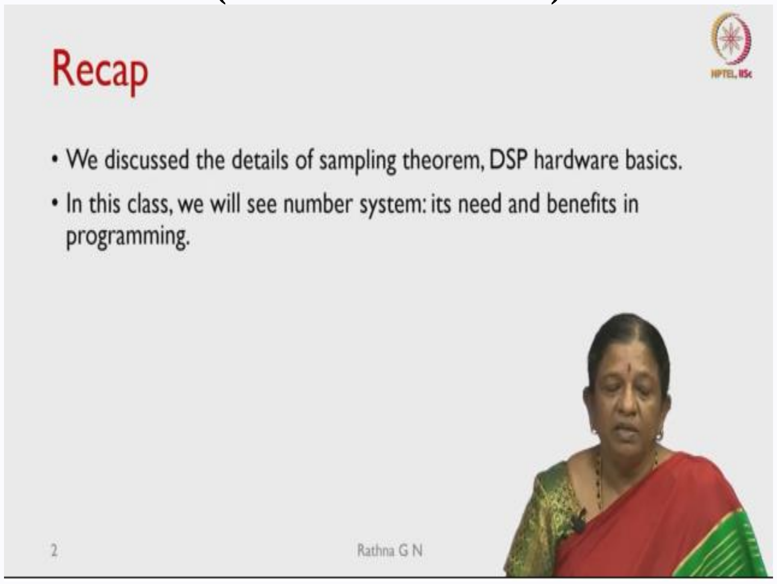
Real - Time Digital Signal Processing
Prof. Rathna G N
Department of Electrical Engineering
Indian Institute of Science - Bengaluru

Lecture - 04
Number System

(Refer Slide Time: 00:28)



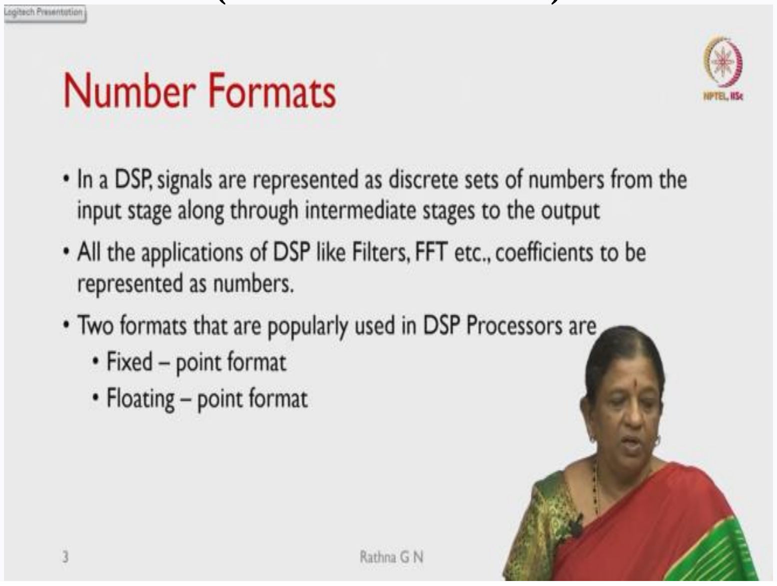
Recap

- We discussed the details of sampling theorem, DSP hardware basics.
- In this class, we will see number system: its need and benefits in programming.

2 Rathna G N

Welcome back to real time signal processing. So, today we will discuss about the number system. Just to give a recap of what we covered in the last class. First, we did sampling theorem, some DSP hardware basics, what we looked into. So in this class we will see number system, its needs and then benefits in programming, although you may think that it is a primary class section, but we will see that how it is beneficial for implementing it in hardware. That is what we will be looking at today.

(Refer Slide Time: 01:01)



Number Formats

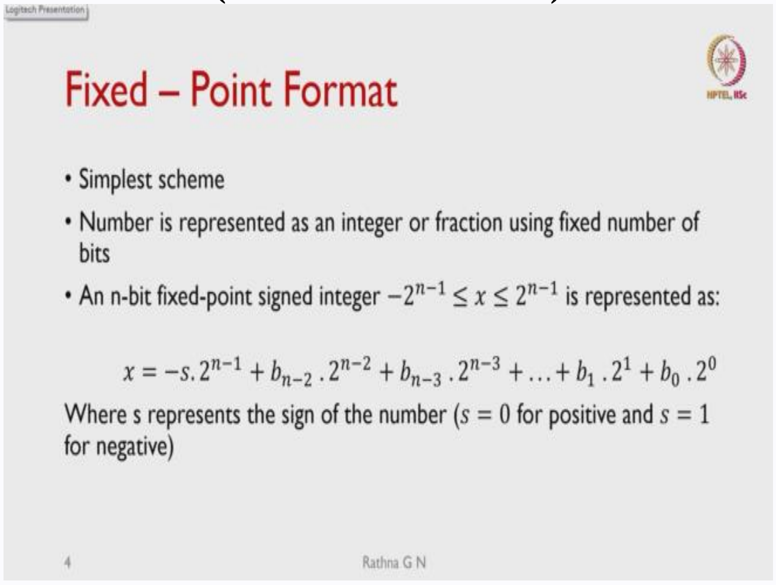
- In a DSP, signals are represented as discrete sets of numbers from the input stage along through intermediate stages to the output
- All the applications of DSP like Filters, FFT etc., coefficients to be represented as numbers.
- Two formats that are popularly used in DSP Processors are
  - Fixed – point format
  - Floating – point format

3 Rathna G N

So first, we will see what are the number formats available. So, we know in DSP signals are represented as discrete sets of numbers from the input stage along through intermediate stages to the output. So, all the stages have to be in the discrete format and all the applications of DSP like filters, FFT, coefficients to be represented all as numbers. So there are 2 formats popularly available in DSP processors, one is the fixed point format.

The other one is the floating point format, most of you might have used the floating point format, still, we will see how to represent that in DSP process, we will look into the thing.

(Refer Slide Time: 01:51)



Logitech Presentation

Fixed – Point Format

- Simplest scheme
- Number is represented as an integer or fraction using fixed number of bits
- An n-bit fixed-point signed integer $-2^{n-1} \leq x \leq 2^{n-1}$ is represented as:

$$x = -s.2^{n-1} + b_{n-2}.2^{n-2} + b_{n-3}.2^{n-3} + \ldots + b_1.2^1 + b_0.2^0$$

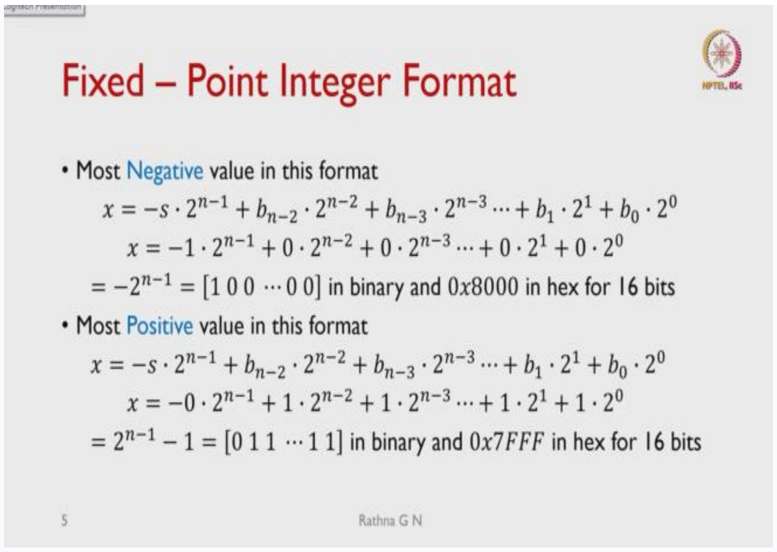
Where s represents the sign of the number ($s = 0$ for positive and $s = 1$ for negative)

4 Rathna G N

So coming to the fixed point format, so, we know that it is the simplest scheme. So, we represent the number as integer or fraction using fixed number of bits. So, we know that an n bit fixed point signed integer as we mentioned it again it is a signed integer what I am representing, so, the representation will be from -2^{n-1} to 2^{n-1} and we know x is the number what we are representing.

So which is given by so minus s dot here s represents the sign bit of the number if it is equal to 0 it is positive number and if it is minus 1, if it is equal to 1 it is a negative number. So, the first number will be $-s.\ 2^{n-1}$ and then the next number will be its magnitude is b_{n-2} what will take it into 2^{n-2} and so on, in the last we have $b_1\ .\ 2^1$ and then $b_0\ .\ 2^0$

(Refer Slide Time: 03:09)



Fixed – Point Integer Format

- Most Negative value in this format

$$x = -s \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + b_{n-3} \cdot 2^{n-3} \cdots + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

$$x = -1 \cdot 2^{n-1} + 0 \cdot 2^{n-2} + 0 \cdot 2^{n-3} \cdots + 0 \cdot 2^1 + 0 \cdot 2^0$$

$= -2^{n-1} = [1\ 0\ 0\ \cdots 0\ 0]$ in binary and $0x8000$ in hex for 16 bits

- Most Positive value in this format

$$x = -s \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + b_{n-3} \cdot 2^{n-3} \cdots + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

$$x = -0 \cdot 2^{n-1} + 1 \cdot 2^{n-2} + 1 \cdot 2^{n-3} \cdots + 1 \cdot 2^1 + 1 \cdot 2^0$$

$= 2^{n-1} - 1 = [0\ 1\ 1\ \cdots 1\ 1]$ in binary and $0x7FFF$ in hex for 16 bits

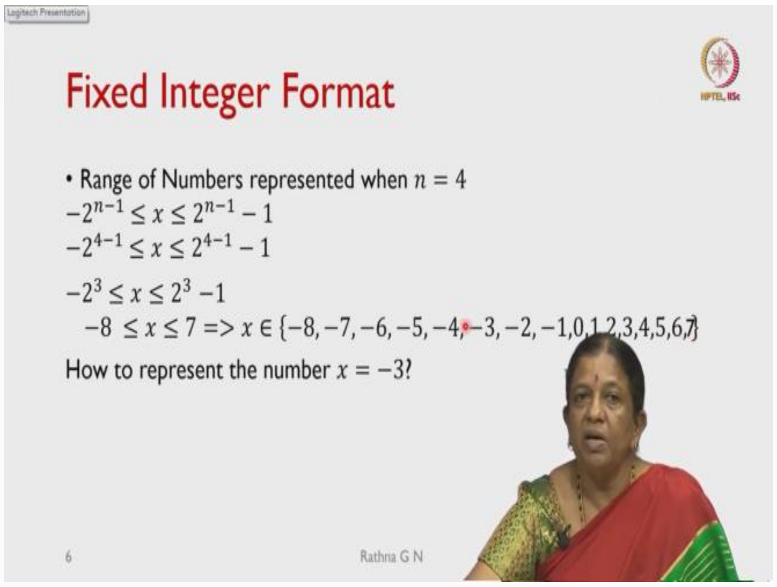
5 Rathna G N

So, all of us know that what is the range this number format can take. So we will see the most negative value in this format is x is given by $-s.\ 2^{n-1}$ to $b_0\ .\ 2^0$. So, if we know the negative bit is equal to 1, so, if a substitute s=1 it is -1.2^{n-1} and rest of the bits is going to be 0. So, this is equivalent to -2^{n-1}. So which is represented in binary format as 1 followed by all zeros.

