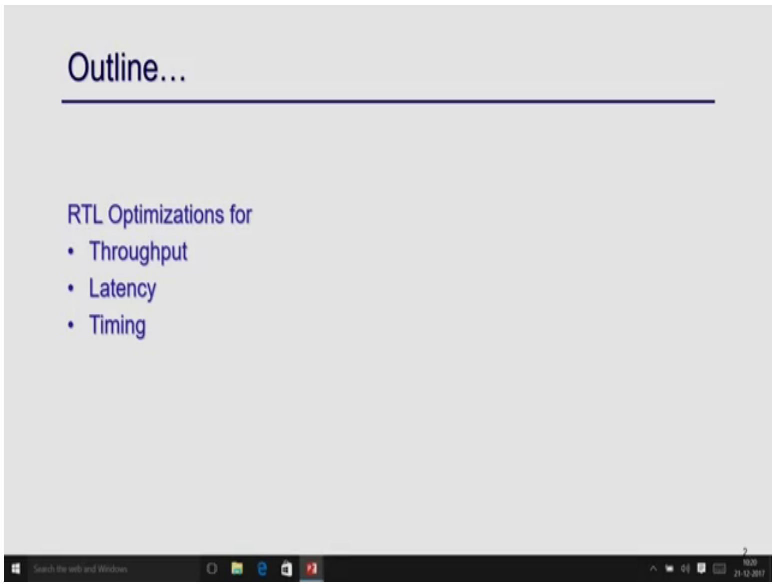
Optimization Techniques for Digital VLSI Design

Dr. Chandan Karfa
Dr. Santosh Biswas
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 06
RTL optimizations for Speed

Welcome everyone. So, in this module we are going to discuss about several RTL optimization specifically targeting improving speed of your design, how to improve your area of your designs, and improving power consumption of your design. So, there are certain kind of RTL optimizations are there and that can be apply in the in your design. So, that your design have a better timing or say idea or power right. So, in these 3 or 4, 4 modules we are going to discuss about all these things. Today the topic of our discussion is that RTL optimizations for speed ok.

(Refer Slide Time: 01:09)



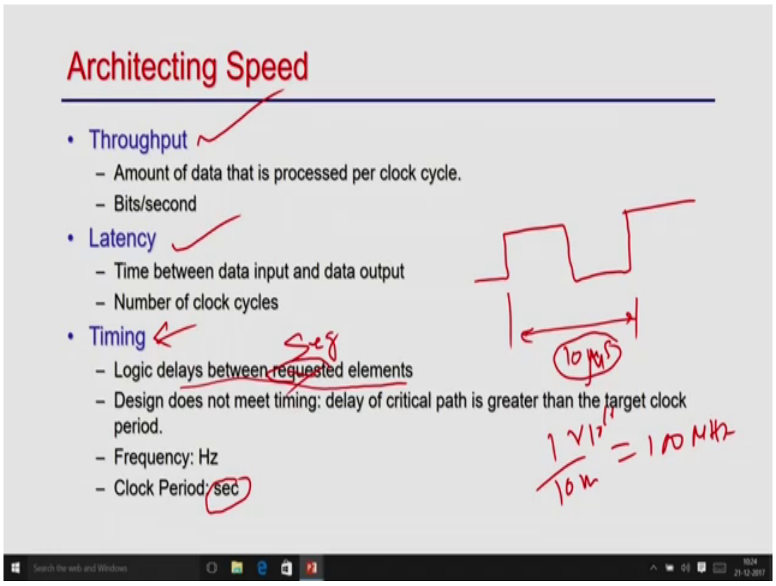
Outline...

RTL Optimizations for
- Throughput
- Latency
- Timing

So, as I mentioned. So, we are going to discuss about that several RTL optimizations today, and one we are talking about the speed of our design there are 3 parameters come into picture, one is throughput, one is latency and the next is timing ok. So, we are going to talk about. So, speed is characterized by one of these 3 terminologies, and we are going to talk about all the optimizations that are targeting all these 3 right. So, now, let us discuss what is throughput. So, you are talking about your design and what is the throughput is kind of amount of data that are processed by per clock cycle right. So, you are in every cycle some data input are that coming and some output is coming out right.

So, you basically the number of bits and that are coming out of your design that is called throughput ok. And what is latency? The latency is something the number of clock cycle is required to process your data right. So, if you give some data at say time t it might say some k cycles after that after k cycle only that output will come into you mean will be available. So, that is k care number of cycles ok.

