P 2. So, they are used for this connecting external memory. So, this address and data base lines are there and for port P 3. So, that is multiplexed with many other functions. So, that creates difficulty so, if you in your application so, if you need to connect both external memory and some peripheral device, then we do not have much option left.

Because we do not have enough port left for that purpose. So, if you connect one 8255 chip with the system. So, it makes you get a number of ports available and, they are very easily programmable. And also if you look into other processors like 8085 and all that is why we do not have any IO port directly. So, you can of course, use this in out instructions to read the values directly from the data base lines, but the data base lines are hanging in that case ok. So, that is another problem. So, we have to do something so, that we can get some buffered input output. So, 8255 will ensure that type of operation. So, this is the pin diagram of 8255. So, it is a 40 pin dual inline chip.

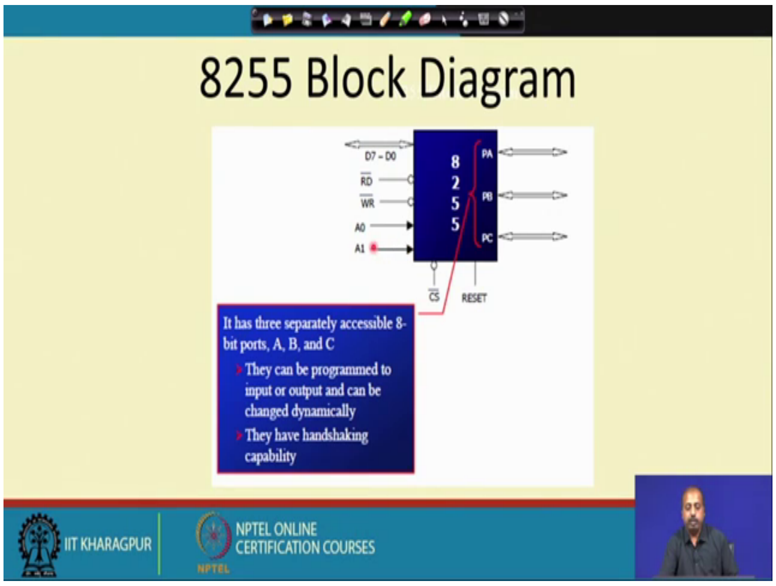
(Refer Slide Time: 20:17)



So, you see if you look into this lab bits that we have pins that we have we have got port a pins port A 0, A 1, A 2, A 3, A 4, A 5, A 6, A 7. So, port A is an 8 bit port, then we have got another port PB A port B. So, PB 0 to PB 7 then, port C PC 0 to PC 0 1 2 3 4 5 6 7. So, that is again another port, and then so these are the 3 ports that we have in 8255 and for getting the values into 8255. So, there are lines D 0 to D 7 which is which can be connected to the data base of the processor, for selecting the chip you have to make this CS bar pin low.

So, you can have some appropriate decoding logic by which the CS bar pin will be activated. So, you will get a this 8255 will get selected by that and, if you and also there are some internal registers and this port. So, for selecting internal registers and ports so, you can select this A 0 and A 1 bits accordingly. So, they are normally connected to the A 0 and a one bits of the CPU address bars and from there we get the value we control these individual registers and, then this read bar write bar lines are there for doing read or write operation for the ports.

(Refer Slide Time: 21:40)



8255 Block Diagram

It has three separately accessible 8-bit ports, A, B, and C

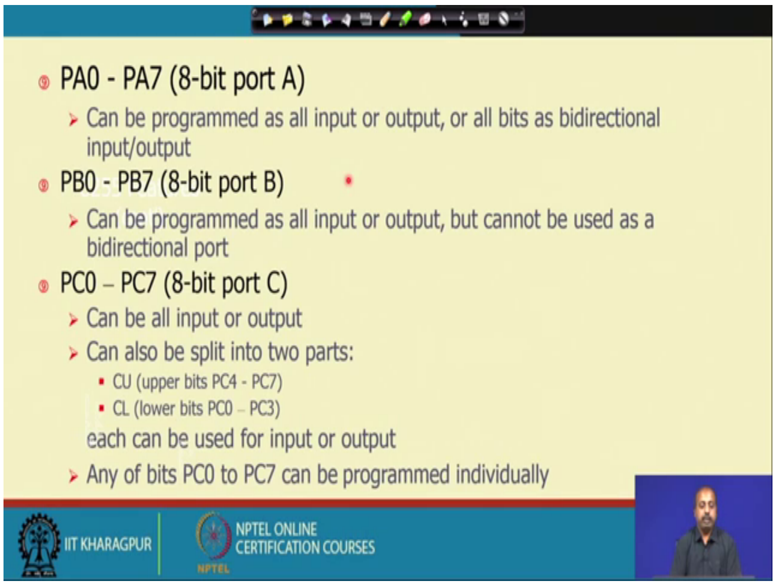
- They can be programmed to input or output and can be changed dynamically
- They have handshaking capability

So, at a block diagram level so you can say that 8255 it has got 3 8 bit ports A B and C, they can be programmed as to input or output or can be changed dynamically and ok, it can the structure. So, you can just change some control word setting of 8255 by which these ports will behave in different fashion.