And then if we use 16 bit representation for this number, then the binary format for most negative number is going to be $0, x, 8, 0, 0, 0$ in hex, and then the most positive value in this format what it can take is $-s.\ 2^{n-1}$. So, this is a number what we have represented as earlier, and then we know the sign bit is going to be zero for positive numbers. So x is represented as $-0 \cdot 2^{n-1}$ and we consider all the rest of the bits later as one then it becomes the number representation is equivalent to $2^{n-1} - 1$.

So, in the binary format, it is going to be 0 because it is a positive number. And rest of the numbers magnitude is represented as 1 in all through in a 16 bit format. The maximum value is given as $0, x, 7, F, F, F$ in hex, we know the number representation hexadecimal, I think most of you would have covered the thing. So, F is the maximum number, which gives us the 15 as the value for it.

(Refer Slide Time: 05:13)



Fixed Integer Format

- Range of Numbers represented when $n = 4$

$-2^{n-1} \leq x \leq 2^{n-1} - 1$

$-2^{4-1} \leq x \leq 2^{4-1} - 1$

$-2^3 \leq x \leq 2^3 - 1$

$-8 \leq x \leq 7 \Rightarrow x \in \{-8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7\}$

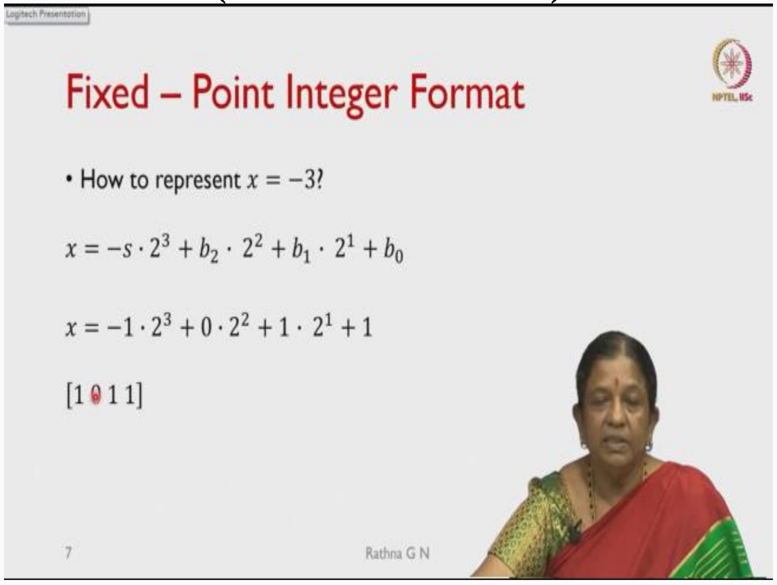
How to represent the number $x = -3$?

6 Rathna G N

So the next one, so we have to represent this number in the integer format. So we know that in when n = 4, the value, what I can represent in integer format is $-2^{n-1} \leq x \leq 2^{n-1} - 1$. So in this, we have substituted $n = 4$. So if we substitute in the next line, so we know that $-2^{4-1} \leq x \leq 2^{4-1} - 1$. So the range is going to be $-2^3 \leq x \leq 2^3 - 1$, which is nothing but $-8 \leq x \leq 7$.

So in this case, we know that $x \in \{-8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, \}$, and then the positive side it is $0, 1$ to 6. And then the last one is 7 in this slide. So we will see that how to represent our number $x = -3$.

(Refer Slide Time: 06:25)



Fixed – Point Integer Format

- How to represent $x = -3$?

$x = -s \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0$

$x = -1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1$

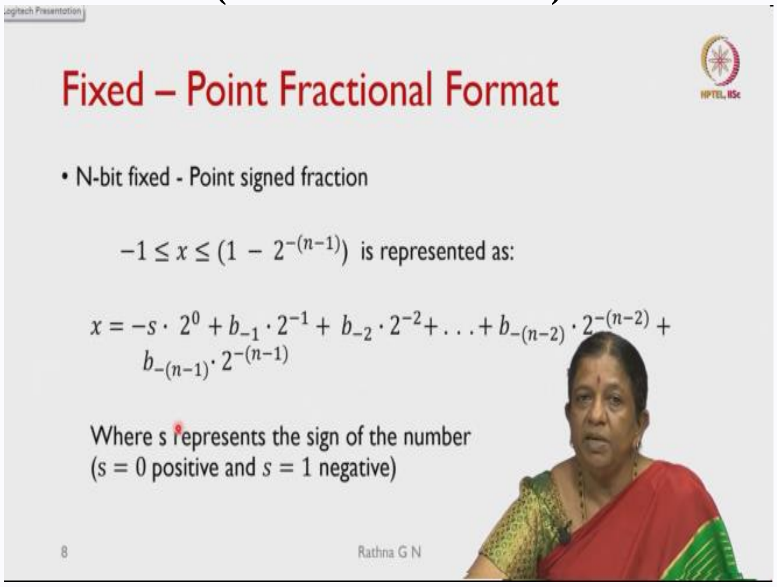
[1 0 1 1]

7 Rathna G N

So, this is the integer format so we said $x = -3$ how we can represent. So, we know that this is a negative number, so $s = 1$. And then we know that when I divide $3/2$, I will be getting one. So the digit what it will be is $1 \cdot 2^1 + 1$ will give me $+3$. And then since we are using the

4 bit representation, this bit is going to be zero. So the number what I can represent in binary is 1 which gives the negative value and $0, 1, 1$ is -3.

(Refer Slide Time: 07:11)



Fixed – Point Fractional Format

- N-bit fixed - Point signed fraction

$-1 \leq x \leq (1 - 2^{-(n-1)})$ is represented as:

$x = -s \cdot 2^0 + b_{-1} \cdot 2^{-1} + b_{-2} \cdot 2^{-2} + \ldots + b_{-(n-2)} \cdot 2^{-(n-2)} + b_{-(n-1)} \cdot 2^{-(n-1)}$

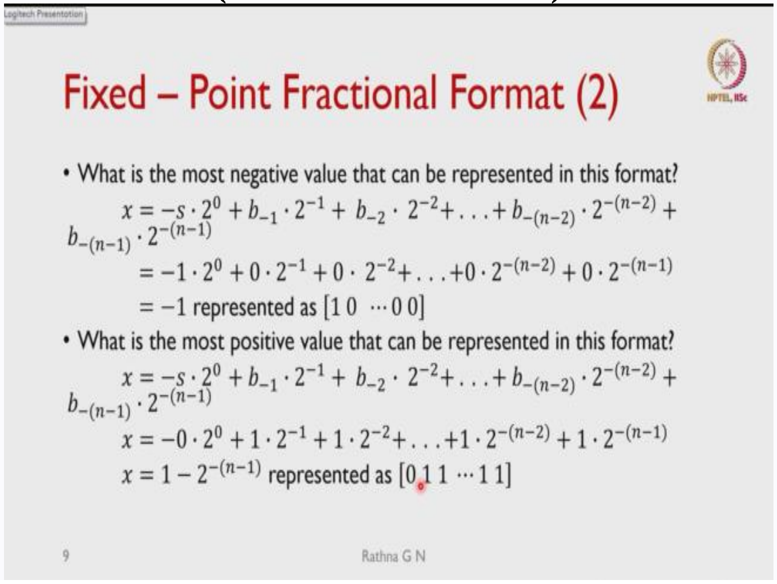
Where s represents the sign of the number ($s = 0$ positive and $s = 1$ negative)

8 Rathna G N

So coming to why we have to go for the fixed point, fractional format, we will see it in a while. First, we will discuss what is this number system? So the N bit fixed point signed fraction or representation, we see that the number x can be represented between -1 to $(1 - 2^{-(n-1)})$. So n is the number of bits what we will be using as normal notion.

So if I represent this in the open format, so x will be $-s \cdot 2^0 + b_{-1} \cdot 2^{-1} + b_{-2} \cdot 2^{-2} + \ . \ . \ . + b_{-(n-2)} \cdot 2^{-(n-2)}$ and then the last number will be represented as $b_{-(n-1)} \cdot 2^{-(n-1)}$. So here, again, s represents sign of the number, if it is equal to 0, it is positive, and if it is equal to 1, it is a negative number.

(Refer Slide Time: 08:14)



Fixed – Point Fractional Format (2)

- What is the most negative value that can be represented in this format?

$x = -s \cdot 2^0 + b_{-1} \cdot 2^{-1} + b_{-2} \cdot 2^{-2} + \ldots + b_{-(n-2)} \cdot 2^{-(n-2)} + b_{-(n-1)} \cdot 2^{-(n-1)}$

$= -1 \cdot 2^0 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + \ldots + 0 \cdot 2^{-(n-2)} + 0 \cdot 2^{-(n-1)}$

$= -1$ represented as $[1\ 0\ \cdots 0\ 0]$

- What is the most positive value that can be represented in this format?

