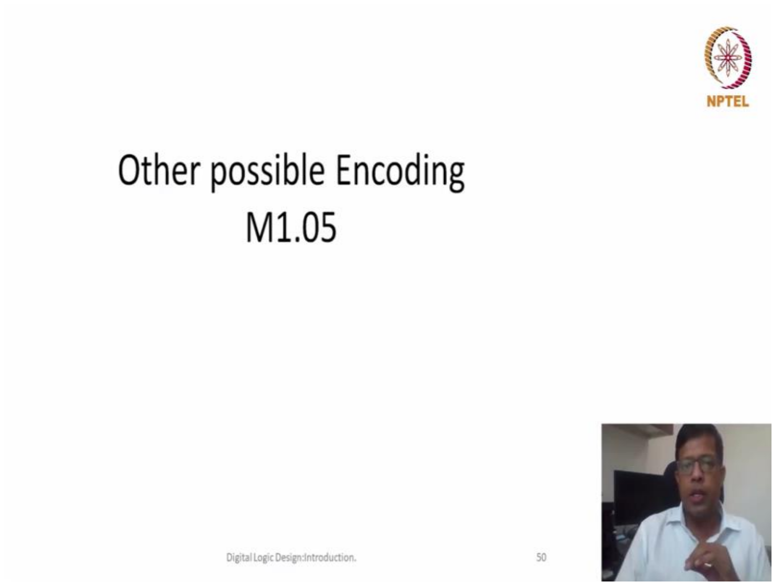
Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Lecture 7
Other Number System

(Refer Slide Time: 00:16)

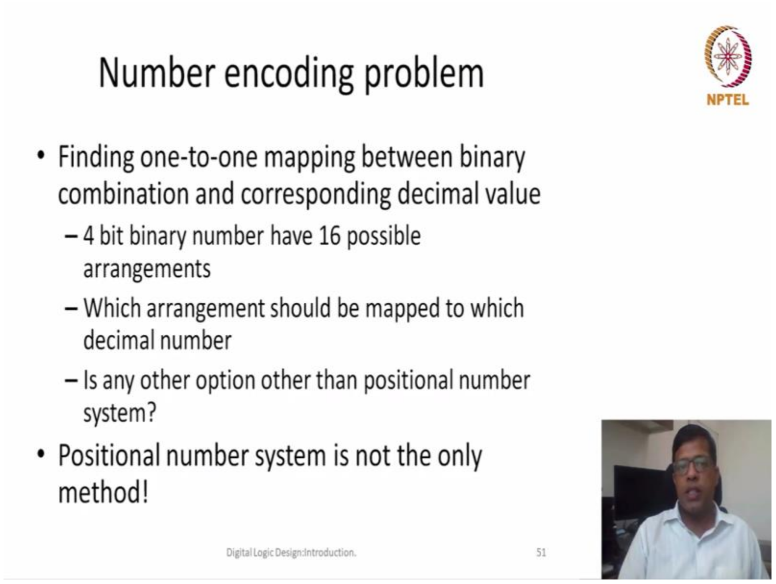


Other possible Encoding
M1.05

Digital Logic Design:Introduction. 50

Hello everyone. Today we discuss what are the other possibilities of Encoding Numbers using Binary.

(Refer Slide Time: 00:28)



Number encoding problem

- Finding one-to-one mapping between binary combination and corresponding decimal value
  - 4 bit binary number have 16 possible arrangements
  - Which arrangement should be mapped to which decimal number
  - Is any other option other than positional number system?
- Positional number system is not the only method!

Digital Logic Design:Introduction. 51

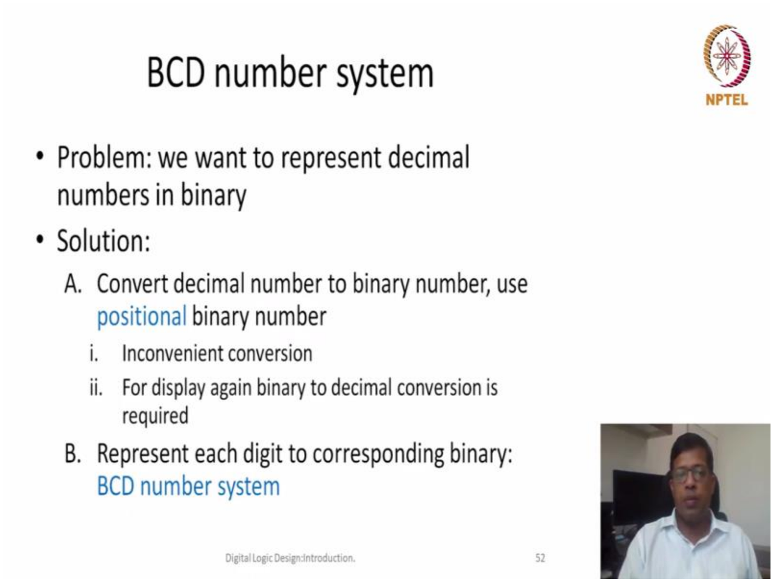
So, like let us reiterate the problem when we were doing the initial lectures. So, we have seen that in a binary number, let us say it is a 4-bit binary number, it has 16 different possible arrangements. So, each of these arrangements of 4 bit could correspond to a number and during our initial discussions of Unsigned Binary numbers, Unsigned umbers and again Sign numbers like representing negative numbers using different ways.

