

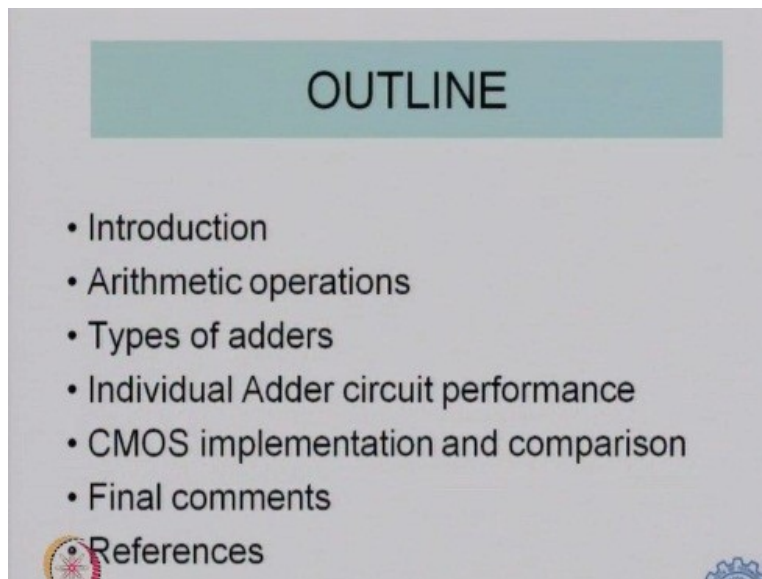
**Advanced VLSI Design**  
**Prof. A. N. Chandorkar**  
**Department of Electrical Engineering**  
**Indian Institute of Technology – Bombay**

**Lecture -10**  
**Arithmetic Implementation Strategies for VLSI**

Hello, welcome to another lecture in the area of VLSI design. I am Chandorkar once again. Today we start with the major activity of VLSI implementation namely we will talk about arithmetic implementation and please remember most of the processors most of the signal processing requirements will require a large amount of arithmetic processing and therefore this circuits are if great relevance.

My talk will this one hour or maybe little bit more I will talk about arithmetic in some form of-- this interaction, I will give some arithmetic operations.

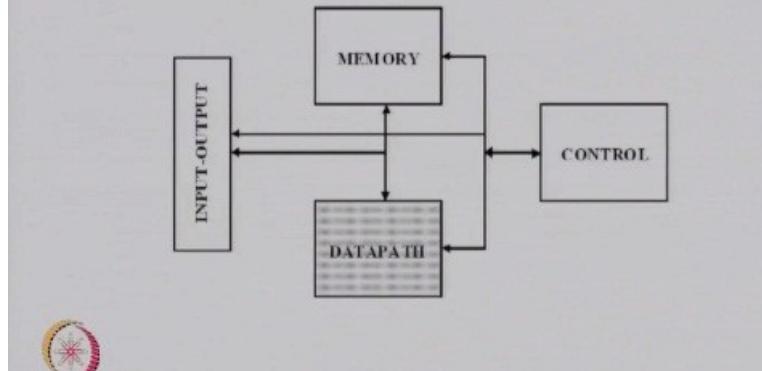
**(Refer Slide Time: 01:01)**



We will talk about adders and then we will talk about individual Adder performance. Then there CMOS implementation comparison and maybe few comments. And in my next talk maybe I will then go over to multipliers. Till then let us start with the something to do with adder operations which are most important. The types of Adder let us start with the basic arithmetic which is used in all computing systems.

**(Refer Slide Time: 01:26)**

## A Generic Digital Processor



A typical digital processor may be called microprocessor or a DSP processor whichever we look at it has-- this is typical structure or architecture one does have. You have an input data input/output port here; data is inputted from here. Then there is a components there is system parts system part which call DATAPATH and this is what under what the current today's lecture we are going to talking about DataPaths.

Then there is a memory which exchanges between Datapath as well as it is controlled by controller. Some kind of a logic is build here and it allows you to do some operations in this datapath to be stored here or return back to path itself to datapath once again to do further recursive kind of operations and then you may output it out. So this is called generic system and in this, this datapath is essentially is nothing but the arithmetic part which allows every other data to be persists in this processor part. So let us see into this DataPath.

**(Refer Slide Time: 02:38)**

## Building Blocks for Digital Architectures

### Arithmetic unit

- Bit-sliced datapath (adder, multiplier, shifter, comparator, etc.)

### Memory

- RAM, ROM, Buffers, Shift registers

### Control

- Finite state machine (PLA, random logic.)
- Counters

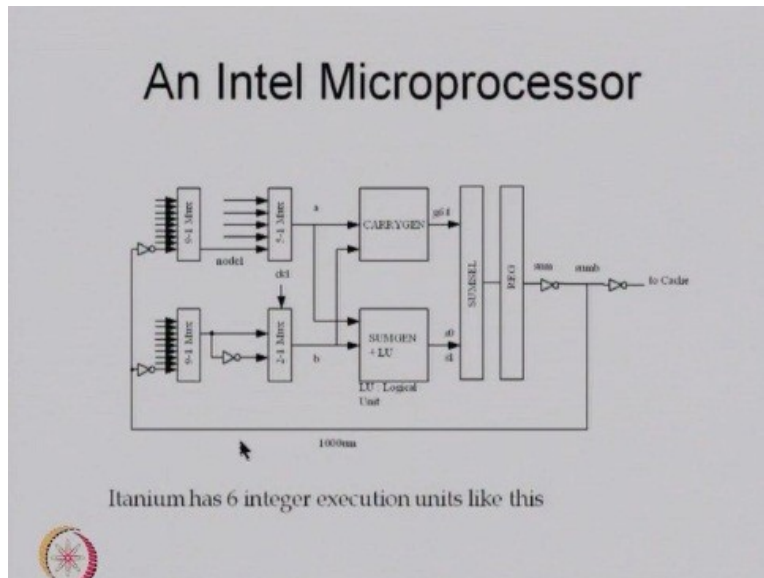
### Interconnect

- Switches
- Arbiters
- Bus

A typical digital architectural building blocks which are shown there, I repeat is an arithmetic unit which may be a Bit-sliced datapath which may have adder, multiplier, shifter, comparator and what not. Then there will be memory elements like RAM, ROM, Buffers, Shift registers. And the control will be some kind of a logic as I said it may be random logic or based on PLA, essentially implementing a Finite state machine.

And you may require timing there because of the data has to flow in the given time so you may have counters settings in the control block. And the finally if not the importance shown here in the last figure but there you can see there are Switches, Arbiters and Buses which actually constituted rather than the smaller part the larger part in the whole digital architecture.

**(Refer Slide Time: 03:34)**



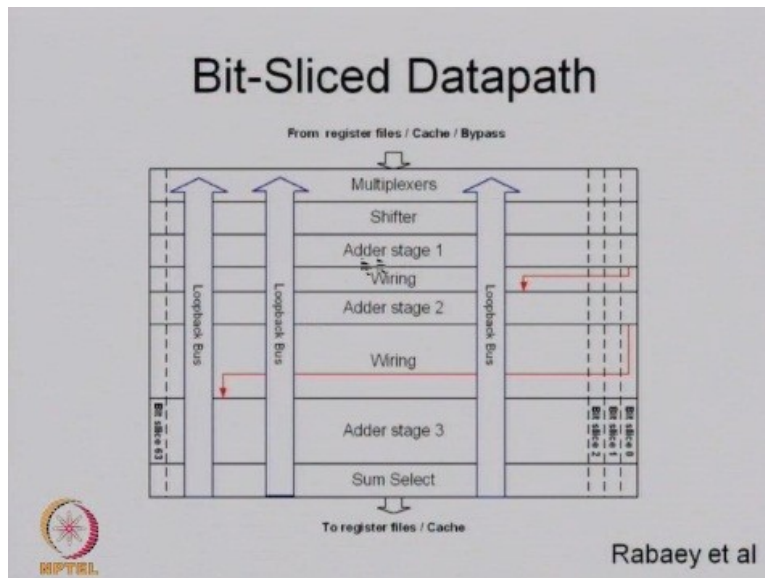
A typical Intel microprocessor is shown here, you know you can see here I just want to show here there is a these are different block shown here and the data is arriving from this is the input data line, this is your output data line may go to Cache and then there will be some kind of these two blocks which essentially do the maths, this is what the control is does there are Bit-slices. And after all these depending on the logical operation you want to do.