$x = -s \cdot 2^0 + b_{-1} \cdot 2^{-1} + b_{-2} \cdot 2^{-2} + \ldots + b_{-(n-2)} \cdot 2^{-(n-2)} + b_{-(n-1)} \cdot 2^{-(n-1)}$

$x = -0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + \ldots + 1 \cdot 2^{-(n-2)} + 1 \cdot 2^{-(n-1)}$

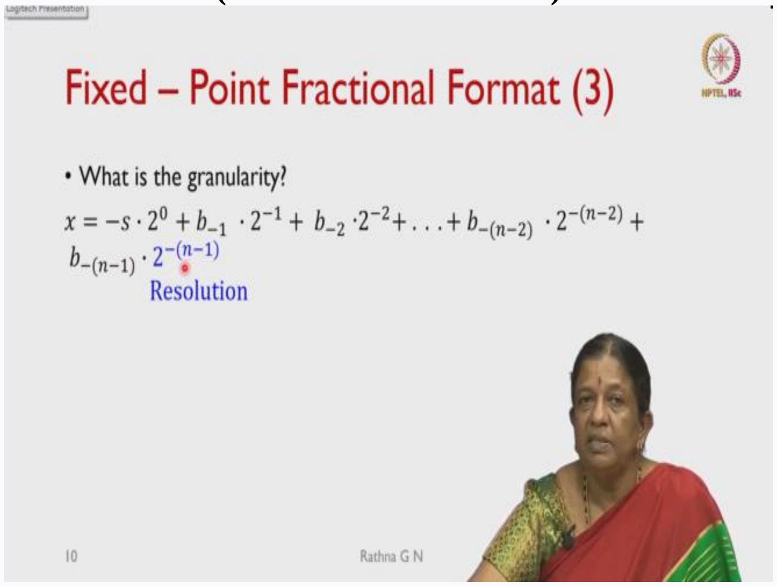
$x = 1 - 2^{-(n-1)}$ represented as $[0\ 1\ 1\ \cdots 1\ 1]$

9 Rathna G N

So we will see how for a fractional format can be represented. So that is the most negative value how we did for the integer format, so we will see for the fractional format, the most negative number, so we will be represented as $s = -1$ and then rest of them are going to be zeros. So which is going to be equal to minus 1 and this is represented as our regular thing 1, all zeros, which is $0, x, 8, 0, 0$ format.

And then what is the most negative value that can be represented in this format, what we will see the thing, so, we have $s = 0$ and rest of the values are going to be 1. So it is represented as $1 - 2^{-(n-1)}$. So in the binary format will be representing as zero the point is, whatever is not given any place bit value is given for the decimal value. So it will be zero point all ones or in the process, it will be taking it as 0, 1, 1, 1, 1. But the notation for us is $x = 1 - 2^{-(n-1)}$ will be the largest value what I can represent in this format.

(Refer Slide Time: 09:42)



Fixed – Point Fractional Format (3)

- What is the granularity?

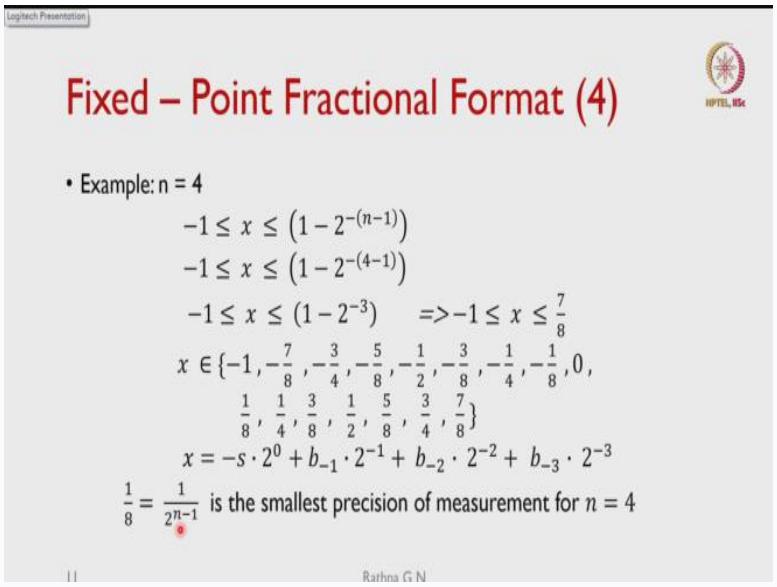
$x = -s \cdot 2^0 + b_{-1} \cdot 2^{-1} + b_{-2} \cdot 2^{-2} + \ldots + b_{-(n-2)} \cdot 2^{-(n-2)} + b_{-(n-1)} \cdot 2^{-(n-1)}$

Resolution

10 Rathna G N

So we will see what will be the granularity of this? Why we have to go for the point, fractional format? So which is given us this way, so $-n \cdot 2^0$. What I have it so in the resolution in this case is going to be $2^{-(n-1)}$ will be my granularity what I can have from this representation.

(Refer Slide Time: 10:09)



Fixed – Point Fractional Format (4)

- Example: n = 4

$$-1 \le x \le \left(1 - 2^{-(n-1)}\right)$$
$$-1 \le x \le \left(1 - 2^{-(4-1)}\right)$$
$$-1 \le x \le (1 - 2^{-3}) \quad => -1 \le x \le \frac{7}{8}$$
$$x \in \{-1, -\frac{7}{8}, -\frac{3}{4}, -\frac{5}{8}, -\frac{1}{2}, -\frac{3}{8}, -\frac{1}{4}, -\frac{1}{8}, 0, \frac{1}{8}, \frac{1}{4}, \frac{3}{8}, \frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8}\}$$
$$x = -s \cdot 2^0 + b_{-1} \cdot 2^{-1} + b_{-2} \cdot 2^{-2} + b_{-3} \cdot 2^{-3}$$

$\frac{1}{8} = \frac{1}{2^{n-1}}$ is the smallest precision of measurement for $n = 4$

11 Rathna G N

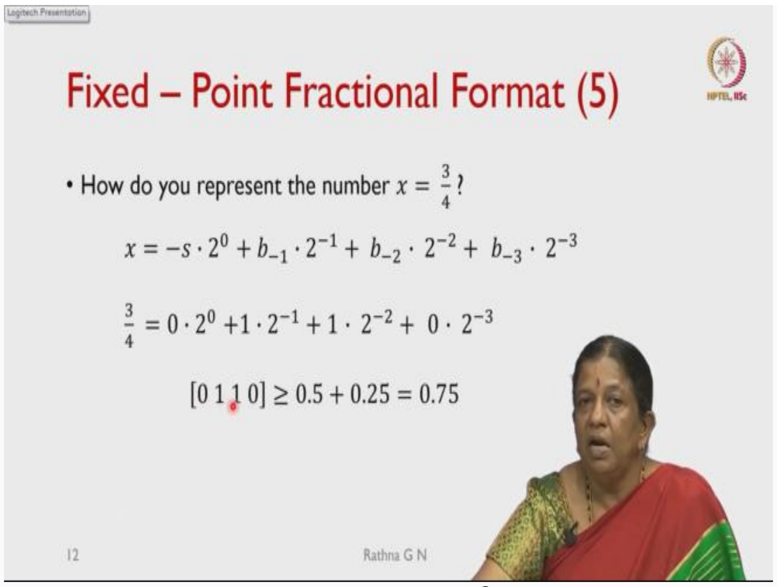
So coming with the example, so we will see $n = 4$, what are the numbers I can represent in the fractional format, so, which is $-1 \le x \le \left(1 - 2^{-(n-1)}\right)$. So, we are substituting $n = 4$ here. So which comes out as $-1 \le x \le (1 - 2^{-3})$, which is nothing but x should be $x \le \frac{7}{8}$ that is what we have to represent then what is the x values it will be taking from -1 to $+\frac{7}{8}$ in what we call it as the smallest value will be -1.

Then next will be minus $\frac{7}{8}$, and so on $-\frac{3}{4}$and then $-\frac{5}{8}$. You may be wondering why I have got $-\frac{3}{4}$, just check $7 \cdot 2$ is going to be 14, $\frac{14}{16}$, what you will be doing it or you will be adding $-\frac{7}{8}$ $-\frac{7}{8}$ 8 will be giving you this value when you simplify the value of it. So like this, you will be going on representing till 0. So and then later on the positive side what you will be representing it as $\frac{1}{8}$and so on, the last number will be $\frac{7}{8}$.

As you can see that how many values you have represented in the negative side and then how many values are represented on the positive side as you will be seeing that it has the 2, 4, 6 and 8. So, that is what the values what you will represent it as on the a negative side and positive side, you will be seeing that it will be 7 as we have we know that 2^{-4-1} is going to be 1- this is going to be 7 in that case, magnitude of it.

So we will see that what is the smallest precision measurement I can have with $n = 4$ which is nothing but $\frac{1}{8}$, which is coming from $\frac{1}{2^{n-1}}$what we said in the previous slide.

(Refer Slide Time: 12:28)



Fixed – Point Fractional Format (5)

- How do you represent the number $x = \frac{3}{4}$?

$$x = -s \cdot 2^0 + b_{-1} \cdot 2^{-1} + b_{-2} \cdot 2^{-2} + b_{-3} \cdot 2^{-3}$$

$$\frac{3}{4} = 0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3}$$

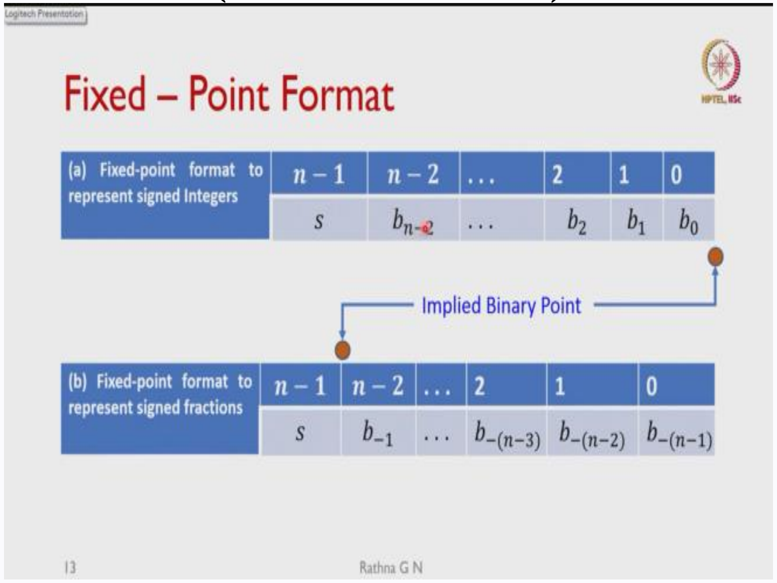
$$[0\ 1\ 1\ 0] \geq 0.5 + 0.25 = 0.75$$

12 Rathna G N

So moving on, so how I can represent the number $x = \frac{3}{4}$. So we will see in the fractional format, how we are going to represent it. So, we know that $-s \cdot 2^0$ this is the representation what we are considering for the fractional format, and then we have to substitute as $\frac{3}{4}$ which is equal to $0 \cdot 2^0 + 1 \cdot 2^{-1}$ which is nothing but 0.5 and then the next number is going to be $+1 \cdot 2^{-2}$ which is 0.25, we all know that $\frac{3}{4}$ in the fractional representation is 0.75.