We have observed that there are various possibilities. Now, we would like to raise this question again that is there any other possibility to represent signed, unsigned numbers using these let us say 4 bit binary numbers. So, which, which arrangement should be there? One thing is clear that it has to be 1 to 1 mapping that each of the 4 bit binary arrangements should map to 1, 1 decimal number.

Which decimal number that is a good question? And that is the question we would try to answer here. And the numbers which we have studied so, far, the way the representation we have created so far, they are all called Positional Number System, because each symbol at a particular position has a certain weights to it. So and that was a standard weighted positional system and now, can there be merit or advantage in having some other system some other way of representing same binary numbers like in binary numbers can can we have when we rephrase it in another word.

So, basically, in the binary is there any other mechanism or any other mapping to map these numbers these binary numbers to any other two decimal numbers? Or should we conclude that the positional number system which we have studied so far is the only method is the best method of representing decimal numbers or representing numbers using binary.

(Refer Slide Time: 02:55)



BCD number system

- Problem: we want to represent decimal numbers in binary
- Solution:
  A. Convert decimal number to binary number, use positional binary number
     i. Inconvenient conversion
     ii. For display again binary to decimal conversion is required
  B. Represent each digit to corresponding binary: BCD number system

Digital Logic Design:Introduction. 52

So, let us pose 1 question that why are we representing decimal numbers in binary in a particular way. So, the problem is we would like to represent any kind of a number, which we understand that this is a number so, let us say we want to represent 10 20 or 1000 in binary. So, the method so far we have learned that let us convert this decimal number into a positional binary number by dividing it by two finding the remainder again dividing by two again find a remainder.

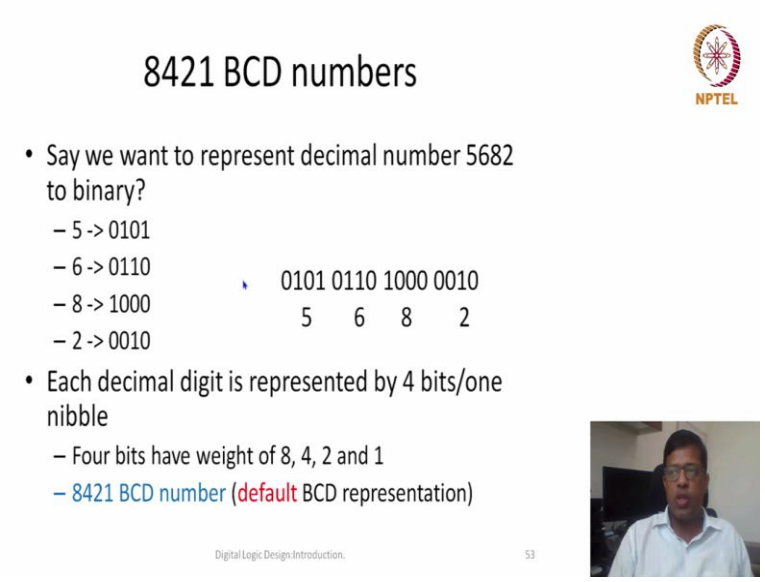
So, this whole process of converting a decimal number into binary number is a procedure in itself, which we have to learn and if we want to develop in hardware, we have to design in hardware such that each decimal number can be represented into binary numbers. So, this procedure once it is done, then the numbers are represented in positional binary system, then we can do addition, subtraction and everything.

But again, when we would like to represent it back to humans, because as a human, we understand only decimal numbers. Although in our first class, we have also seen their various different, different number systems like base 12, base 15, or Roman numbers which are there but let us assume for the sake of convenience, that decimal numbers are the standardized way of representing numbers now, and we as a human our eyes are more accustomed to look at decimal numbers to understand these decimal numbers to see them in magnitude to see them in quantity and comparing them doing arithmetic operations.

So, whenever computer would like to see show us back the numbers again has to convert that binary number into decimal number and then show it back. So, is there or can there be any other method which is more convenient which can solve this particular problem that we would like to represent the numbers. See, the problem is we have to represent the numbers in binary, why we are using only positional number system.