You can actually do further ahead there is a register here, there is a buffers here and finally you have some kind of an output. This is—Itanium has 6 integer execution units like this and it is interesting to see why they are so fast. A typical processor or arithmetic block which I was talking about has it is called Bit-Sliced; these are Bit-sliced each is slice of that; you can say it is a Bit 0, Bit 1, Bit 2, Bit 3 and Bit n.

And each is processes simultaneously and/or at time serially depending on the architecture you are, you have register block here you have adder block, you have shifted block in your multiplier block, so you get Data-in, first is Registered in; then during the control it gives data to the adder and once the addition operation is done or multiplication operation is done depending on. Sometimes you can see adder can be help by doing an operation called shift add shift which may add to become a multiplier as well.

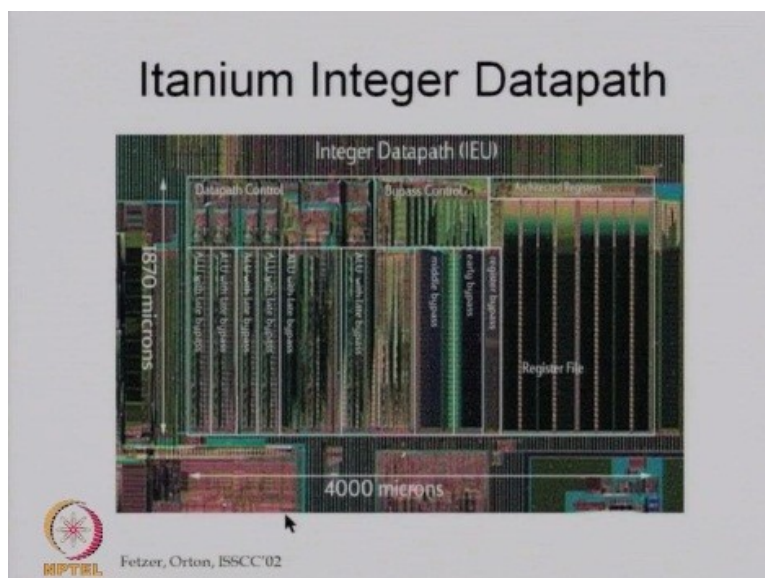
So we will see that these are the processing elements, I repeat in a normal arithmetic you require register and adder, a shifter and a multiplier. Please remember this most of the sliced have been taken from Rabaey's book and I wish that they will not have any objection because this is just to show you the figure which is nothing specific.

**(Refer Slide Time: 05:31)**



The Bit-Sliced Datapath which is shown here is again you can see from here, there are number of multiplexer, shifter, adder stage one and two, and there are wirings to connect this okay and this is your number of bits coming logical systems and this is how the data actually is flowing from input with the output and therefore it is called Datapath.

**(Refer Slide Time: 05:55)**



A typical Itanium Integer Datapath is shown here. This is roughly 4000 microns. So you can see from here integer datapath I is shown here; this is a digital control, datapath control, this is also a bypass control. Since the other things and these are Registered Files. So one can see that any of the processor part will require lot of data processing which is essentially arithmetic.

The major application of Datapath number of datapath blocks require in any processor application will be larger in the case of DSP, Digital Signal Processor and why because they are becoming most important elements in most of the systems which we often use.


**(Refer Slide Time: 06:47)**

**DSP Applications**

1. Communications- 3G (4G!!), Wireless, RADARs etc.
2. Consumer Applications.
3. Robotics.
4. Speech, Music, Audio, Video Processing applications.

Issues....

- ☐ Battery size,
- ☐ Programmability,
- ☐ Parallel architectures,
- ☐ Higher Clock rates.

 **Lars Wanhammar**

For example, in communications either your 3G or 4G or in wireless or any RADARs you will require lot of DSP processing it maybe data processing, it maybe a voice processing or it may be a video processing but you will require lot of data processing in this case. In the consumer applications, all of you are using mobiles and what not TVs and everything you require lot of processing these days and therefore most of the consumer applications we need DSP system.

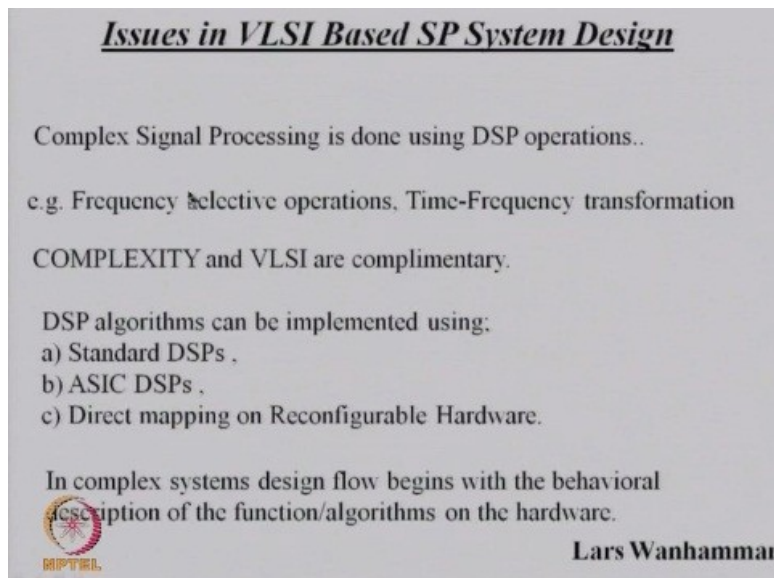
Robotics is another area which is very common and becoming very important for integrator manufacturing in many big large industries. And finally as I said in the Video, Audio, or Music system or speech processing you require a DSP. So what are the issues when I design a DSP processor essentially Digital Signal Processor, one is most of the case they maybe handle systems. So there may be an issue of battery size.

Because that is availability space as well as the weight it has to carry. Then for example it may all signal processing applications will require programmability which is in contrast in normal VLSI system where you may have other parameters control but not necessarily programmability, whereas in the signal processing or any other microprocessor applications programmability is the dominant requirement of any such processor designs.

Parallel architectures there are maybe you know, whether to reduce whether to improve the speed and reduce power there may be many parallel architectures build around there where the processor is designed and fabricated. And of course one is expecting that should run in very high speed typically gigahertz rate is what is expected and therefore these are the applications for these kind of application these are the issues which one has to take care when one designs a DSP processor.

If you are looking using a-- let say this signal processing system may have large amount of circuit tree or blocks so you will like to implement on silicon for example so we say what are the issues in the VLSI implements base system design.

**(Refer Slide Time: 09:02)**



**Issues in VLSI Based SP System Design**

Complex Signal Processing is done using DSP operations..


e.g. Frequency selective operations, Time-Frequency transformation

COMPLEXITY and VLSI are complimentary.

DSP algorithms can be implemented using:

- a) Standard DSPs ,
- b) ASIC DSPs ,
- c) Direct mapping on Reconfigurable Hardware.

In complex systems design flow begins with the behavioral description of the function/algorithms on the hardware.

 **Lars Wanhammar**

So here are some things which mean we are very worried about. Once is complex signal processing is done using DSP operations, huge complexity of operation is very important-- large

numbers as well as complex operations are done. There would be a frequency selective operations, Time-Frequency transformation and therefore it may lead to large complexity and therefore VLSI are also-- circuit will also be very complex.

Then in the case of DSP algorithms can be implemented using Standard DSPs, ASIC DSPs or Direct mapping on Reconfigurable Hardware. In complex design flow, begins with the behavioral description of the function/algorithm on the hardware itself, like you can use a (()) (09:41) itself, implement your DSP processor, of course in turn you may get-- you may not reach very high speeds at least but functionally it may be working relatively sufficient speeds.

So if you look at the VLSI and a processor design, you know one case see we have seen already in the case of VLSI chip design.