So, 0.5 is given by this weight and then the next bit will be representing 0.25. So $0.5 + 0.25 = 0.75$. So, the binary representation is $0, 1, 1, 0$ since it is a positive value, we have put it as 0 and then we have these two are the representation the last bit is going to be 0.

(Refer Slide Time: 13:41)



Fixed – Point Format

(a) Fixed-point format to represent signed Integers	$n-1$	$n-2$	...	2	1	0
	s	b_{n-2}	...	b_2	b_1	b_0

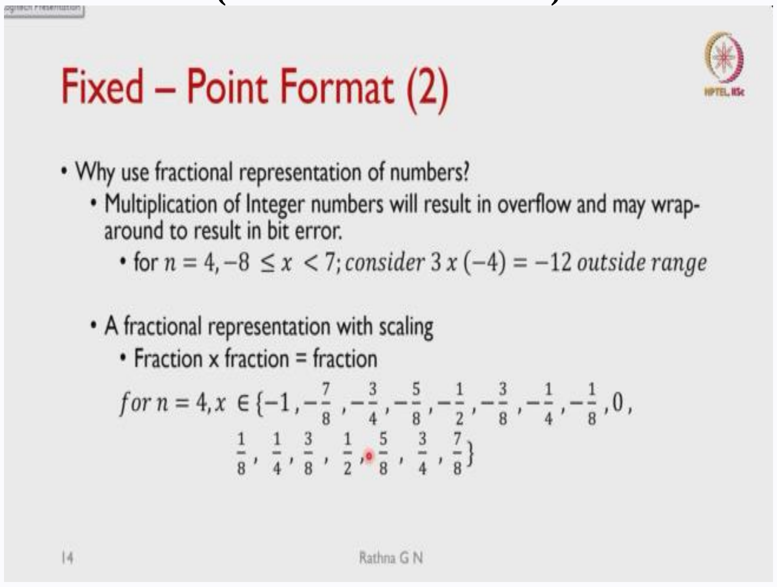
(b) Fixed-point format to represent signed fractions	$n-1$	$n-2$	...	2	1	0
	s	b_{-1}	...	$b_{-(n-3)}$	$b_{-(n-2)}$	$b_{-(n-1)}$

13 Rathna G N

Coming with the fixed point format and then how we are representing it all of us know that even the integer representation is a fixed point number system, but where is the point actually, you will be seeing that we usually discard this it comes at the end of the number. So, the decimal point will be at the end of whatever number you have considered. And then the same you will be seeing it in the fixed point format, I have the sign bit and immediately there is this thing what we assume is a decimal point.

So this since it does not have any bit allocation for it to say for my rest of the numbers, we assume that it represented are immediately after our sign bit. So, the next numbers what will be representing is b minus 1 and b minus n minus 1. Whereas in the case of integer you will be seeing that sign bit will be having the n minus 1 place and then later on it will be b n minus 1 will be its magnitude and so on till b zero.

(Refer Slide Time: 14:56)



Fixed – Point Format (2)

- Why use fractional representation of numbers?
  - Multiplication of Integer numbers will result in overflow and may wrap-around to result in bit error.
    - for $n = 4, -8 \leq x < 7; consider\ 3\ x\ (-4) = -12\ outside\ range$
  - A fractional representation with scaling
    - Fraction x fraction = fraction

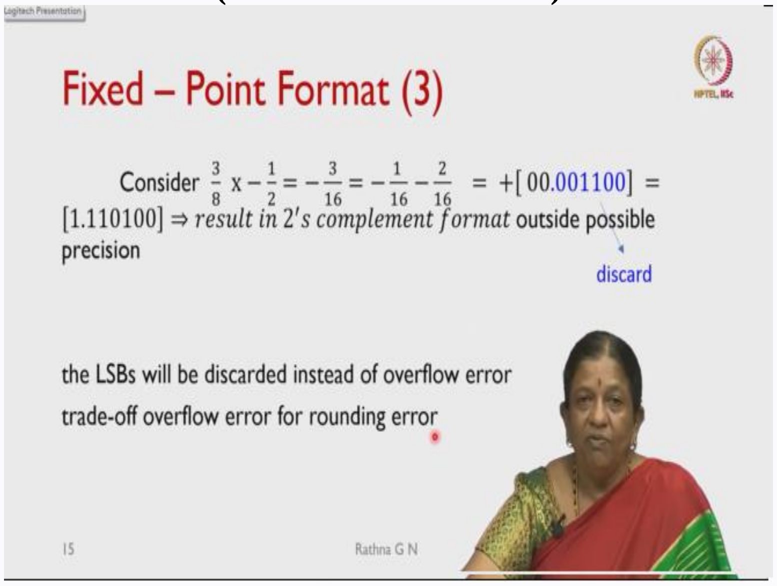
$$for\ n = 4, x \in \{-1, -\frac{7}{8}, -\frac{3}{4}, -\frac{5}{8}, -\frac{1}{2}, -\frac{3}{8}, -\frac{1}{4}, -\frac{1}{8}, 0, \frac{1}{8}, \frac{1}{4}, \frac{3}{8}, \frac{1}{2}, \frac{5}{8}, \frac{3}{4}, \frac{7}{8}\}$$

14 Rathna G N

So coming to next what is the fixed point format? So why we have to use the fractional representation of numbers, this is what we will see the thing, all of us know that any multiplication of integer numbers will result in overflow, and may wrap around to result in bit error. So hold on for a while, we will see that how it can happen. So for an n = 4, the representation, what we have is minus 8, x can represent between -8 to 7.

So consider the multiplication of $3 \times (-4)$. So the result has to be -12 all of us know. So you will wonder why I am talking about the primary class multiplication. So but as we can see that the range, what I can represent is between -8 to 7, so -12 is outside the range. So the same thing, if I represent in the fractional representation, so I assume $n = 4$. So then what I will be having is the representation in this format. So how can I do that? Basically, whether can I bring this one in this format, so that I will not be outside the range of it.

(Refer Slide Time: 16:17)



Fixed – Point Format (3)

Consider $\frac{3}{8}$ x $-\frac{1}{2} = -\frac{3}{16} = -\frac{1}{16} - \frac{2}{16}$ = +[00.001100] = [1.110100] ⇒ *result in 2's complement format* outside possible precision

discard

the LSBs will be discarded instead of overflow error

trade-off overflow error for rounding error

15 Rathna G N

So for that, what we will do is, we will how we can represent this $\frac{3}{8} \times \text{x} \times -\frac{1}{2} = -\frac{3}{16}$. So what did we do? Can you check the thing here, so we have because $n = 4$ format, I have taken the thing, so the number what I can represent is 8, basically, so I have divided the number $\frac{3}{8}$, the same thing, what I did was $-\frac{4}{8}$, I have done here, so which is nothing but $-\frac{1}{2}$.

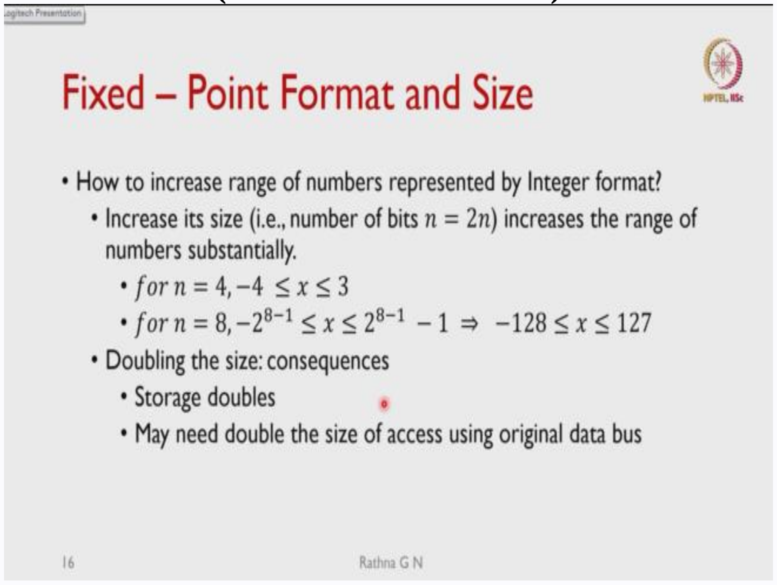
So when I multiply these two numbers, the result is going to be $-\frac{3}{16}$, which is nothing but I can represent it in this format, $-\frac{1}{16} - \frac{2}{16}$, then what happens to the number in the binary format, so we result in, we know that it should be a negative number. So initially, what we will put it is a plus value, what we will be giving the thing 0, 0, 1, 1 by 16, what we have it, and then the next one is $\frac{2}{16}$ is 1 and 0, 0.

So, which is equivalent because this is a positive number, what I have represented, I have not represented the negative value, here, I will be considering the 2's complement of this number. When I take that 2's complement, I will be the number what will be represented is 1.110100. So that is what it says is we will be having in 2's complement format. So what we say is, it is outside possible precision, what we say the thing.

So, in this case, what we do is we will be discarding one of the zeros here. And then the lower values, we can discard and then take only the 4 bits. So that is what we say the LSBs will be discarded instead of overflow error. So that is what the thing, what is the trade off, in this case?

Trade off is overflow error for rounding error, what we have to consider? What is the value represented? Can you tell me the thing in this case.

(Refer Slide Time: 18:50)



Fixed – Point Format and Size

- How to increase range of numbers represented by Integer format?
  - Increase its size (i.e., number of bits $n = 2n$) increases the range of numbers substantially.
    - $for\ n = 4, -4 \leq x \leq 3$
    - $for\ n = 8, -2^{8-1} \leq x \leq 2^{8-1} - 1 \Rightarrow -128 \leq x \leq 127$
  - Doubling the size: consequences
    - Storage doubles
    - May need double the size of access using original data bus

16 Rathna G N

So now, we know that we have a trade off between overflow and then rounding error, what it is coming in, can I work on it to avoid these things. So we say how to increase range of numbers represented by integer format, first, we will consider it. So we can say that because my multiplication is going to cause me twice that of the number of bits what I need it, so whether I can increase the number of bits n to $2n$.