So, let us explore some other ways. So, one convenient or intuitive method which seems to be there that we have in a in a decimal number each digit has 10 different symbols 0 1 2 3 4 5 6 7 8 9. So, why cannot we represent all of these 10 different symbols using a 4-bit binary number because to represent 10 different symbols, we would require 4 bits, can we use these 4 bits to represent this each of the digit and then we represent a whole number using a collection of these 4 digits. If we represent in this way, we will call these numbers as BCD or Binary Coded Decimal Number System.

(Refer Slide Time: 06:27)



8421 BCD numbers

- Say we want to represent decimal number 5682 to binary?
  - 5 -> 0101
  - 6 -> 0110
  - 8 -> 1000
  - 2 -> 0010

```
0101 0110 1000 0010
  5    6    8    2
```

- Each decimal digit is represented by 4 bits/one nibble
  - Four bits have weight of 8, 4, 2 and 1
  - 8421 BCD number (default BCD representation)

Digital Logic Design:Introduction. 53

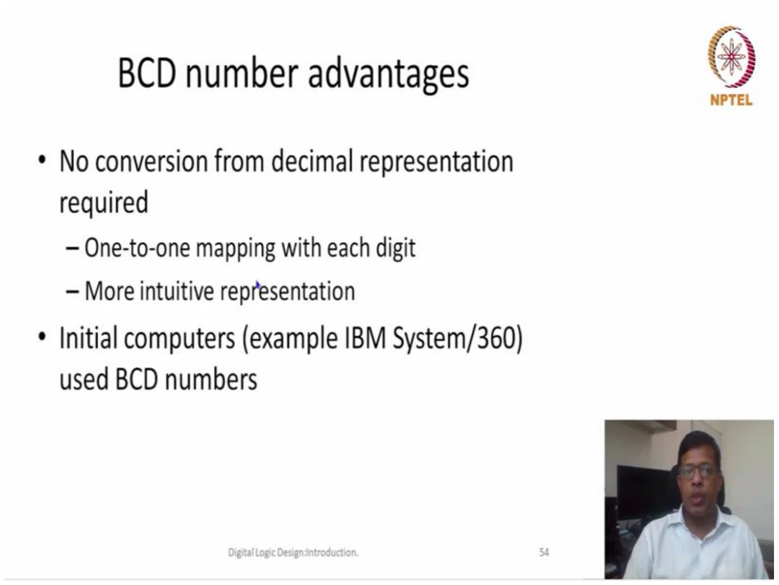
So, in a in the simplest form, let us say, let us say see this example, we would like to represent 5 6 8 2 as a binary number. So, we, we can represent like a 5 as 0101, 6 as 0110 8 as 1000 and 2 as 0010. So, these are the standard representation in our positional number system also, in positional number system, we represent all the symbols like this.

So, we simply concatenate all of these numbers, and we say that, to represent 4-digit decimal number, we would require a 16-bit binary number. So, that means, whatever number of digits are there in decimal number, we multiply it by 4. And that means number of bits are required to represent it in, in binary. This particular form of representing is, is simple, intuitive, and it would work quite well because we are able to see these numbers and the most important thing is that, the way we visualize numbers, we can easily visualize these numbers, we just need to break them into groups of 4 group of 4 we call nibbles. So, if we break them in nibbles, and then we can directly recognize it as a decimal number.

Now, this particular form like one more interesting thing here that here in using 4 bits, we can have 16 different combinations. So, which of the 16 combinations should be mapped to each of these 10 symbols we have in decimal? That is also a good question. So, let us say this particular representation we call 8421. Because each digit each binary digit will have will have a weight. So, for example, the first bit will have a weight of 1, the second bit has a weight of two and third bit has a weight of 4 and 4 bit it has a weight of 8.

So, because each digit represent, represent different weights, so that is why it is also called 48421 BCD numbers. So, this is this is the most convenient, most standard BCD representation. And each, each decimal number or each decimal digit is represented by, by a bit of 4.

(Refer Slide Time: 09:18)



BCD number advantages

- No conversion from decimal representation required
  - One-to-one mapping with each digit
  - More intuitive representation
- Initial computers (example IBM System/360) used BCD numbers