**(Refer Slide Time: 10:05)**

**VLSI and Processor**

VLSI specifications are (Area- Delay- Power- Speed)  
Vs  
Programmability of Digital Processor

Best VLSI designs are OPTIMAL and therefore they are specific.  
Programmability must allow flexibility and therefore concept of Generalized architectures and hence NON-OPTIMAL.

*We take Middle Path.....*

 NPTEL

Lars Wanhammar

Typical specification, there we are worried about area, we are worried about speed or delay in speed of source delay and power and of course in synchronized systems speed may not be directly involved were some of the delays which you are controlling. Whereas in the case of digital processor design or signal processing designs major requirement is programmability. Now best VLSI designs are OPTIMAL and therefore they are specific.

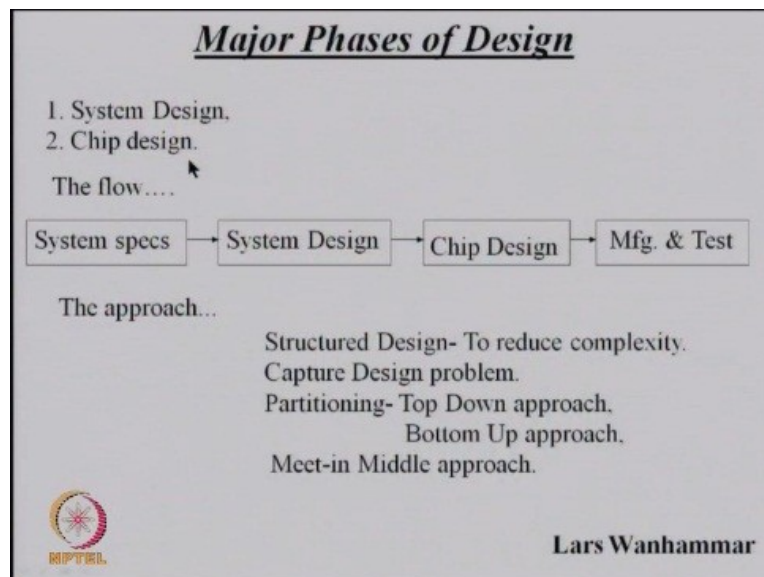


Once you say you are optimal means they are specific. Once you want to implement a digital processor which has programmability because it should allow you flexibility and therefore the concept of generalized architecture and hence NON-OPTIMAL. Now, one can see if you have a good chip design which is hard completely dedicated hardware there I can design a very good chip a very good VLSI design VLSI IC which is optimally performing the desired specification.

Whereas in the case of programmability one does not know what exactly has to be done every now and then same processor will be used for multiple applications and in that case and different kinds of data may also be appear and because of that programmability is very, very relevant for the case of DSPs. And since they are-- you cannot use the specific architectures this is this must have generalized architecture.

And obviously such circuits or such blocks may certainly NON-OPTIMAL. So how do we go about so it said that if you are implementing a high speed some kind of a signal processing then you should take a middle path look for this and this middle of them is ideal. So typically what are the phases in design of such signal processing systems?

**(Refer Slide Time: 11:58)**

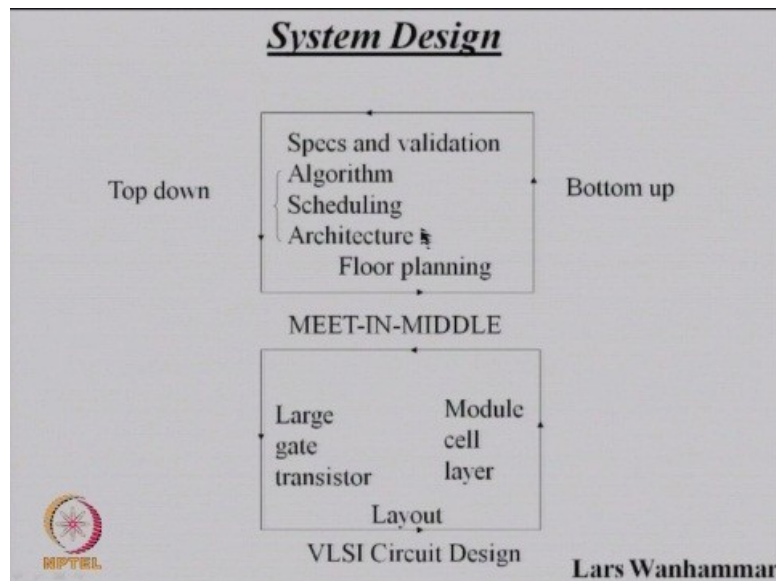


We have a System Design and we have a Chip Design. So you start with, this is the kind of flow you may have is System Specification, then you design system on the behavioral, structural as well as (( )) (12:09) level and from there you may design a chip okay down a physical design can

be use or can use AISC can be used or programmable chip can also be used on which you can put your design. And finally once they are manufactured you should be able to test them.

The approach as I say is structured design- To reduce complexity, capture design problem, partitioning- top down approach, bottom up approach one has to decide which way to go about and Meet-in Middle approach is what basically we will like to do.

**(Refer Slide Time: 12:42)**

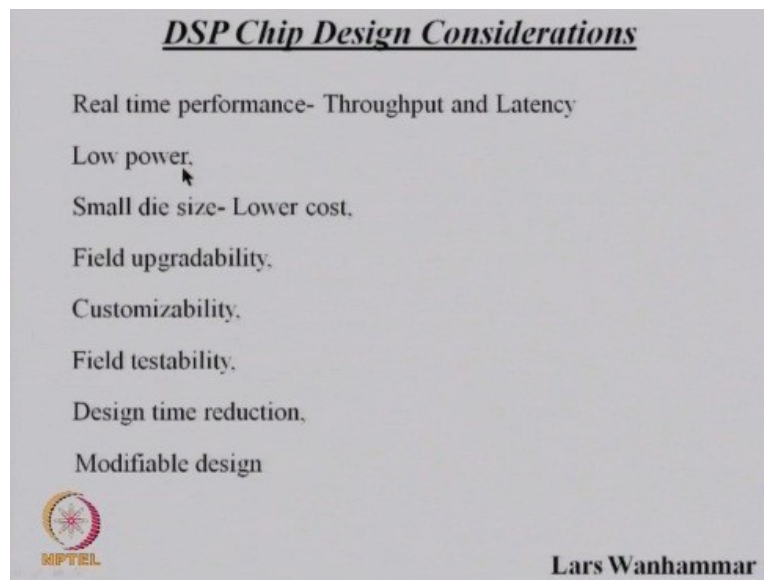


So here is the first part is the Signal Processing part, the other is of course a VLSI part in some way, so if you are doing a top down design you start with Algorithm, Scheduling Architecture, Floor planning, and if you are designing the bottom of kind of situation then you say a Large gate transistor, Module, cell, layers and layout. So in between what you do is you get some kind of a pre-design blocks or some kind of a IPs available to you whose layout are tested.

And you know about the performance maybe use them to implement these algorithms using standard scheduling for the given specification of the system. And design take architecture through which use some of the already pre-design blocks and do designs something additionally if you need. So this is one Top Down and this is we call a Bottom Up applications and this is most cases any system design will actually use this approach.

So if you are really looking for DSP designs then real-time performance throughput and latency are the two major criteria of any chip design, DSP chip design because it is a real-time performance and given how long what is the throughput that is data which will available per unit time or something. And how long this data can be made available after inputs are impressed is called the latency, so these two issues are very crucial in design.

**(Refer Slide Time: 14:13)**



Then the next and the most-- these days a foremost among them of course DSP will require all these but the major chip design requirement will come from the Low power. We also require small die size - Lower cost, we want field upgradability. I should be able to modify online on chip that a programmability should be good and you may be able to add additional blocks that is pre-design blocks maybe available to you and you can add upgradable to the earlier versions.

It should be customizability that you know for a given application you should be able to convert to customer's application. Then it should be testable in the field because if it is testable only at the (( )) (14:54) end then it will be very difficult to modify. So it should be field testable and obvious reason to say the money the lower cost design time should be very small comparatively and of course they should be modifiable designs.