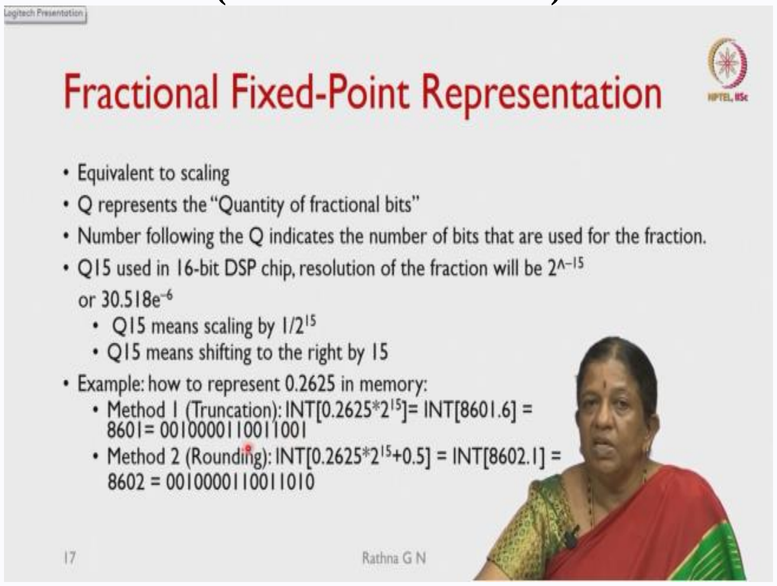
So this will increase the range of numbers substantially. So for $n = 4$, my representation is between -4 to 3, when I increase by 2 times this n, then $n = 8$. So the range of numbers what I will be representing is $-2^{8-1} \leq x \leq 2^{8-1} - 1$. So which is equivalent to $-128 \leq x \leq 127$.

So what will be the consequence of doubling the size of this number, one is, our storage has to double, because I have to, instead of 4 bit numbers, I will be storing them as 8 bit, this is a smaller example what we have taken. So this is not the scenario in real time. So real time, most of the numbers will be 16 bit what we can represent in hardware, most of the things, I think all of you will be using the number format as 32 bit long is 64 bit.

And then a long and then longer double precision, what we will be using is 64 bit, but you will be seeing that how the consequence is going to be nearby. So, the storage is going to double so my hardware has to increase. So what we are discussing is the real time signal processing. So real time capabilities is going to come down so that we can do that or not, we will see when we take up some examples.

And then we may need double the size of access using the original data bus. When we take up the architecture, we will see that by increasing this how my bus width is going to increase and then how it can increase the hardware as well as how it can slow down and one more thing you have increased it to 2n. But if I store this value and then use it with my 4 bit value, next I will be getting it as 12 bit can I go on increasing in this fashion forever, it is going to be a what I will say is no end for it for this increasing of the numbers.

(Refer Slide Time: 21:46)



Fractional Fixed-Point Representation

- Equivalent to scaling
- Q represents the "Quantity of fractional bits"
- Number following the Q indicates the number of bits that are used for the fraction.
- Q15 used in 16-bit DSP chip, resolution of the fraction will be 2^{-15} or $30.518e^{-6}$
  - Q15 means scaling by $1/2^{15}$
  - Q15 means shifting to the right by 15
- Example: how to represent 0.2625 in memory:
  - Method 1 (Truncation): $INT[0.2625*2^{15}] = INT[8601.6] = 8601 = 0010000110011001$
  - Method 2 (Rounding): $INT[0.2625*2^{15}+0.5] = INT[8602.1] = 8602 = 0010000110011010$

17 Rathna G N

So hence, we have to restrict the thing. So, that is the reason why we use the fractional fixed point representation. So it is what we call it as equivalent to scaling. And then usually the fractional number representation is going to be represented in Q format. So what is that Q represent is quantity of fractional bits, I will be using in my implementation, the number following the queue indicates the number of bits that are used for the fractional representation as they represent as Q15.

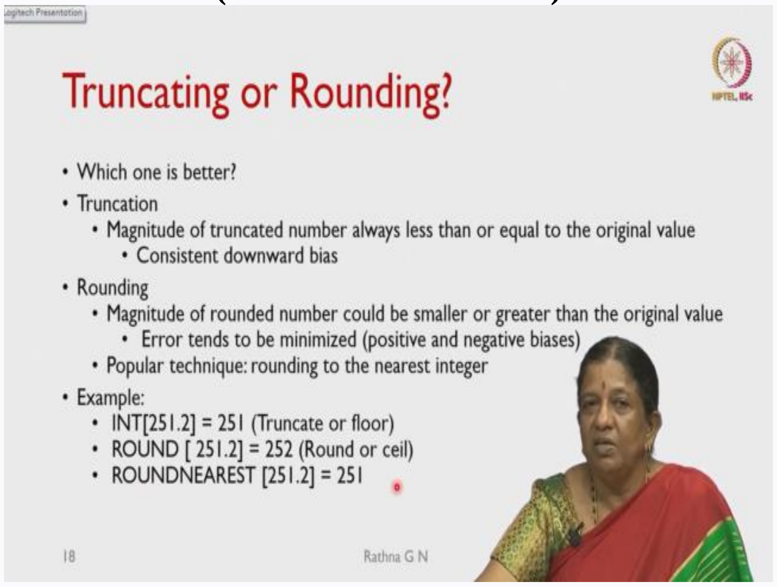
I know it is a 16-bit DSP chip; basically, the resolution of the fractional will be 2 to the power of minus 15. So or it can be $30.518*e^{-6}$ in decimal value what I can represent. So what are the things here, Q15 means I will be scaling the number by 15 that is $1/2^{15}$. And then once again Q15 means either I can scale by this division or I can do the shifting because all 1 shift right operation is going to be divided by 2.

So I can shift right by 15 times so that I will be resulting in the value of it. So how will be incorporating when I take up the architecture we will be coming back to this number, how I can do this shifting otherwise, each shifting will be costing us every clock cycle. So I need 15 clock cycles to shift if I have want to do 15 bit right shift, but we will be doing it in 1 clock cycle.

We will look into this in the architecture when we take. So now as an example, how we are going to represent 0.625 in memory. The first one is we can use the truncation method 1 what we call it, so we will be calling it as an integer value. So which is nothing but $0.2625 * 2^{15}$ what I am multiplying it the resulting integer values 8601.6 in the truncation will be truncating that whatever number coming out of the decimal point, so it will be represented as 8601. So you will be seeing in the 15 bit format.

This is how in the binary 15 bit binary format is this one. And in the second case, we can do the rounding. So, what we will be doing, so we will be adding 0.5 into the result what we are getting it so, then we will be discarding the normal way are whatever number comes out of the decimal point so which will be 8602 here the binary representation what you will be seeing, so you have added 0.5 so the number has gone to $+1$ in this case, so it will be 1, 0 the rest of the numbers remain same. So this is the positive number what we have represented.

(Refer Slide Time: 24:57)



Truncating or Rounding?

- Which one is better?
- Truncation
  - Magnitude of truncated number always less than or equal to the original value
    - Consistent downward bias
- Rounding
  - Magnitude of rounded number could be smaller or greater than the original value
    - Error tends to be minimized (positive and negative biases)
  - Popular technique: rounding to the nearest integer
- Example:
  - INT[251.2] = 251 (Truncate or floor)
  - ROUND [251.2] = 252 (Round or ceil)
  - ROUNDNEAREST [251.2] = 251

18 Rathna G N

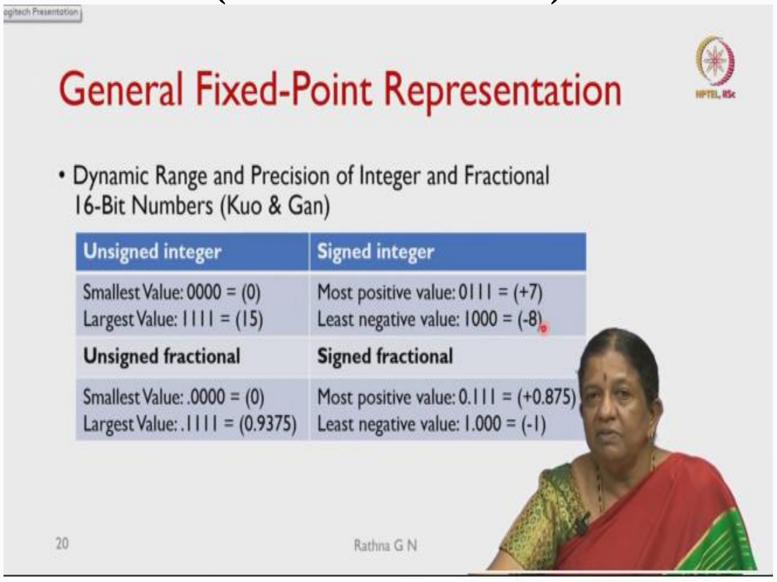
So whether you want to go for truncation or rounding, which one is better? So, we will see what happens with the truncation. So in this case the magnitude of the truncated number always less than or equal to the original value. So that is we will be seeing that it is consistently downward bias what I am going to have it whereas in the rounding, what happens magnitude of rounded number could be smaller or greater than the original value. So then what happens to the thing?

Error tends to be minimised, that is positive and negative bias is what I am going to have it and popular technique is rounding to the nearest integer. So, we will be seeing in hardware when I have to do rounding then I have to add 0.5 to whatever the number what it is resulting in and then to the represented to the nearest value. So, there is additional addition is going to happen. So, which will be costing us again extra clock cycle.

So, the simplest one will be a truncation, but if you are not able to get the required result, then you have to go for the rounding and then have the additional whatever clock cycle that is going to result from it. So as you can see INT 251.2 so, we know that it is truncate or we call it as floor it is 251 it is not going to cause any damage to the number represented whereas if it is rounding also it will be resulting in the same value.

Sorry, rounding is you are adding 0.5, so it becomes a 251.7 so, which is going to be rounded to 252. So, we call it as rounder, it is going to be sealed value what you will be considering in normal number representation or I can have a round nearest one more distinct function defined 251.2 which is going to be 251. So, you will be seeing that truncation and rounding to nearest almost it is matching, whereas a rounding has increased the value of the number.