Digital Logic Design:Introduction. 54

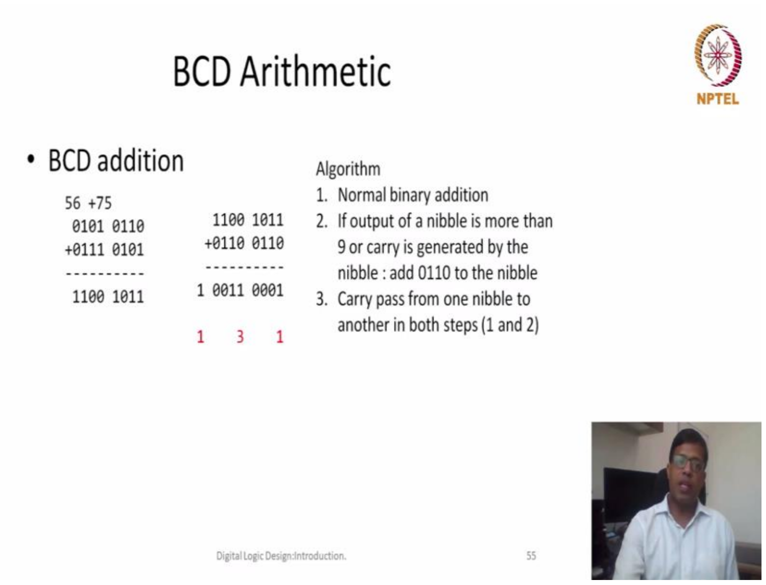
So this, this BCD representation is quite, quite interesting, useful because it is most intuitive, there is a 1 to 1 mapping. So, if we have any decimal number, we can quickly we need not to have kind of a conversion, conversion formula or that divided by two decimal things or some,

some algebraic procedure. But given a decimal number, we can directly represent them in binary. So, there is only 1 lookup table so each for each symbol, we have to replace it with 4 group of bits.

So far, it is so, so interesting or so far that when, when we build our initial computers, generation 1, generation two in those computers, we could not even think of the positional binary system what we have currently. So, at that time, because the only requirement of a computer was it has to have discrete discontinuous digits. And decimal digits are and also it has to be, it has to work with 0 and 1 binary system.

This BCD numbers were, were satisfying both of these criteria, that is my initial computers, we were using these BCD numbers and several BCD numbers were invented in that era. So, which had certain advantages and disadvantages. So, because of this intuitiveness wherever we just need to represent number. Now, what is the disadvantage that for representing it is good, it is sufficient. But when it comes to arithmetic, arithmetic means addition, multiplication, subtraction, these BCD numbers would be a slightly complicated because you have gap of the 6 numbers.

(Refer Slide Time: 11:10)



BCD Arithmetic

- BCD addition

```
56 +75
 0101 0110
+0111 0101
----------
 1100 1011
```

```
  1100 1011
 +0110 0110
 ----------
1 0011 0001

1   3    1
```

Algorithm
1. Normal binary addition
2. If output of a nibble is more than 9 or carry is generated by the nibble : add 0110 to the nibble
3. Carry pass from one nibble to another in both steps (1 and 2)

Digital Logic Design:Introduction. 55

So, let us say I want to do addition. Now, in addition, I am taking some example. So, let us say I want to add 56, and 75. Now, if I am trying to do addition using BCD. So, with this example, we will be able to understand that what kind of challenges we will see. So, when we want to do an addition of 5, 6 and 75, or any two particular numbers, first thing we have to do is we have to represent them using BCD.

So, 5 would be represented using 0101, and 6 would be represented using 0110, and 75 7 would be represented using 0110, 0111 and 5 would be represented using 0101. Now we want to add so we will align them. So basically, 5 would be just below 6 and 7 would be just below 5. And we can use the same binary addition rules which we have learned for our, for our addition to binary addition.

So, that means 1 plus 0 will become 1 again 1 plus 0 will become 1 1 plus 1 will become 0 and there is a carry of 1 and that carry will come here. Now similarly, 1 plus 1 is 0 carry of 1 1 plus 1 is 0 carry 1 1 plus 1 plus 1 is 1 and carry of 1 and carry comes here. Now, both of these numbers are cannot be represented cannot be is not a correct representation of our binary BCD numbers. In BCD the maximum or the largest digit could be 1001 that means 9 and so, to make this correction what we need to do is we need to add 6 to this, this output.

Why 6 because now the maximum number was 9 and if anything which is more than 9 if I add 6 then it will make it make it correct. So, for example, we will see using this so we will if the output is if output of my binary addition is more than 9 then I have to add 6 to that particular nibble. So, after adding 6 so for example the output was like after adding 2 BCD numbers here, the output 6 plus 5 will become 11.

Now 11 is more than 9. Now if I will add 6 to it, then it will become 1 and a carry will also be generated. Let us see here is so 1 plus 0 is 1 and 1 plus 1 is 0 carry of 1, carry of 1 plus 1 equal to 0 another carry generated carry plus 1 equal to 0 another carry generated and that carry would be added here.