**(Refer Slide Time: 15:10)**

## *Rabaey's Rules*

**RR1** : The 'RIGHT' structure has to be chosen for the functional unit before attempting optimization .

≈ ≈

**RR2** : The 'CRITICAL PATH' of the circuit should be identified and its length should be minimized.

**RR3** : Only the number of the transistors dose not decide 'CIRCUIT SIZE', indeed Interconnects are decisive for area.



There are three Rabaey's rules which you should follow in any chip design. 1. The RIGHT structure has to be chosen for the functional unit before attempting optimization. 2. The CRITICAL PATH of the circuit should be identified and its length should be minimized. First thing you see that critical path we know all know is the path from input-output which has the largest delay. So first look at it can we minimize by actually connecting or by algorithm or by this whether critical path lengths are smaller.

So you can have larger speed. And the finally, only the number of transistors does not decide circuit size it is indeed the Interconnects and are decisive for area as well as for you know - For example, you may say that I may have million transistors – or 100 million transistors, but you will not believe that the area of 100 million transistor will be less than 30% many times on a chip whereas the other 60% area maybe actually taken care are taken by interconnects.

And therefore in deciding the size of transistors is not the only current area of the chip size you are going to have but also requirement of interconnect is also going to decide. Now we start with our arithmetic. Today, we will look into variety of number systems, I have been giving you some introduction, let us do some basic thinking once again, though all of us know about it in many ways but we will just go through if not very detail way but at least go through the basic arithmetic requirements.

**(Refer Slide Time: 16:55)**

## Number System

### Definition :

A number system can be defined by the set of values that each digit can assume and by interpretation rule that defines *mapping of sequence* of digits and numerical values.

For example : Number  $X$  sequence  $[x_0, x_1, \dots, x_{w_d-1}]$ , has a unique representation as;

$$X = \sum_{i=0}^{w_d-1} w_i x_i$$

Here  $w_d$  is word length and

$w_i$  is weight associated with digit.

Conventional :  $i^{\text{th}}$  power of fixed integer 'r' (radix),

$$w_i = r^i$$



Lars Wanhammar

So we start with number, a number system can be defined by the set of values that each digit can assume and by interpretation rule that defines mapping of sequence of digits and numerical values. This is slightly a definition which looks very funny but at least if you read, you do not make really (()) (17:10) but if you see in real life without much worry about you can define a number correctly.

A number  $X$  for example, which are the sequence of  $X_0, X_1, \dots, X_{w_d-1}$  then we say  $X = \text{sum of } i = 0 \text{ to } w_d - 1, w_d$  is called the what length and is some of  $w_i$   $x_i$  is associated weight and  $x_i$  is essentially the bit at position of the bit. Here conventional  $i$ -th power of fixed integer 'r' called radix one can say omega  $i$  is art of the power  $i$ . Now this statement if you all know very well is not that difficult to understand.

**(Refer Slide Time: 18:08)**

$$178 = 1 \times 10^2 + 7 \times 10^1 + 8 \times 10^0$$

$$\underline{r = 10}$$
 Binary  $r = 2$   

$$(1101)_2 \Rightarrow 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (13)_{10}$$

If you see it you take it, take a decimal number and you say you have a number 178, so how do you actually think it is 178 so the way we write 100 into 10 to the power of 2 that is 100 + 70, 7 into 10 to the power 1 is 70 + 8 that is 8 10 to the power 0 is what, so we say here the radix which essentially 10, okay. I is now varying I is 0,1, 2 okay. And then the  $W_i$  can be written in this form, so a width of the word length is 3 bits, here 178.

But essentially it can be represented as  $r$  of course as I say is 10 so it is a decimal number. In the case of binary, what we say earlier we know about it in the case of binary radix is 2 and in hexagon it is 16; octal it is 8, so all that we are saying is the radix is for the 6 integer radix are to the power whatever we write we can get each bit and for that if we add them together as we wrote here in the case of this add all of them this is  $W_3 X_3 + W_2 X_2 + W_1 X_1 + W_0 X_0$  up to the word length.

Then we can get the net  $x$ , this is what the 10 digit of sorry the decimal number can look into. Similarly, one write—if I write 1101 4-bit number which essentially what I am saying this is one into 2 to the power 0, in the case of binary then 0 into 2 to the power 1 + 1 into 2 to the power 2 + 1 into 2 to the power 4 so one can see here the number in decimal essentially is sorry I am very sorry 2 to the power 3, so this number is 8+2; 12+1 13 so this is 13 digits; this is say 2 and is equal to 13 in 10.


So what is essentially trying to tell that if I know my  $r$ ; I can always represent any number in terms of  $W_i$  and therefore some of all such  $W_i X_i$  can give me up to the word length and generate my numbers.

(Refer Slide Time: 20:39)

**Fractional Fixed Point Arithmetic**

...is used in most of the Processor applications for calculations.

1. Signed magnitude :  $X = (1 - 2x_0) \sum_{i=1}^{w_d-1} x_i 2^{-i}$   
→ Depends upon Signs of the operand!!
2. 1's complement :  $X = -x_0(1 - Q) + \sum_{i=1}^{w_d-1} x_i 2^{-i}$  , where  $Q = 2^{-(w_d-1)}$   
→ Multiplication is more difficult!!
3. 2's complement :  $X = -x_0 + \sum_{i=1}^{w_d-1} x_i 2^{-i}$  , where  $Q = 2^{-w_d-1}$   
→ Mostly used.  
 3's complement is a crowd....so no more!!
4. Binary offset :  $X = (x_0 - 1) + \sum_{i=1}^{w_d-1} x_i 2^{-i}$


Lars Wanhammar

Now there are apart from the integer numbers there will be some fractional fixed point arithmetic is used in most of the processor application for calculations. Now for example there are number of ways in this numbers can be represented one of course is called Signed magnitude number. This is of course possibly we are talking about binaries but true for all but I now, this is now only showing binary number.

So we say number  $X$  is equal to  $1 - 2x_0$  summation of  $i=1$   $W_d-1$  which is  $i 2$  to the power  $-i$  so what we have done is essentially we said the first the last all the bits are kept as fractional power, so rest power is kept as it is. But the first bit before the integer is essentially the sign bit. This of course depends on the science of the operand, we will show some numbers for that then the other most commonly used mathematically used binary number is called once compliment.

So there we said  $X$  is equal to  $-X_0, 1$  minus  $q_0$  plus sigma  $i=1$   $W_d - 1$   $x_i 2^{-i}$  where  $Q$  is  $2$  to the power  $-w_d-1$ . Now in this multiplication is more difficult in the case once this so we will then change over the 1's compliment to another compliment number.  $X$  is equal to  $-x_0$  now we do not

need to define Q here because we already equipped it is not required in this directly we can write  $-x_0$  plus  $i$  to the power  $-1$ .

Now by doing this what we did essentially we are added, oh sorry, the Q must be here, so what we are essentially saying by doing this what we have achieved is we will see this why complements are commonly used. It finds that these kinds of representation can done if we do then this can be used extensively in almost all arithmetic. There is of course a joke from the book Wanhammar book which says 3's compliment to the cloud and so no more.

Of course logically one can say you can have 3's compliment, 4's compliment and multiple compliments. And finally the version of compliment is also possible what we call Binary offset and we will see little later about— $x$  can be  $x = 0 - 1 i_1 W_d -1 x_i 2_i$ . So these are the four possible ways numbers can be represented and one normally use is 2's compliment, someone use binary offset rarely signed magnitude and very rarely 1's compliment.

Now why 2's compliment is so very commonly used the number if the some lies.

**(Refer Slide Time: 24:09)**


**Why 2's Complement**

→ If the sum lies in the proper range, several 2's complement numbers can be added even though partial sums may show "Overflow".