So, you can program it as input port you can program it as output port and, you can also program it as bidirectional ports in some cases some limited cases and also you can program some of the port bits to be bit addressable by individual bit setting can be done, they also have handshaking capability. So, if you are connecting a number of devices with your say microcontroller or microcontroller of microprocessor, you want to connect some devices. So, the data transfer will take place via handshaking, then this can be done ok. So, this 8255 handshake mode can be utilized for that.

Other important pin lines that we have D 0 to D 7 read bar right, but A 0 A 1 CS bar and reset. So, we will see their utility.

(Refer Slide Time: 22:50)



- PA0 - PA7 (8-bit port A)
  - Can be programmed as all input or output, or all bits as bidirectional input/output
- PB0 - PB7 (8-bit port B)
  - Can be programmed as all input or output, but cannot be used as a bidirectional port
- PC0 – PC7 (8-bit port C)
  - Can be all input or output
  - Can also be split into two parts:
    - CU (upper bits PC4 - PC7)
    - CL (lower bits PC0 – PC3)

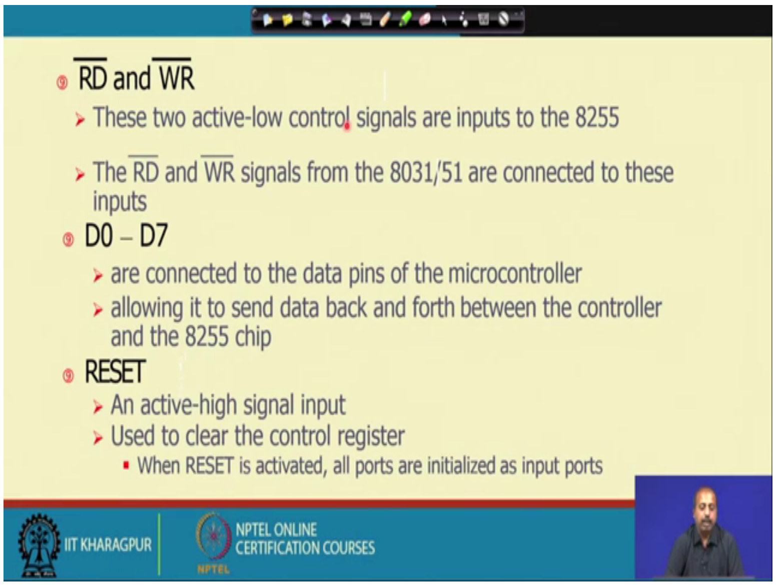
    each can be used for input or output
  - Any of bits PC0 to PC7 can be programmed individually

So, this port P PA 0 to PA 7 so, these are 8 bit lines for port A. So, they can be programmed as all input or output or all bits as bidirectional input output. So, this is so you cannot do a bit wise programming for port A. So, this entire 8 bit has to be programmed either in input mode, or in output mode or in bidirectional mode.

Then this PB 0 through 7 so, they can be programmed as input or output again the same thing, but it cannot be used as a bidirectional port, only port a can be used as bidirectional port B cannot be done that way and we have got port C. So, port C is PC 0 to PC 7 again this is an 8 bit port. So, again all can be done as input or output. So, it again it cannot be put in by directional mode, but this port c can be splitted into 2 port. So, we can say port C upper bits and port C lower bits CU and CL. So, upper bits are PC 4 to 7 and lower bits are PC 0 to 3.

So, each can be used separately for input or output purpose and, these bits of port C they can be programmed individually. So, this has got that bit addressability feature. So, if you are using say 8085 type of microprocessor where you do not have bit addressability of ports. So, you can connect one 8255 chip with that and, then you get some bit addressable locations for input output.

(Refer Slide Time: 24:23)



- $\overline{RD}$ and $\overline{WR}$
  - These two active-low control signals are inputs to the 8255
  - The $\overline{RD}$ and $\overline{WR}$ signals from the 8031/51 are connected to these inputs
- D0 – D7
  - are connected to the data pins of the microcontroller
  - allowing it to send data back and forth between the controller and the 8255 chip
- RESET
  - An active-high signal input
  - Used to clear the control register
    - When RESET is activated, all ports are initialized as input ports

Then we have got this read bar and write bar lines. So, these are 2 active low signals that are the used as inputs to 8255. So, read bar and write bar so, they can come from they may come from 8031 8051 and they may be connected to those inputs, then we have D 0 to D 7. So, they are actually D 0 to D 7 they are data pins that we can connect to the microcontroller.

So, we can send data packet to the 8255, or you can get the data packet back from them like, you may want to control some of the port configuration that way you want to output something to the 8255, or if a particular port is configured as output. So, you want to send data to that port. So, that way you can send the data from microcontroll, or microprocessor through this D 0 to D 7 to that port, or if a port is configured as input port. So, you can transfer the content from that input port to the microcontroller by that.

Then we have got this reset. So, reset is these are active low active high signal. So, this is used to control the register. So, clear the control register. So, all ports will get initialized as input port. So, this is this is the default configuration when you reset the 8255 chip all ports are configured as output ports. So, if you sorry as input port so, if you want to operate 8255 in input mode only. So, you do not have to do any control word setting. So, you can start directly, but if you want that this is this has to be some of the port bits should be output, then we have to put some control word in those things.

So, this way we can use 8255 for various operations, ok. We will see some examples by which we can do this interfacing.