So, rest of the addition 0 plus 1 is 1, 1 plus 1 is 0. Another cary is generated 1 plus 0 is 1 plus a carry of 1 plus this 1 equal to 0 and carry of 1 is generated. So, this gives us the correct results. This we can say is, is another 4 bit BCD number that would represent 1 and this is 3 this is 1. So, after this correction of adding 6 we will receive the correct addition result and when the 6 need to be added?

So, 6 need to be adding in two cases 1 if, if the output of addition of a nibble is, is more than 9 or if carry is generated, so, for example, there is a number a 9 here number 9 here 9 and 9 both are added it become 16 9 plus 9 2 are added become 18 and 18 is more than 16 So, that means a carry would be generated here. So, if there is a carry which is generated out of addition of two nibbles or the sum of two nibbles is more than 9 then 6 would be added.

And the other thing is whatever carry so, let us say a carry is generated in this the two editions, which we have to understand which, which we are doing in the first edition, if carry is generated then carry would be added by the carry, carry would be added to the second nibble and while doing the second edition of correction like where to each nibble 6 would be added if the sum is more than 9 then also if carry generated that carry would be propagated to next nibble.

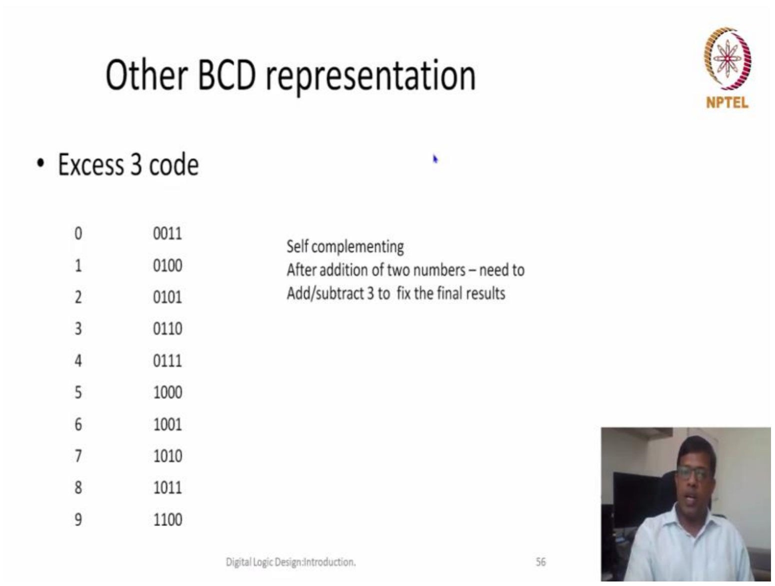
So, essentially, each BCD addition would require two additional steps 1 is a standard addition standard binary addition the second would be based on the condition, condition there are two conditions 1 if some is more than 9 or if carry is generated because of that particular nibble then 6 would be added in that nibble are a result of that nibble. So, in this way BCD addition would, would work perfectly fine.

So, after these two steps, we can do addition similarly, if we do we want to do subtraction, if I want to do subtraction then also the similar steps would be required there would be a correction that is needed. So, here we can see that for representing BCD numbers representing decimal numbers this BCD number system is good even for addition, the, the number of additional steps are, are minimal and thus can be used, but if you want to do a multiplication if you want to do division, the number of overhead or amount of overhead is going to be non significant, because every time we have to take into account essentially we are working with a base of two in with a system where basis 2 but actual basis 10.

So, that that will always be able to apply these correction factors. Whenever we are doing multiplication, division subtraction. But still, because earlier computers they were they were handling very small amount of computation. And at the time, the advancement in, in hardware was not that optimal. So that is why at the time, they were still willing to use BCD numbers in, in the computer's itself.

So, even this did that the example that IBM 360 which I gave it was it was a computer which it was a second-generation mainframe computer which whose life was more than 10 years. But only after integrated circuits were invented. And third generation microcomputers were invented, then we shifted completely to a positional number system. So, what during that era were BCD numbers were using computers, they also invented some other representation which can overcome these this, like additional steps or some shortcomings of these 8421 representations of BCD numbers. So, I will give one at least as an example here.

(Refer Slide Time: 19:10)



Other BCD representation

- Excess 3 code

0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Self complementing
After addition of two numbers – need to
Add/subtract 3 to fix the final results

Digital Logic Design:Introduction. 56

One, which is quite popular was Excess 3 Codes. In Excess 3 Codes this, I am covering this as a part of this lecture just to understand or just to see the beauty of representation and what kind of things we can do with the representation. So here, I am like, whatever, standard BCD or 8421 BCD was giving plus 3 was added. So, 0 was represented using 0011 and 1 was represented using 0100. And so on for the rest of the elements.

So, we can again, again, recall this particular thing that using 4 bits, we can have 16 different representation what out of those 16 we need to select only 9 symbols. So, basically for these 9 symbols we need to find which mapping is, is going to give us the best results. So, one of these mapping is this Excess 3 code. The beauty of this Excess 3 code is it has a very, very nice beautiful property which is called self complementing.