An Example: Assume we have.  $(1.110)_2, (0.100)_2$  &  $(1.001)_2$

$$S_2 = (1.110)_2 + (0.100)_2 = (\text{overflow}) + (1.010)_2 = -6/8$$

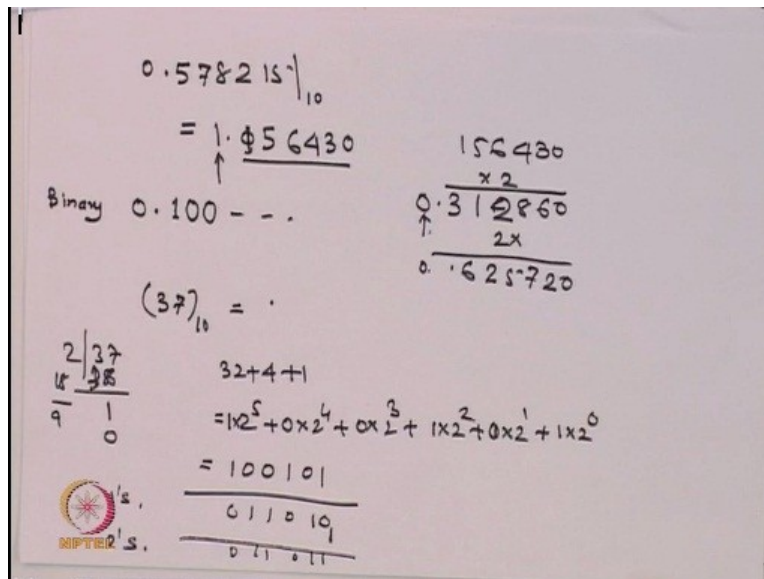
$$S_1 = (1.010)_2 + (1.001)_2 = (\text{overflow}) + (0.011)_2 = 3/8$$


Lars Wanhammar

I suppose all of you are aware and let say you have a number which is 1's compliment. I think you are fully aware the way we do 2's compliment and 1's compliment is that given a number for example, if you have a--.



(Refer Slide Time: 24:24)



Let us say it is a fractional number point 0.578215 any number we can take then if I want to represent this is your decimal number I want to represent this into binary number so first think I multiplied by 2, now this one is essentially the first bit will come 0.1 in binary. Okay. The next bit can be take this value 156430 multiplied by 2, so you get 68 12 11 and 3 and since this is 0, so the next bit is 0.

Now take this again to multiply 0 1 7 5 2 6 and since again there is a 0 here the next bit is 0, keep doing this till you achieve some number, equivalent number of binary. This is essentially we say we convert this into a normal binary. So if I want to make any binary number into a 2's compliment, all that we do is we add, we actually complement number whatever ones and zeros and ones we do that number is variable, then we actually add 1 and we get 2's compliment. Okay.

Or take simple number of say 37 in decimal, then it is essentially if I divided by 2, so let me divide 37 by 2 and let us see 18 sorry 36 so I get 18, so 1 then I keep dividing 18 so I get 9 so I get 0 and I keep doing this number so I say 37 is essentially  $32 + 4 + 1$  which 2 to the power 5 plus 0 into 2 to the power 4 plus zero into two to the power of 3 plus 1 into 2 to the power 2 plus 1 into 2 to the power sorry 0 into 2 to the power 1 plus 1 into 2 to the power 0.

So, the number is 100101 so this is your 37, so if I want to take the complement 1's complement just complement this 010110 is ones complement, add one so I get as 2's complement. Now this is very standard arithmetic, nothing great about the - only why I am actually showing just to show you this numbers was, if I have fractional numbers, you can do similarly by actually multiplying by two on the fraction side and divide on the integer side so you can always get this.


**(Refer Slide Time: 27:42)**

**Why 2's Complement**

→ If the sum lies in the proper range, several 2's complement numbers can be added even though partial sums may show "Overflow".

An Example: Assume we have,  $(1.110)_2$ ,  $(0.100)_2$  &  $(1.001)_2$

$$S_2 = (1.110)_2 + (0.100)_2 = (\text{overflow}) + (1.010)_2 = -6/8$$

$$S_1 = (1.010)_2 + (1.001)_2 = (\text{overflow}) + (0.011)_2 = 3/8$$


Lars Wanhammar

So let us see the example which I am showing here. You have 2 numbers to add, 3 numbers to add 1.101102, 0.1002 and 1.001 also 2 that is in binary. And let us say I add these two numbers, okay, 1.102 plus this. If I add this, one can see from here 0+0 is 0, fine, 0+1 is 1 so that is fine. 1+1 and there is the problem 1+1 is 0 but then there is a overflow has appeared and that is the additional bits have appeared.

Now, one can see from here, this one coming here essentially tells you it is a sign bit, so the number is minus, okay, because this number was minus so always this product add is also minus because this is larger than this. So if so the net number is minus and you can see from here this is 2 to the power -1, 2 to the power -2. However, since it is a complement number to get the back, you must actually keep these first two digits.

So this becomes 110, 110 is six divided by it because 1 upon 2, 4 so it is 8, 2 to the power 3, 2 to the power -1, 2 to the power -2, 2 to the power of -3 which is 8 and this number which I convert

decimal to be 110 which is 6 so -6 by 8. But you actually if you do not truncate the extra bit you can never get this 6 by 8. So, there is an overflow.

Similarly, if I do this the last two numbers 1.00 into this, then I get 0.011 which essentially is 101 which is 3, and sorry this is 3 this is both minus therefore the net number is zero here, so value is plus. 11 is 3 and 3 bits so 1/8 so 3/8 so I can see from here this because of the additional bit which appears, because of the number I have chosen in sign bits are variable the twos complement automatically rejects the overflow.

And that is why probably most of the partials sums in any adder or even later multiplier, we probably look for twos complements. The another number system which is often used in DSPs at least or ASIC based DSPs is called redundant numbers system. It is used as I said in most of the DSP chips which is based on ASICs.


**(Refer Slide Time: 30:21)**

**Redundant Number System**

- ★ Its used in ASIC based DSP chips.
- ★ It simplifies arithmetic.
- ★ It speeds-up arithmetic as no long carry-chains.
- ★ Its cost is high as more complexity is involved.
- ★ Zero and sign detection as well as conversion from conventional to redundant may reduce the speed.

There are three representation of the digit codes:

1. Signed Digit Code (SDC),
2. Canonic Signed Digit Code (CSDC),
3. On-Line Arithmetic.

 Lars Wanhammar

The advantage of redundant numbers system is it simplifies arithmetic. It speeds up arithmetic as no long carry chains. Its cost is high as more complexity is involved. Zero and sign detection as well as conversion from conventional to redundant may reduce the speed because it will take some time finite time to do these operations and – but still one of the major reason why RNS numbers or redundant numbers system is used is very easy to implement using ASIC chips.

ASIC based designs because it simplifies the arithmetic and to some extent you may lose some speed here but you may gain little larger speed because of there is arithmetic change is no long carry chains here. So you may gain some speed here, lose some speed here, net you may get little more speed than what you thought otherwise. However, it is not free because it has a cost as high as more it becomes extremely complex chip design and therefore cost of design increases.

So one cost wise its higher, speedier, simpler to implement however these two cost as I said. Now if you look at the three possibilities in which this redundant number system uses Digit codes. There are three representations known, one is called signed digit code - SDC. Canonic signed digit codes - CSDC, On-line Arithmetic Code which we shall see later. As I said three most of the Sign magnitude number has only one bit representing one, either it is + or 0.

**(Refer Slide Time: 32:02)**

**Digit-Codes**

There are three representation of the digit codes:

1. Signed Digit Code : Each digit has it's own sign i.e. -1,0 or +1 values.

- So long carry propagation is avoided.
- May reduce addition operation in some multiplication algorithm.

A number  $x$  in the range  $2+Q \leq x \leq 2Q$  where  $Q = 2^{-w}$


In the signed Digit Code (SDC)  $x = \sum_{i=1}^{w+1} x_i 2^{-i}$  Where  $x_i = -1,0,+1$

An example :  $(15/32)_{10} = (0.01111)_{2c}$  can be written as either  
 $(0.10001)_{SDC}$  or  $(0.01111)_{SDC}$

Similarly,  $-(15/32)_{10} = (1.10001)_{2c}$  can be written as either  
 $(0.10001)_{SDC}$  or  $(0.01111)_{SDC}$

Minimal SDC will have minimum Non Zero Digits so useful in Multiplication

**Lars Wanhammar**



However, in the case of Sign digit code each digit has it is own sign that is it can have a minus one value zero value or +1 value. Since it has three values, so long carry propagation is avoided because you do not have to do - carry occurs because there is a plus sign or minus sign appears during the operations. Since each bit is carrying it is own sign bits so one does not have to evaluate carry too long and therefore it may be speedier.