(Refer Slide Time: 27:24)



General Fixed-Point Representation

- Dynamic Range and Precision of Integer and Fractional 16-Bit Numbers (Kuo & Gan)

Unsigned integer	Signed integer
Smallest Value: 0000 = (0) Largest Value: 1111 = (15)	Most positive value: 0111 = (+7) Least negative value: 1000 = (-8)
Unsigned fractional	**Signed fractional**
Smallest Value: .0000 = (0) Largest Value: .1111 = (0.9375)	Most positive value: 0.111 = (+0.875) Least negative value: 1.000 = (-1)

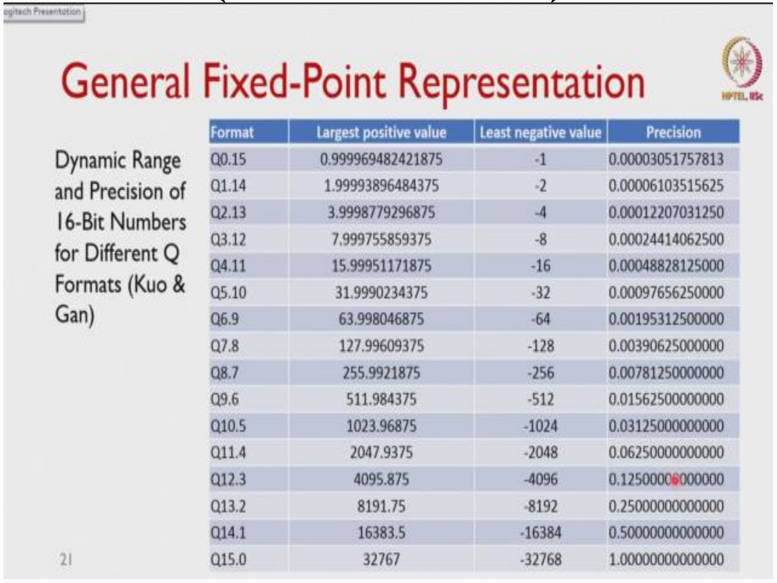
20 Rathna G N

So, we will see that general fixed point representation how we can do that is given by Kuo and then Gan for 16 bit numbers. So, if it is an unsigned integer, and then how it is going to represent it in the signed integer format, the smallest value for unsigned integer is 0 and then the largest value it is going to be $1, 1, 1, 1$. So it is going to be 15 and then unsigned fractional number, what I will be having it as smallest is 0, and then the largest value what I can have is $1, 1, 1, 1$ which is 0.9375.

When I come to the sign representation, so the first value is 0, so, all the 3 so, this is plus 7, and then the least negative value is $1, 0, 0, 0$ which is minus 8 and in the sign fractional the number representation we know that 0.1111 which is nothing but plus 0.875 will be the maximum value and then least negative value 1.00 which is minus 1 in this case, we have taken number of bits equal to 4.

So, you can increase it to second bit yours sorry 16 bit and see what will be the minimum and then maximum value of what you can represent with this you can look at it. So, the minimum value will be 0 and then the maximum value will be 65536 in this case 2^{16} what you will be having it 2^{16-1} will be the value and then in this case, it is going to be 7FFF positive number will be 32767 and negative number is going to be minus 32768. And similarly you can calculate in the unsigned and then signed values.

(Refer Slide Time: 29:36)



General Fixed-Point Representation

Dynamic Range and Precision of 16-Bit Numbers for Different Q Formats (Kuo & Gan)

Format	Largest positive value	Least negative value	Precision
Q0.15	0.999969482421875	-1	0.00003051757813
Q1.14	1.99993896484375	-2	0.00006103515625
Q2.13	3.9998779296875	-4	0.00012207031250
Q3.12	7.999755859375	-8	0.00024414062500
Q4.11	15.99951171875	-16	0.00048828125000
Q5.10	31.9990234375	-32	0.00097656250000
Q6.9	63.998046875	-64	0.00195312500000
Q7.8	127.99609375	-128	0.00390625000000
Q8.7	255.9921875	-256	0.00781250000000
Q9.6	511.984375	-512	0.01562500000000
Q10.5	1023.96875	-1024	0.03125000000000
Q11.4	2047.9375	-2048	0.06250000000000
Q12.3	4095.875	-4096	0.12500000000000
Q13.2	8191.75	-8192	0.25000000000000
Q14.1	16383.5	-16384	0.50000000000000
Q15.0	32767	-32768	1.00000000000000

21

So coming to the general fixed point representation. So we will be seeing what are the dynamic range and precision of 16 bit numbers for different Q format. This is also taken from Kuo and then Gan. So you will be seeing the first one as we said Q representation now will give me how many fractional bits represented to the right of it whatever represented here if I give 0.15.

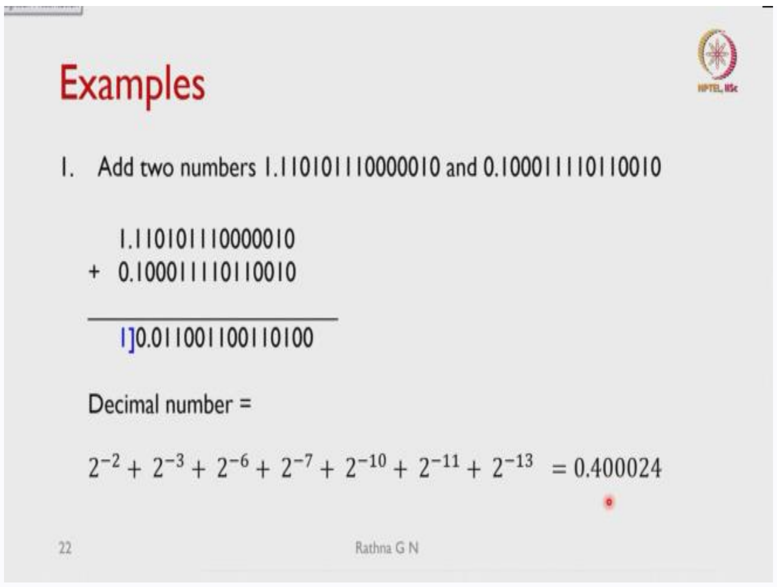
So, I know that all the 15 bits are represented as the this thing of fractional number, the maximum that is largest positive value, what I can represent is 0.999, what you will be going on and then representing it, and then the least negative value is going to be 1 and precision what I want to have it is represented with this number. Same thing, if I want to allocate 1 bit to the integer here, 0 bits I have included to the integer here I want to include 1 bit to the integer and 14 bits are for the fractional number, then my range largest value will be 1.999. 2, that is, least negative value I can have it is -2, and this is the precision what we will have. And then but I do not want to represent any fractional number, I want to have it as an integer, then I will be calling it as Q15.0, then I know that the 15 bits are represented as my fractional number, the value, what I will be representing is 327672 minus 32768.

So, the precision in this case is going to be 1.00. So it depends on what precision you want to have for your application. Based on it, you will be taking your number system, you may be wondering why you have to discuss about this number system. As you know that in the present scenario, FPGA is mostly used for most of the applications because of low power. So whether when I am writing my own code.

All of you know that in C you will be representing it as a hash define you will be doing it INT or float or double or whatever number but when you are using in the hardware, so you have to be careful whether I can save my resources, save my power. So that is what one has to look at it. So when you have to do that, you have to play around your fixed point representation of the number.

So that you know for what application you are using it, what is the dynamic range, I have not discussed it here in the dynamic range in this case so we will be discussing it later. And then what is the precision what I want to have.

(Refer Slide Time: 32:40)



Examples

1. Add two numbers 1.11010111000010 and 0.10001111011001

```
   1.11010111000010
 + 0.10001111011001
 ------------------
  1]0.01100110011010
```

Decimal number =

$2^{-2} + 2^{-3} + 2^{-6} + 2^{-7} + 2^{-10} + 2^{-11} + 2^{-13} = 0.400024$

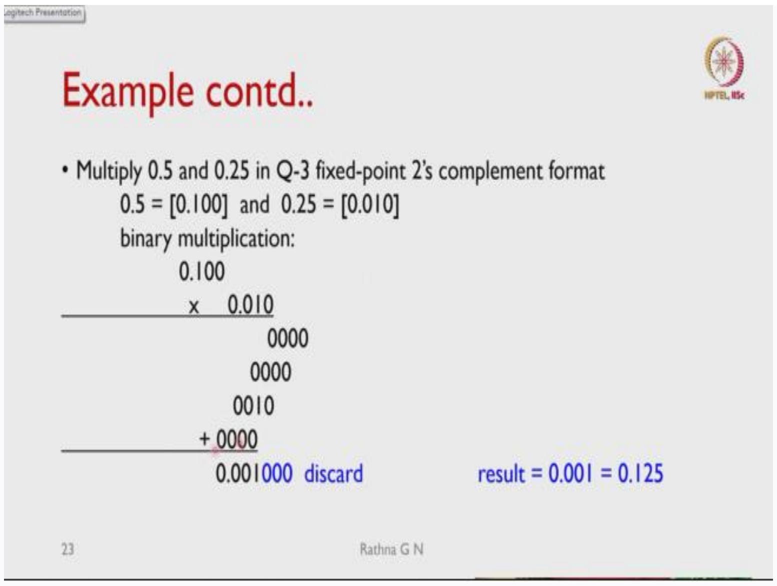
22 Rathna G N

So coming to a few examples, what we will take it how we are going to do all of you may be wondering why how to do addition of 2 numbers. So here I am going to do it in binary. So you will be seeing the binary number is given in this way. So when I add these 2 numbers, so what is the thing is going to happen, I will be getting you will be seeing that it will, if you add the thing just to show you that have you will be adding it, this will be from right to left, what the addition is going to happen?

This is 0, when I add 1 plus 1 zero, which will be carry and then you will be getting 1 here, and then you will be adding this further. So when you come to the last one, you will see that this is 1+1=0. And when I put 1 here, so it will be 1 plus 1 is going to be zero, and then I will be representing it as 1, 0 here, then what we do is as shown in the blue thing here, I will be discarding this number, whichever coming at the most place of the number.

So the decimal number, what I have represented in this case is because it is a positive number, and then 0, 1, this place is 2^{-1} I know and this is 2^{-2}. So I am taking wherever 1 is occurring only those values, I am putting it here. So you will see that $2^{-2} = 1$ and next is 2^{-3}, and next will be 2^{-6}, 2^{-7}. And then the last one will be your 2^{-10} 2^{-11}, and then the 2^{-13} . So when I add this number, the value in decimal, what I will be achieving is 0.400024. So, this is how the addition is going to happen in our hardware.