So, if you see here, let us say I want to invert this particular bit 1. So, if I invert all the bits here, what I will get is a 1011. And let us see what do I represent using 1011. So, 1011 is represented using 8, 8 is represented using 1011. So, the beauty here is 8 plus 1 is going to be 9. Similarly, we can do this particular experiment with any of this bit, we will find that when we invert all the 4 bits, then the output is 9 minus that particular number.

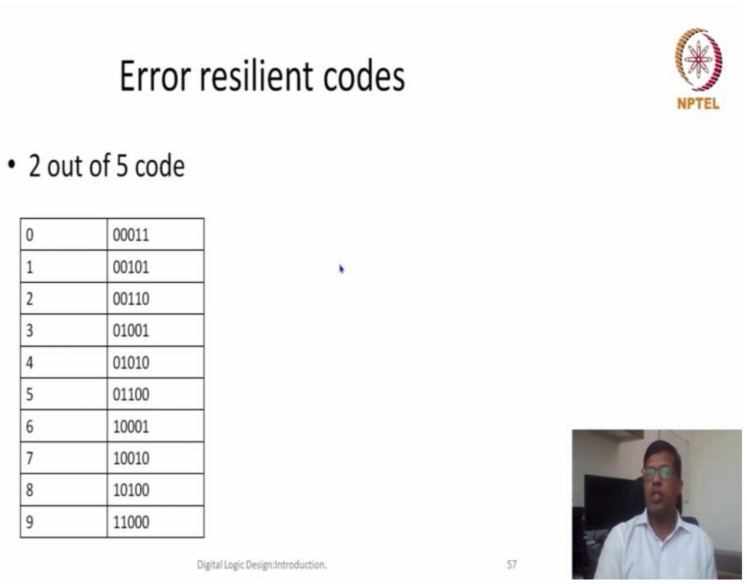
So, this particular property is called self complimenting property and it is a very useful property whenever we are doing subtraction. So, you, you have seen in two's complement numbers, also, they were also complimenting, self complimenting. So, whenever we, we were inverting them, or we were doing finding complement of them, then we were representing was, we wanted to have 2 score n minus that particular number as a complementing number.

So here also, we are achieving the same thing, let us say we want to do subtraction, then what we can do is we can do the complement of that number and then use the addition to do subtraction. So that way, I can do addition and subtraction using the same code, just by inverting the numbers. The other thing is, in our previous 8421 representations, sometimes we had to add a 6, sometimes we need not to have need not to add 6, here, the problem has been has been simplified that in some cases here, we need to add 3, or we have to subtract 3 to fix the final results.

So, this according to hardware generation, it is found to be simpler. If addition and subtraction is done using this excess 3 code. One more thing, we have to understand that all of these codes are mostly invisible to the software developer or to the end user. It is only internal hardware representation. So, it does not matter that whether 0 is represented using 0000 or 0011. One additional place where people have used this excess 3 code is because here 0000, as well as 1111 was not used in encodings.

So, they, they could use all of these things to represent errors or to represent some kind of issues. So those issues could be reported, let us say there was in the communication, there was an error. So it could be reported using the symbol 1111, which means that some, some error has occurred in the in the communication or computation. So, this, this way, this, this form and another representation of our, our BCD numbers. Now can there be something at other possibilities?

(Refer Slide Time: 23:57)