I may show you some actual design using digit codes, or at least maths behind it. May reduce addition operations, some multiplication algorithms, because not every algorithms but many

algorithms if you use which are used in multiplications. In number  $X$  in the range of  $2 + Q$ ,  $X < 2Q$ , where  $Q$  is again  $2$  to the power  $Wd$  is the word length. In the case of SDC which is called Signed Digit Code one can represent this as  $R^{rwd-1}$  where  $X_i$  is  $-1$  it is a weight  $X_i -1, 0$  or  $+1$ .

So take an example  $15/32$  in this and  $2$ 's compliment this number is  $0.01111$ , five ones, sorry four ones then in SDC I can represent that since it is a  $0$  so positive number, so I can now represent this number as it is  $0.0111$  or I can also represent the same number into its compliment that is  $1001$ . And either in the true form or in the compliment form this number can be represented and wherever you need compliment, you can use this code if you want it true number you can use this code and your operation will be very much simpler.

Particularly, like in subtractor you do not have to then use this code you may use this code actually. Similarly  $-15$  by  $32$  can be represented in  $2$ 's complement as  $1.10001$ , okay. Now either, now since it is  $1$  and is a Sign magnitude, so you know it is minus, so you write  $10001$  SDC as this number which is the minus sign or you actually give it a compliment of this, complement of all of it, one remains one.

But the rest bits are complimented, so  $0$ , so this represents  $-15$  by  $32$ , this represents  $+15$  by  $32$ . Now what is the advantage of SDC which I already stated to some extent but the major interest in implementing all arithmetic hardware using SDC has that it will create minimum non zero digits because if your number or the coded data becomes has larger zeros, then real multiplications.

Those zero terms can be directly put to zeros and therefore the partial sums which you are creating then may actually won't require to evaluate every now and then. And this idea that since the SDC reduces the number of one's compared to  $0$ s, this can be useful particularly multiplications.

**(Refer Slide Time: 35:31)**

## Digit-Codes

### 2. Canonic Signed Digit Code (CSDC) :

CSDC is a special case of SDC.

Each number has a unique representation.

Useful in simplifying multiplication.

It is represented as,  $x = \sum_{i=1}^{w_d} x_i \cdot 2^{-i}$  where  $x_i = -1, 0, \text{or } +1$

That is,  $x_i \cdot x_{i+1} = 0, 0 \leq i \leq w_d - 1$

Here no two consecutive digits are non-zero

CSDC has always Minimum number of Non-Zero digits ( $\cong W_d/3$ )

Which is  $W_d/2$  in normal binary representation.



Lars Wanhammar

The other possibility is called Canonic Signed Digit Code actually it is SDC but each number has a unique representation instead of two possibilities we define this is the one number which the way we will define. Here no two consecutive digits are non-zero where as in the first case it can be. So the idea in because of the way we define our canonic form we say the two nearest numbers can never be one and that is the great thing.

Because you are certainly 50% of them will become zero at any cost and because of that we may have much lower zeros or much larger zeros than the data which reduce the CSDC has minimum number of non zero digits which is  $W_d$  by 2 is normal – see normally you may have a possibility of 50% 1s and 50% 0s. This will always be less than 50% so number of zero will always be larger which means in multiplication in specific the time taken to multiply will be very, very small and therefore you have a faster multiplier using canonic Signed Digit Codes.

**(Refer Slide Time: 36:36)**

## Digit-Codes

### 3. On-Line Arithmetic :

This code uses latency. The  $i^{\text{th}}$  Digit of the result is computed using only the first  $(i + \delta)^{\text{th}}$  digits, where  $\delta$  is positive constant.

Once first  $\delta$  digits are available the first digit of result can be computed and following digits can then successively computed for new digit of operand.

Extremely useful in recursive algorithm having long word length



Lars Wanhammar

The third and interesting one is On-line Arithmetic Code. This code uses latency the  $i$ -th digit of result is computed using only the first  $i + \text{delta}$  digit where delta is a positive constant. Once first Delta digits are available, the first digit of result can be computed and following digits can successively computed for the new digit of operand. This is a very useful method of doing arithmetic.

Because if you see a typical DSP chip or DSP processing system like an FIR filter or take any recursive filter which have recursive algorithm, you have a number of adders and multipliers and chain of depending on taps you use in this case you can delay, you can do once till some delta bits are completed and then the first bit start coming. So initial delay but then every time cycling you have another bit coming out so overall the speed may improve.

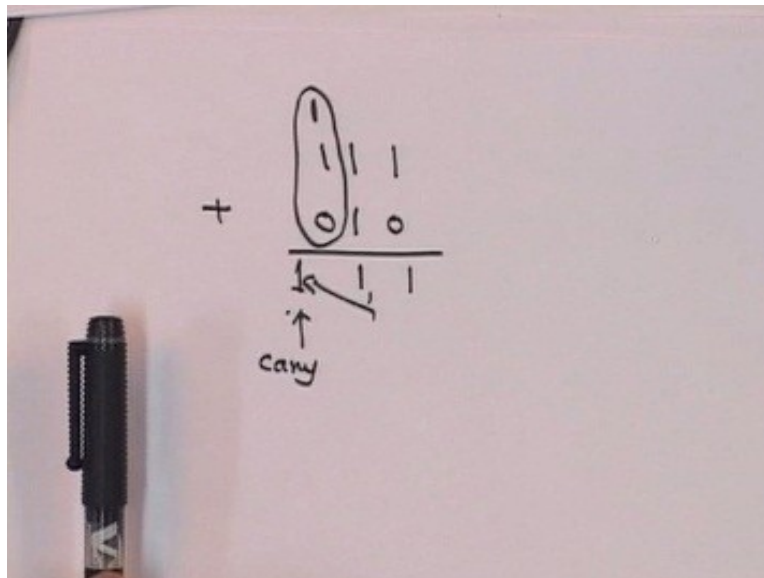
This is like using some kind of a pipeline. So online arithmetic is equivalent of putting some kind of a pipeline in coding and because of this kind of coding you can do pipeline circuit equally saying that it will be faster. Another possibility of representing digits in the case of any system particularly for binary which we are interested in, okay, why are we doing all this I keep saying that I have some things to always look for.

I am looking for high speed arithmetic operations at low power. If your number of operations are smaller; then one may say the power is reduced. If your data flow is faster and gets the data

output faster, then you say your speed is higher. So we are looking for both specifications one the high speed and low power and we keep looking for a variety of ways or variety of codes in which it can be minimized or speed can be increased or the delay can be minimized.

There is in most arithmetic which you use for adders take it any arithmetic serial or bit parallel or any of them, you have two numbers to add and you may generate a carry and till the next 2 digits are to be required maybe I can give you a very trivial example.

**(Refer Slide Time: 38:55)**



If you have a number 11 and you have a number this, so you add 1 and 10 and now this one is carry. So if you have another 10 or 10 here this one carry has to appear here and then you will add, so this operation cannot be performed till this carry has been generated which means for the successive bits carry must come from the last operation of addition and that means it always be delayed.

If you have a large amount of bits – 8-bit, 16-bit, 32 or 64-bit data then the last bit carry will be coming so slow I mean by so much delay per bit that you will find that the net system output will be very, very slow. So one variety of techniques which we are going to work today on later, we are going to look into this feature saying that can we actually avoid the carry propagation itself, okay. We will see in a circuit ways or we will see different codes we are seeing.



We will see that another possibility of avoiding carry propagation in a real system is Chinese remainder theory using a Chinese remainder theorem. Now please remember this is, I am not very sure but it is some kind of few BCs or few ADs time this remainder theorem was created few ADs not BCs this theorem was created and that time probably it was a mathematical thinking which was later used by electrical people. So let us see what is the theorem.