(Refer Slide Time: 34:42)



Example contd..

- Multiply 0.5 and 0.25 in Q-3 fixed-point 2's complement format

```
        0.5 = [0.100]  and  0.25 = [0.010]
        binary multiplication:
                0.100
              x   0.010
        ----------------
                    0000
                   0000
                  0010
              + 0000
        ----------------
               0.001000  discard          result = 0.001 = 0.125
```

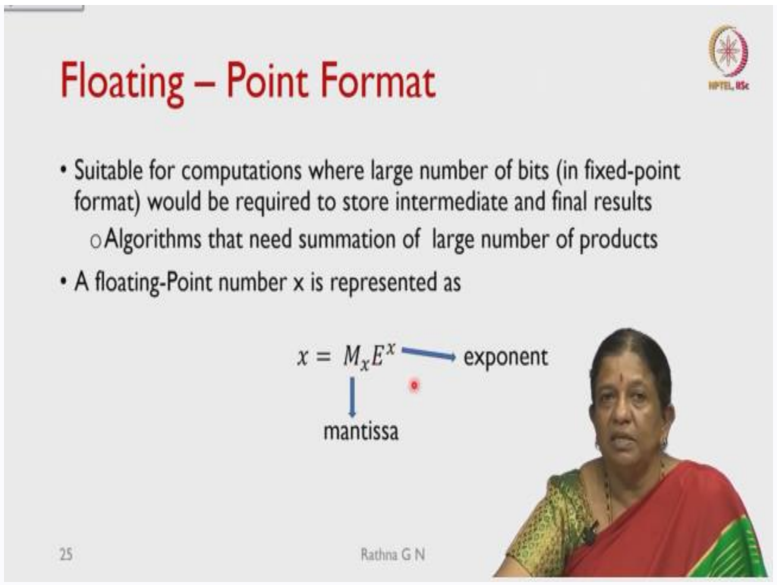
23 Rathna G N

And coming to the multiplication, what is the thing is going to happen? So I have to multiply 0.5 and 0.25, what I say is Q3 format, so that means to say that I have 4 bits, it is in the signed format, what I am going to do the multiplication, so 1 bit for sign and 3 bits are going to be for my fraction. So 0.5 I know that 2^{-1} is 0.5, so the other 2 numbers are 00 and 0.25, 0.010. What I will be considering this is $2^{-2} = \frac{1}{4}$ which is 0.25.

So I want to do this multiplication, I have taken a simple so that I am not doing any truncation or rounding in this case. So, which can be represented in binary format that value what I have taken the thing. So I will be multiplying 0.100 into 0.010. So we will be bit by bit multiplication is going to happen. So, you will be seeing that one after the other, then what is the how we are going to calculate the final value all of you know that it is going to be numbers have to be added that is what we have been taught in our primary also.

Then we will be adding it and then what we are going to do is because my format what I have input and output is in Q3 format, only I can represent 4 bits, the LSB bits I will be discarding which is shown in blue here, all three zeros are going to be discarded result is going to be 0.001 which is nothing but 0.125. So, you will be seeing 0.5 multiplied by 0.25 will be resulting in 0.125 in decimal. So, you will be seeing that how fast the calculator or your computer gives, but what goes on is this one inside your hardware. So, how this can be done faster and other things we will see in the architecture.

(Refer Slide Time: 36:55)



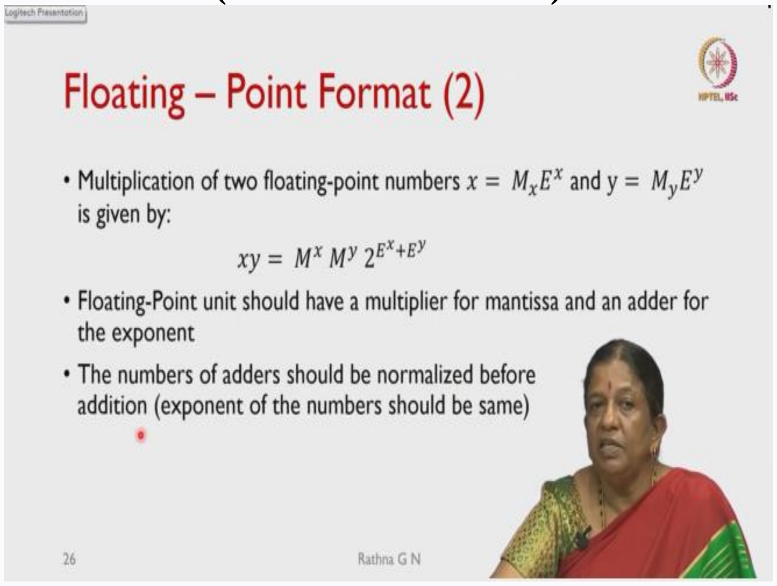
Floating – Point Format

- Suitable for computations where large number of bits (in fixed-point format) would be required to store intermediate and final results
  - Algorithms that need summation of large number of products
- A floating-Point number x is represented as

$$x = M_x E^x$$

So, now, what we say is what is this point format and then size what we have to look at it. So, which format will compromise between precision, overflow and then storage needs. So, one method we said we can increase the number of bits, the other way of doing it is the floating point number which will give me all this solution. So, how it is going to give me the thing. So, we will say that suitable for computations where large number of bits and fixed point format would be required to store intermediate and final results and the algorithms that means summation of large number of products, in that case we will be using the floating point number. So, if x is going to be represented in floating point format as $x = M_x E^x$ so, this we call it as mantissa. And this we call it as exponent this is a non standard format, we will see the IEEE format what it is the standard has defined.

(Refer Slide Time: 38:04)



Floating – Point Format (2)

- Multiplication of two floating-point numbers $x = M_x E^x$ and $y = M_y E^y$ is given by:

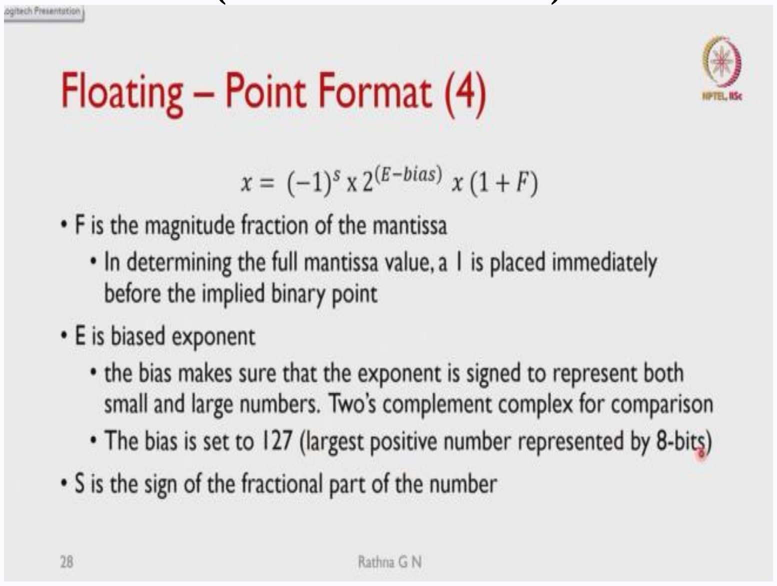
$$xy = M^x M^y 2^{E^x + E^y}$$

- Floating-Point unit should have a multiplier for mantissa and an adder for the exponent
- The numbers of adders should be normalized before addition (exponent of the numbers should be same)

So, in this case when I do the multiplication I have 2 plots floating point numbers, we know that it is $M_x E^x$and then y is given by $M^y \cdot E^y$. So, we know that we multiply the mantissa and then add the exponents to get the result xy. So, floating point unit should have as you can see, to do the multiplication and do the addition for the multiplier for the mantissa and an adder for your exponent this is the architecture definition what we are getting from the floating point.

So the numbers of adders should be normalised before addition that is the exponent of the number should be same for adding it is not as simple as my multiplication.

(Refer Slide Time: 38:55)



Floating – Point Format (4)

$$x = (-1)^s \text{ x } 2^{(E-bias)} \ x\,(1+F)$$

- F is the magnitude fraction of the mantissa
  - In determining the full mantissa value, a 1 is placed immediately before the implied binary point
- E is biased exponent
  - the bias makes sure that the exponent is signed to represent both small and large numbers. Two's complement complex for comparison
  - The bias is set to 127 (largest positive number represented by 8-bits)
- S is the sign of the fractional part of the number

28 Rathna G N

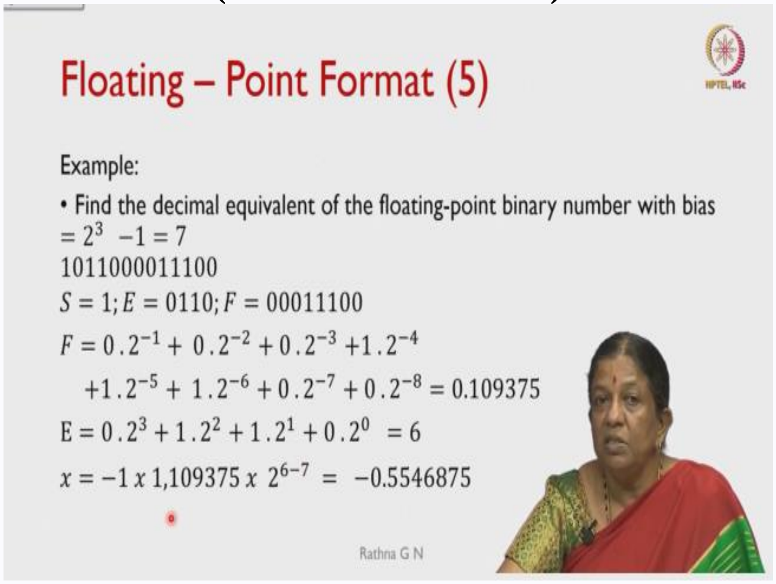
So, when I come to IEEE format representation, we call it as IEEE 754 which is defined in 1985 this thing format for single precision that is 32 bit floating point that number is represented as $x = (-1)^s \text{ x } 2^{(E-bias)} \ x\,(1+F)$. So here s, E and F are all unsigned fixed point format, s will be representing your sign bit, E will be representing exponent biased.