Error resilient codes

- 2 out of 5 code

0	00011
1	00101
2	00110
3	01001
4	01010
5	01100
6	10001
7	10010
8	10100
9	11000

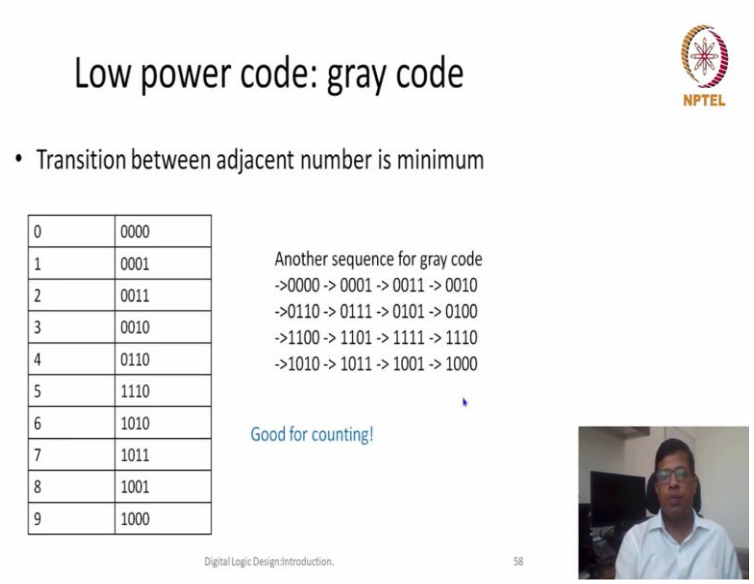
Digital Logic Design:Introduction. 57

So, let us see that if my, my challenge is that I want to design an error resilient code. So, I am working over a noisy channel that if I transmit some number, in the end, I should receive the same number. So, and there is a lot of noise in the channel. That means 0 could flip to 1 or 1 could again flip back to 0. So, if I want to make sure that such kind of errors does not happen, so we introduce some sort of redundancy.

So, this 2 out of 4 code is also a BCD representation because I can represent again, these 10 symbols using this 2 out of 5 code. Here the property is that each of these code is 5 bit code and only two 1s are there. But these two 1s are at different positions. So, if at the output of channel these, the output is not two 1s, but only 1 1 or 3 1s or 4 1s so, that means some error has occurred and we can ask our system to resend the information.

So, this way it acts as at least as a error detection system. So, error could be detected with a greater probability that there is some, some error that has happened. So, here also there are only these 10 combinations and we choose that which combination we would use to represent symbol 0 symbol 1 symbol 2 or symbol 9. So, this is how a 4 error is resilient, we can use some different kind of encoding.

(Refer Slide Time: 25:51)



Low power code: gray code

- Transition between adjacent number is minimum

0	0000
1	0001
2	0011
3	0010
4	0110
5	1110
6	1010
7	1011
8	1001
9	1000

Another sequence for gray code
->0000 -> 0001 -> 0011 -> 0010
->0110 -> 0111 -> 0101 -> 0100
->1100 -> 1101 -> 1111 -> 1110
->1010 -> 1011 -> 1001 -> 1000

Good for counting!

Digital Logic Design:Introduction. 58

Now, there is one more scenario that let us say we wanted to reduce the power. Now, how do we reduce the power we say that the adjacent like transition between adjacent numbers should be should be minimum. So, let us say this is the property of a low power design or a low power system that number of transitions should be minimized and number of transitions cannot be minimized then 1, so, we make sure that whenever we are incrementing a number 0 to 1 total number of transitions are, are at most 1 or always equal to 1 whenever we are incrementing.

So, here you can observe that if this is the representation of 0 and 1 the number of transitions are 1 and from 1 to 2 again the number of transition is 1 only this particular bit is flipping and from 2 to 3 again only this bit get flipped. So, number of transitions is still 1. So, from 3 to 4 also this particular bit will get flipped only 1 bit is flipping otherwise, if you remember from 3 to 4, there will be 3 bits that would flip in a positional number system or a conventional binary number system.

So, in this binary in this gray code, we will have only 1 bit flip from 0 to 1. So, again, let us say we want to represent only 10 digits from, from this 4-bit numbers then we could have several combinations possible. But it is not that like, if I have this 4-bit numbers or 4 bit binary number, I can essentially have all the 16 combinations of gray codes or green numbers I can have 16 different symbols, even gray code is general enough that if I want to represent let us say 32 different symbols are 32 numbers, I can still represent it using, using gray code, the only thing is encoding is slightly different.

So, in our, our digital design, we use these gray codes whenever we would like to reduce the number of transitions between the adjacent numbers. So, if we would like to get a generic method of writing these gray codes, so this table could, could help. So, these are gray code for up to 16 numbers. So here you will find one interesting property that 0 is represented like this, and then 1 and then 1, 1 is representing 2 and 1 0 is representing 3. So, this is also same here.

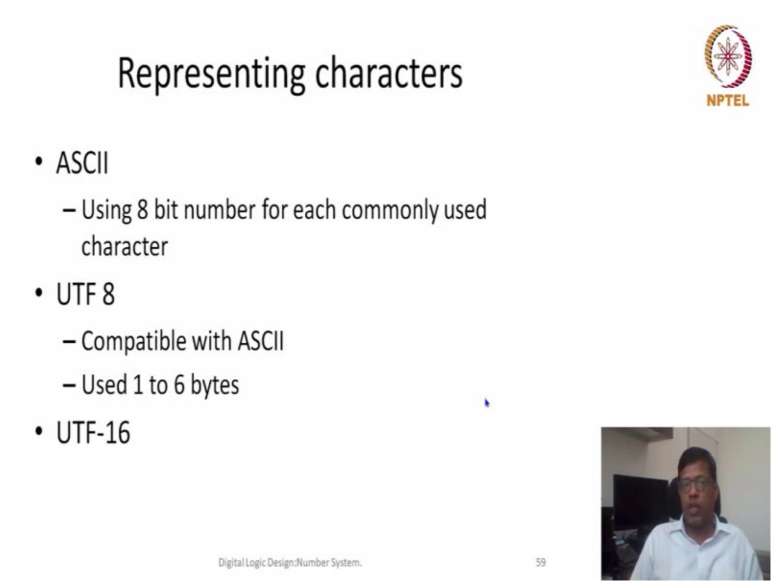
Now after 3, we have made this the next digit third digit as 1, so this become 4. And the rest of the thing is essentially in the inverse order, whatever we have done here, so this will become 11, then 01 and then 00. The next I want to represent after this, this is 4, 5, 6, 7, I want to represent now 8, so for representing 8, I will make the MSB as, as 1 and then we will again, so this from 8 to 12, 8 to 16 it is again the mirror image of 0 to 0 to 7 numbers.