**(Refer Slide Time: 40:43)**


**Residue Number System(RNS)**

Avoids carry propagation.  
uses old Chinese Remainder theorem

Remainder theorem :- For a fixed point integer, the RNS number is represented by a set of residues obtained after integer division by *Mutual Prime Factors*;  $m_i, i=1,2,\dots,p$ .

An integer number  $x = q_i m_i + r_i$   
Where  $q_i, m_i$  &  $r_i$   
are integers, then  $x = (r_1, r_2, \dots, r_p)$

<p>Example : Modulo :- 5, 3, 2. Range=5.3.2=30. In RNS 9 is = (4,0,1) → 19 is = (4,1,1) 3 is = (3,0,1) 8 is = (3,2,0).</p>	<p>Addition : <math>9+19=(4,0,1)+(4,1,1)</math> = <math>(4+4)_5, (0+1)_3, (1+1)_2</math> = <math>(3,1,0)=28</math></p> <p>Product : <math>8 \cdot 3 = (3,2,0) \cdot (3,0,1)</math> = <math>[(3 \cdot 3)_5 \cdot (2 \cdot 0)_3 \cdot (0 \cdot 1)_2]</math> = <math>(4,0,0) = 24</math></p>
--	---



The theorem says for a fixed point integer, the RNS number is represented by a set of residues obtained after integer division by mutual prime factor which is  $m_i$  is equal to  $m_l$  which is  $i$  is equal to 1,2 up to  $P$ . And the integer number  $x$  then is  $q_i m_i$  plus  $r_i$  where  $q_i, m_i, r_i$  are integers then we say  $x$  is equal to  $R_1 R_2$  and things like that. Now this is not very obvious after we read this statement of Chinese Remainder theorem.

One does not really get a feel that it is really going to do a great job but let us see here some examples and some method I will show you. The way it is,  $M$  what is called Modulo, we say we can choose a modulo for say three bits 5,3,2 is arbitrarily chosen, do not worry these are only numbers, any number any basis number can be chosen. Choose 5, 3 and 2. Then we say the product of the three, 5, 3 into 2 is 30 is the range of bit number of possible range in which this operation can be performed.

Now we want to know what we are going to represent any number in decimal 9, 10, 12, 20, 100 whatever it is we should be able to represent in this modulo form 5, 3, 2 three bits of number will come corresponding to base 5, 3 and 2 and we will figure it out what is the remainder of that. And once we get that remainder that will represent the number 9 or 19 or 3 or 8. Here is for example take a question of 9 and since we are starting with modulo 5, 3, 2.

So we first pick up the number 5 and divide it to 9, 9 is divided by 5. So if I divide 9 divided 5, I get 5 into 1 is 5 or maybe I can show you 9 I divide it by 5, so I get remainder is 4. So the first residue number presenting 9 In the 3 bit modulo 5, 3, 2 is 4. Next take the case of 3 so  $9 / 3$ , 3 into 3 is 9 so we now get a residue 5 into 1 is 5, 3 into 3 is 9 the residue number is 0. So I get Residue 0 and finally that 2 and 9 so I say 2 into 4 is 8 and I get residue number.

So I get 401 as the number which represents as the residues the number 9 for a modulo of 5, 3, 2. Please take it if I choose any other modulo for example I choose 2, 3, 1 or 2, 3, 4 for example as modulo then the same number nine 2 into 4 is 8 so 1, 3 into 9 is 0 and 4 into 9 is 1, so now this number in this modulo will be representing 9. So it is not that it is unique but choice of a modulo decides the range of bits you want to work at and based on that one can choose depending on the range you have and RNS number can be represented.

Based on this we have done calculations for 19, 5 into 3 so 15, so remainder is 4, 3 into 6 18 remainder is 1, 2 into 9 = 18 remainder is 1 so you say 411 for this modulo represents 19. For three one can say if I divide 3 by 5, 3 5 into 0 so residue is 3, 3 by 3 is 1. So Residue is zero 2 by 3 residue 1 so 3 is represented as 301 similarly 8 is represented as 320. Please remember if I change the modulo, these numbers will also change.

The statement I made that you do not need carry because you are representing only their numbers in residue, so we have an addition. For example,  $9 + 19$  is addition I wanted to do. 9 is of course 401, 19 is 411, so if I do this I add this to  $4 + 4$  which is under modulo 5,  $0 + 1$  which is under modulo 3 and  $1 + 1$  which is under modulo 2. So if you see it this becomes 8 1 and 2, okay, 812 now this one 8, 1 and 2 can again be represented in terms of 532 8 is  $8 / 5$  is 3 as a residue.

This is 1 divided by 3 which is residue 1 because 3 into 0 is 0, so 1 and 2 / 2 is 0 2 into 1 is 2, so residue is 0. And you can see whether this 310 represents 28 in 532, you can just take it 5 into 5 is 25 residue 3, 3 into 9 is 27 residue 1, 2 x 14 is 28 residue 0. So which means the number 310 represents 28 19 plus 9 is 28. Now you can see from here these digit numbers which we have used here does not require any representation of minus signs, okay.

So if you represent your numbers in this there is no carry required because there is no carry generated actually. Now let us look at this number. Same numbers are now getting let us say I multiply 8 into 3, so this 8 into 3 is 3 into 2 into 0 is 8, 3 into 0 into 1, now I make a product of the two. So I do individual products 3 x 3 is 9, 2 x 0 is 0 and 0 x 1 is 0 and each has a modulo of 531, this gives me 9.

But if I have a 9 in terms of modulo 5, this is residue 4, 00 will give me residue 00 in this case, okay. So since there is a 400 you can again divide it by, you can check it 5 x 4 is 20 residue 0, 3 x 8 24 is 0, 2 x 12 0 so 400 represents in this modulo 5, 3, 2, 2, 4. Now this residue number system has been tried successfully. However, one can see from here though the overall the speed maybe faster because there are no carry chains in addition.

And therefore in products but there is a requirement coming because you will have to convert all your numbers for a given modulo. Firstly, we have to make a choice how to choose a modulo, so some hardware is required to make a proper chart depending on the data, making that choice then you will have to convert them the numbers into RNS form and then do these operations. They themselves maybe very fast, one doesn't know how fast these can always be for a given data.

And therefore on an average whether one can always say that RNS system will be faster is slightly talking too big but in most cases, in most data which you see this may be a valid statement and therefore we may say that because of this, we may be able to solve the arithmetic much faster. For your information I have designed an FIR filter using, I means my student and my colleague Prasad Desai we have designed a 4 bit 8-bit FIR filter using RNS number.

And this has been actually accepted by Texas Instruments. So before we start having seen the coding part or digit part, we now look into possibility in which we can make the arithmetic code.


**(Refer Slide Time: 49:43)**

Arithmetic

1. Bit Serial: Less of Hardware, so saving on the Silicon Real state.
2. Bit Parallel: Simpler in implementation and appears to give faster circuits !!

- ☐ In Reality, ratio of speed in the two cases is very 'small' as parallel implementation has long carry propagation paths.
- ☐ Area of 'Bit Parallel' chip will be larger[by word-length( $W_d$ ) times]
- ☐ Both are roughly similar in power dissipation.

Therefore the optimum is *Series-Parallel* for chip implementation.