And then F is the significant in this case and we say that this is the implied binary point here also will not be allocating any bits to the decimal point. So you will be seeing that the significant is represented with 23 bits, whereas the exponent is with 8 bits and sign bit is with 1 bit. If it is 0, it is going to be positive, significant and if it is 1, negative significant. So, you will be now wondering why exponent is biased because why I can explain it also can represent it in positive and negative.

So I am not allocating any bit for the thing. So we will be doing the biasing so that to avoid this positive or negative number representation for the exponent. So we will see as this thing example here, which is nothing but $(-1)^s \text{ x } 2^{(E-bias)} \ x\,(1+F)$. So we know that F is the magnitude fraction of the mantissa. So in determining the full mantissa value.

So IEEE format specified that 1 is placed immediately before the implied binary point, 1 dot format, that is where you will be adding 1 to the whatever you have the fraction here. So, E is the biased exponent, so, the biased makes sure that the exponent is signed to represent both the small and then large numbers. So 2's complement complex for comparison. So the bias is set to be 127, that is largest positive number represented by 8 bits with which is equal to 2^{8-1}. S is the sign of the fractional part of the number.

(Refer Slide Time: 41:31)



Floating – Point Format (5)

Example:

- Find the decimal equivalent of the floating-point binary number with bias $= 2^3 \ -1 = 7$

1011000011100

$S = 1; E = 0110; F = 00011100$

$F = 0\,.\,2^{-1} +\ 0\,.\,2^{-2} + 0\,.\,2^{-3} + 1\,.\,2^{-4}$
$\quad + 1\,.\,2^{-5} +\ 1\,.\,2^{-6} + 0\,.\,2^{-7} + 0\,.\,2^{-8} = 0.109375$

$\mathrm{E} = 0\,.\,2^3 + 1\,.\,2^2 + 1\,.\,2^1 + 0\,.\,2^0 \ = 6$

$x = -1\, x\, 1{,}109375\, x\ 2^{6-7} \ = \ -0.5546875$

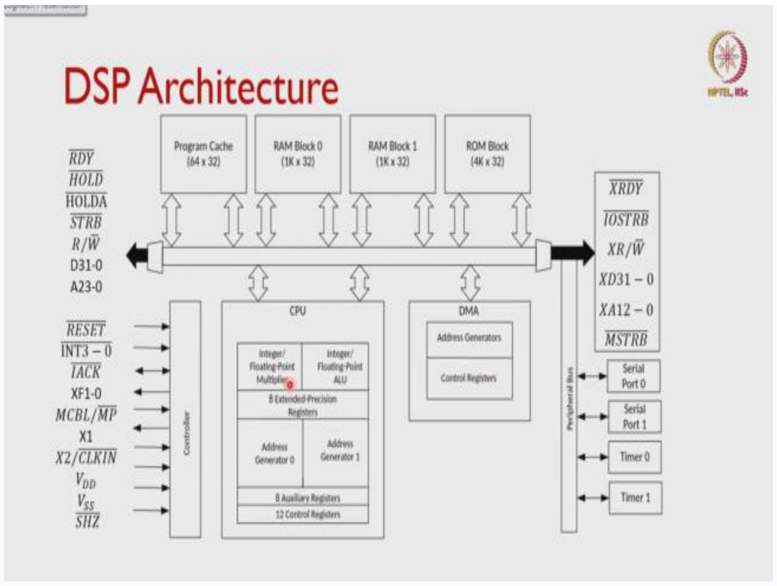
Rathna G N

So as an example, how we are going to represent we will see the thing find the decimal equivalent of the floating point binary number with bias$= 2^3 \ - 1 = 7$. So you will be seeing this is the number what you have been given in binary format. So we know that sign bit is 1 in this case, and buyers have been given as 7, so I will be using 4 bits for my exponent, so which is 0110, and rest of them are fractional bit.

So which is nothing but 00011100. So when I represent it fraction part $0\,.\ 2^{-1}$, and then all the 3 of them are 0, and the next number it will be $1\,.\ 2^{-4}$, and then you will be seeing 1 and rest of them 0, which is nothing but 0.109375 is F value. And then experiment we will see the thing, $0\,.\ 2^3$ and then $1\,.\ 2^2$, and then $+\,1\,.\ 2^1$ and $0\,.\ 2^0$, which is equivalent to 6.

So when I put it in its format, we know that sign is 1 so it is a negative number -1 into what we have is because $F\ +\ 1$ what I have to do it, so it is 1.109375 x $2^{(E-bias)}$, bias we have is 7, 6-1=2^{-1}. So this is divided by 2, which is equivalent to minus 0.5546875 is the value of what I can represent.

(Refer Slide Time: 43:10)

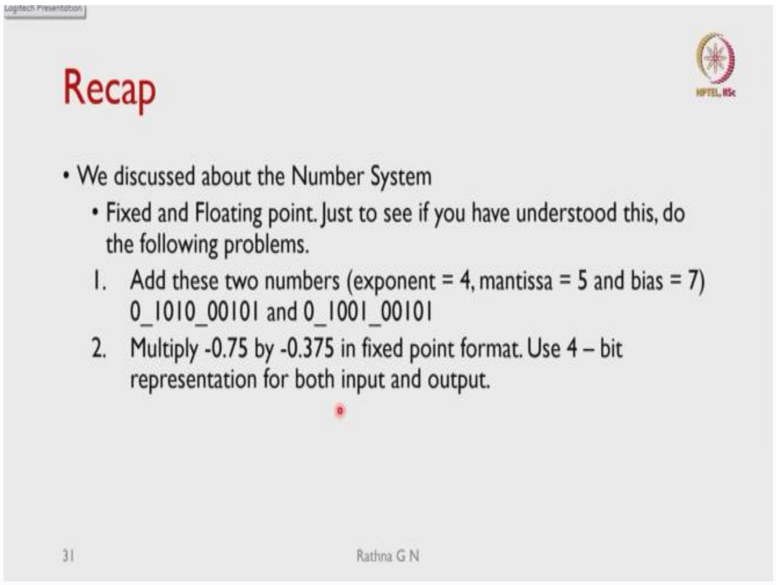


So we will see a little bit on the DSP architecture and we will be taking detail one in the next class. So, what we have is, as you can see here, so I have for the floating point DSK 6713 processor what I have considered here, so you have the integer or floating point multiplier here and then I have an integer floating point ALU addition and then we have some extended precision registers and we have some address generator 0 and 1 in this and then we have some auxiliary registers and some control registers.

So, these are the compiler what you would be feeding in and this is the data bus, which is data bus is 32 bit long, and then the address bus is 24 bit and then we have something we call it as programme cache and this is the RAM what we have it which is 1K into 32 bit each. So, RAM block 1 what we have it 1K into 32 bit here and then the ROM for storing our coefficients and everything which is 4K into 32 bit.

And some of the peripherals what you will be seeing connected here through the peripheral bus and then we have the like any other microprocessor 8 if I would have studied the thing, I have the DMA controller, which will be generating address generator and control registers. So directly I will be able to take from the memory and then load into the hardware.

(Refer Slide Time: 44:57)



Logitech Presentation

Recap

- We discussed about the Number System
  - Fixed and Floating point. Just to see if you have understood this, do the following problems.
    1. Add these two numbers (exponent = 4, mantissa = 5 and bias = 7) 0_1010_00101 and 0_1001_00101
    2. Multiply -0.75 by -0.375 in fixed point format. Use 4 – bit representation for both input and output.

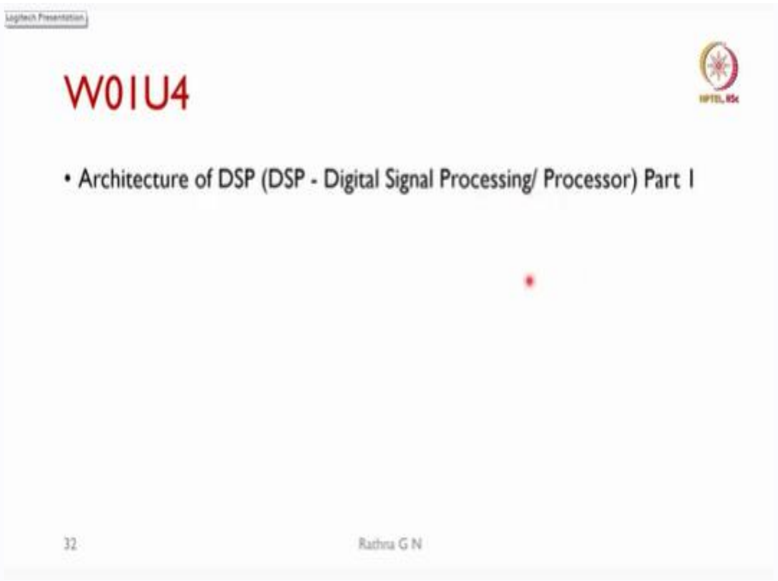
31 Rathna G N

So, coming to recap, what we have done in this class is, we discussed about the number system. So to see that whether you have understood or not the fixed and floating point. So I have given 2 assignments here, please work it out. And then I will be showing you how we can compute this one. First one is add these 2 numbers, here it is non IEEE standard format what I have given the thing, exponent = 4, mantissa = 5 and bias = 7.

So these are the 2 numbers one has to add. So please keep it in mind, we did the multiplication, it was very simple. We have multiplied mantissa and then added the exponent. In this case you will be seeing that the exponents are different, you have to bring it to one format. I will see that which format you will choose it, whether the higher one or the lower one. And then we will discuss why which one is better to choose the thing in that next class the second assignment what I have given is minus 0.75 by minus 0.375.

So, we did the simple positive number multiplication. So, I want to it is going to be a trick to do multiplication of 2 negative numbers I know the result is going to be positive. So, you can trick me by removing the minus sign, but I want you to keep this minus sign and then do in the 2's complement, so that using the 4 bit for both input and output and what will be the result whether it is going to be correct or not, what I want to you to try it out so that you can play with me in the number system in this fashion.

(Refer Slide Time: 46:52)



Logitech Presentation

W01U4

- Architecture of DSP (DSP - Digital Signal Processing/ Processor) Part 1

32 Rathna G N

So with that we will end this today's class. So in the next class, we will be taking architecture of DSP. That is we will be using DSP synonymously for signal processing, as well as processor, Part 1, what we will be taking up in the next class because this number system will help us to see that what should be our architecture for designing it, although I have shown you one DSP architecture, we will see how we will be going about to build it. So those who are interested in building on their own DSP processor, you can use FPGA to build it. Thank you.