So that means my 8 is represented using 1100. And then 1101 is the image of this. And 1111 is the image of image of this so that way, like in the reverse order, we keep on following all these numbers so we can get 8, 9, 10, 11, 12, 13, 14, 15 so, similarly, from 15, if I want to get, get up to 32 numbers, so, the reverse the order of the numbers would be reverse in the in this order by adding one more 1 here and after that we will go in the reverse order only the MSB would be 1 and the rest of the numbers would be in the reverse order of 0 to 50.

So, this way we can represent any, any size gray scale gray code numbers, the overall conclusion here is that if I want to represent any number we have to we need not to bind ourselves to positional number systems, we have to see that what is the end goal? is the end goal is, is not, not addition not arithmetic, but only a representation of a count or my end goal is counting not the multiplication, not division or not square root not other operations, but only a representation of a number.

Then we can choose any of these codes or maybe we can invent our own code to represent a number so that we can meet our goals. Goals could be power, goals could be a having less possibility of error or goal could be easiness of representation or it could be aligned to the hardware which we are going to interface with. So, with this like we have no, no we, we now know that different how different representations could be there.

(Refer Slide Time: 31:56)



Representing characters

- ASCII
  - Using 8 bit number for each commonly used character
- UTF 8
  - Compatible with ASCII
  - Used 1 to 6 bytes
- UTF-16

Digital Logic Design:Number System. 59

Now, let us say we would like to answer one more question that what if I would like to I do not want to only represent numbers, but I would also like to represent characters like A, B, C or even the special characters like percentage m percent or plus sign of plus some sign of minus. Now you know the answer already because all the textbooks so far you have studied before coming here in, in this class he has seen numbers asking numbers and especially in, in your gel class of computers and introduction to computer science also you have seen these ASCII numbers.

So, these ASCII numbers are commonly used to represent any kind of characters or any kind of special character which are there on the keyboard or commonly found characters. They also use these control characters like enter tab all those things. So, they are they are representing characters and for to represent each character they use these 8 bit numbers, now and if I have an array of characters that also you know that let us say we call them strings, and then we use conservatively these 8 bit numbers or array of bytes to represent a string.

And then again, one more thing there that let us say I do not want only understand the characters, which are given in this English keyboard, we are Indians. So, there we should be able to represent Hindi characters Punjabi characters and we have at least 18 other languages which are notified in constitution and 100 other languages which are not notified. So, to represent any character in all those languages, we require some extended set.

So, this extended set is the one which is popular which is standardized is called UTF-8. So, in UTF-8, it can have more number of bytes. So, instead of 1 byte would be as same as ASCII character, but at rest, there could be other bytes which could, could be represented which can which can have the extended coding. So, this, this UTF-8 and UTF-16 both have been standardized by, by standardizing bodies and it can be used to represent all the extended character sets which is possible anywhere in the world.

So, the there are some languages which has so many different kinds of bytes and characters representation there. So, this UTF-8, UTF-16 can be used to extend to can be used to represent almost any kind of letters which are there in any language in the world. And people keep on extending if there is a new characters for example, some couple of years back a rupee symbol was introduced, so that was also added to UTF-8 and UTF-16.

So, that we can add these new symbols also, which can be standardized by computer programs and the standardized information can be used for printing. So, this way we can we have almost seen any kind of information could be represented using, using binary numbers. Still one question remains in starting of this course, we said that binary numbers means they would be quantized they would be discrete, is it does it mean that I cannot represent a real number a real number means a number which would have decimal points, rational numbers or irrational numbers, irrational numbers which they could be infinite number of digits after decimal points.

So, can he represent those real numbers, numbers which are not quantified or which are not quantized or which are not discrete in nature? If we cannot represent real numbers, then our computation would be severely limited to integers. So, this is the question we will try to answer in our next lecture. So far, thank you very much, and enjoy doing your exercises and tutorials. Thank you.