 NPTEL

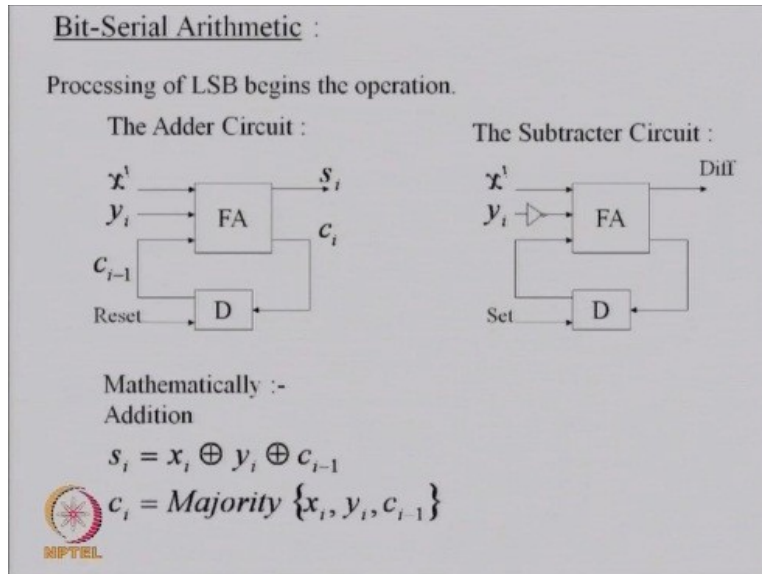
Lars Wanhammar

Now there are two ways one can do maths or arithmetic in the case of circuits - one is called Bit serial, another is called Bit Parallel. We will see this as we go ahead. The Less of Hardware, so saving on the Silicon Real estate, I mean it will be slower obviously because every bit is entered, some processing has been done, some part maybe feedback for the next data to be processed. So on an average it will be slower but not necessarily every time.

We will see some faster serial bit processing. However, in normal cases it looks slower. However, it may require only one block to design because at a time only single bit is entering, so your silicon area is smaller. Now bit parallel, all the bits are simultaneously being processed; obviously it will be faster however the hardware will be very large comparatively.

So we are trying to see can we do in between Bit Serial, Bit Parallel, partially this partially that and can we do better and in reality ratio of speed in two cases a very small as parallel information has a long carry propagation paths. Area of Bit parallel will be larger, both are roughly similar in power dissipation and therefore that is not every criteria. And therefore as I keep saying the optimum maybe series parallel combinations.

**(Refer Slide Time: 51:06)**



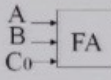
A typical circuit shown here is bit serial arithmetic processing of LSB begins the operation. This is your full adder for example, this is your D flip flop or 1-bit register, you have data X 0 Y 0 first appear out of X I Y I and this is the sum is Xi or Yi Ci -1 and the carry generator is maturity of Xi, Yi, Ci majority gate of these three. So in the – this full adder this is what has been performed, you create Si and you get first S0C0.

And this has to be now going for the next X1 Y1 data when it comes this C1 has to be fed back or next bit last carry has to be now reply to this and therefore a delay element has been added. When the next bit appears second bit appears, you are already one-bit delay so one-bit delay is provided from here and now three things Xi Yi Ci minus 1 do another Si Ci and you keep doing operations till the bits are completed.

Please remember that last when the bits are clearly over you still have to do one more round because the last carry has to be re fed which will go into the MSP of the sum. So it is one more additional operation, then the number of times you go through please remember this is also a requirement in bit serial. Now the subtractor circuit is just at that you use complement number, one is complement X Y X i Y Y and then you do the same can be operated for a subtracter.

**(Refer Slide Time: 52:53)**

Bit-Parallel Arithmetic :  
 Full adder is the basic element and uses 2C number system.



$S = A \oplus B \oplus C_0$   
 $C_1 = A \cdot B + A \cdot C_0 + B \cdot C_0$

Thus,
 
$$S_i = A_i \oplus B_i \oplus C_{i-1}$$



$$C_i = G_i + P_i C_{i-1}$$

$$G_i = A_i \cdot B_i \quad \text{Generate signal}$$

$$P_i = A_i \oplus B_i \quad \text{Propagate signal}$$

For CMOS implementations, it can be further reduced as:

$S_i = \overline{C_{i-1}}$	If	$P_i = 1$
$S_i = C_{i-1}$	If	$P_i = 0$
$C_i = C_{i-1}$	If	$P_i = 1$
$C_i = A_i$	If	$P_i = 0$

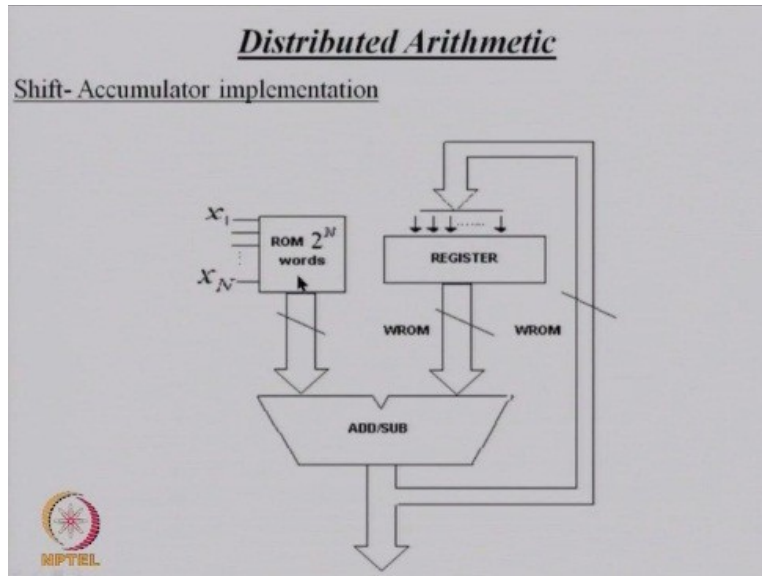
 

Whereas if you look at bit-Parallel here is that bit parallel circuit, full adder is the basic element and you just see number system okay, 2 is complement, this C is like big but doesn't matter. So ABC FA sum and C1, this is the sum and carry terms. This can be written in the generate form. So we now generate a term - two signals we generate, we say generate signal which is  $A_i \cdot B_i$  and we generate a propagate signal which is  $A_i \oplus B_i$ .

And one can show there without, next time we will come back and see that  $S_i$  is  $\overline{C_{i-1}}$  if  $P_i$  is 1, please remember  $P_i$  is  $A_i \oplus B_i$ , so if A is 0 and B is 1 or B is zero and A is 1 only then  $P_i$  is 1, otherwise  $P_i$  is 0 okay. So if any of the bits are same, two of the bits are same then  $P_i$  is 0 and that case the sum is equal to  $C_{i-1}$ . In the case of  $C_i$ ,  $C_i$  is equal to  $C_{i-1}$  if  $P_i$  is 1,  $C_i$  is equal to  $A_i$  if  $P_i$  is 0, I can show you this next time by putting a table and 2-bit table to clarify what I am talking about.

The Bit-Parallel, you can have all bits simultaneously so many full adders you will have  $A_0, B_0, A_1, B_1, A_2, B_2$  and you can actually connect the output or carry to the next block, next block, next block of course the carry has to propagate even in the case of bit parallel though the nut processing maybe faster.

**(Refer Slide Time: 54:40)**



In the last part before, we go to the actual ones is called Distributed Arithmetic which is what most processor will use. Your data has been input data is stored in a ROM which is 2 to the power in words. Then you have an add subtractor or any other operation particularly add subtract shown here, this receives the data. And the output is either given if you wish or it goes through a register for delay elements to create 8 bits can be simultaneously.

This is like a parallel serial, this is partly serial and then all bits are returned if they are carries, they come back through a ROM again and this together now you create the new sum and new carry. So this distributed arithmetic is the most likely arithmetic which probably one will use in the case of most of the processes. So we will stop here for the day. We will start with the adders, specific adders particularly kinds of adders, their circuits, their possible speeds, which one to use and what is the criteria of choice of an adder to be used.

Please remember if you are a VLSI person, your choice is only based on either you are reducing the power or increasing the speed and at times reducing the number of transistor and hence the area. So somehow based on VLSI implementation requirements next time we shall discuss this adder designs and we will have many of them. In fact, there are a large number of possible ways in which addition can be performed.

And the basic idea in every time is to see how reduce the carry part if possible. Most of this work which I am going to use as the basic work which is given in Rabaey's book on (()) (56:17) and others, Nikolic and Chandrakasan, so you can also look into them. However, I will explain them which sometimes one does not read properly in the books so I will explain. And then in the end I may give you some other adders like link adders or some other adders which does a lot of good things.

We will also look into possibly low power implementations and then show you that in a real advanced VLSI chips now there are many possibilities in which adders can be implemented. Thank you for the day.