(Refer Slide Time: 02:12)

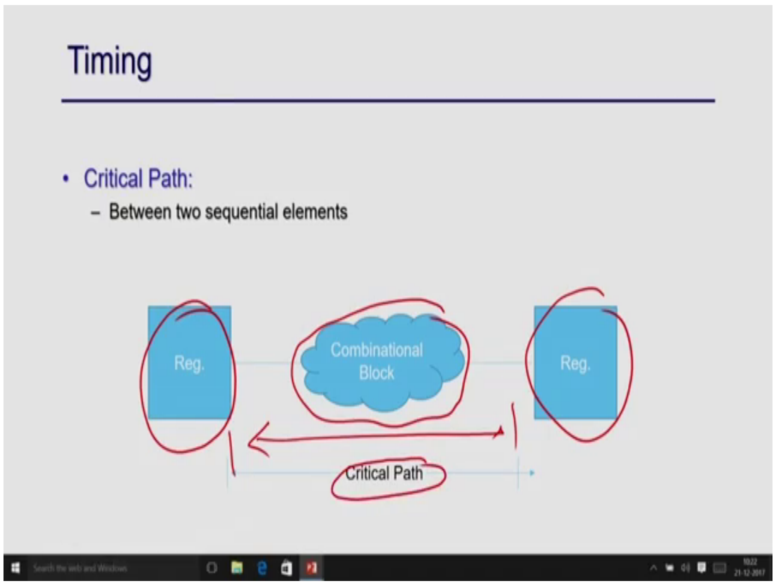


Architecting Speed

- Throughput
  - Amount of data that is processed per clock cycle.
  - Bits/second
- Latency
  - Time between data input and data output
  - Number of clock cycles
- Timing
  - Logic delays between ~~requested~~ Seq elements
  - Design does not meet timing: delay of critical path is greater than the target clock period.
  - Frequency: Hz
  - Clock Period: sec

So that is latency the number of your design and the next is timing. So, timing is something is related to the computational delay between 2 register, the maximum computational delay between 2 registers.

(Refer Slide Time: 02:25)



Timing

- Critical Path:
  - Between two sequential elements

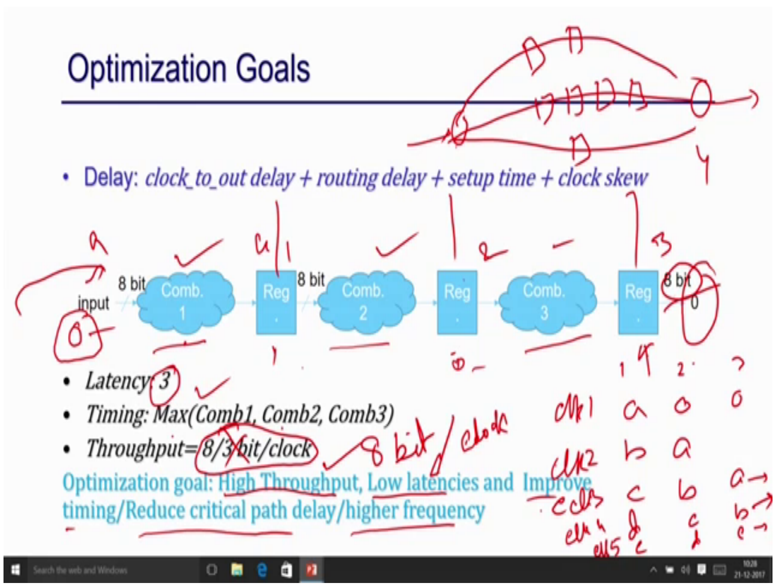
So, suppose there is a register here and there is another register here, and there are some logic circuits here right. So, combination magic and all some functional units there is 2 sequential elements here, and it need some time computational time to execute this right. So, the delay between this register through this is called the computational delay of that circuit right. And your timing will be determined by whether maximum size maximum of such delays right. So, you have several pair of registers right. So, there is say 100 register in your design. So, there may be say 500, 600 path between any 2 registers right and between all possible paths between 2 registers, maximum delay will determine the maximum computer bell that is called critical path of your design right

So, this is very well known. So that is the logic between 2 sequential element, sequential elements and so, this is basically and your timing or the frequency that can your circuit will achieve will be determined by this factor right. So, you maximum that clock period should be say if you say this is your clock. So, that clock should be minimum the clock it will be minimum of that critical path delay right. So, if say your clock period. So, and your clock period will be determined by this.

So, suppose your delay is say 10 millisecond. So, you will get 1 by 10 millisecond right your clock period is 10 milliseconds. So, your clock will be in seconds and frequency is 1 by 1 by clock period. So, that will give say a 100 megahertz because this is 10 to the power 6. So, this is say microseconds (Refer Time: 04:15). So, this is microseconds. So, this will get 100 megahertz right. So, your clock period will be determined by this maximum computational delay right and your objective will be maybe different. So, your object may be only improve your throughput of your design right. So, then you have to apply certain kind of techniques that will actually improve the throughput of your design or you might try to improve the latency of your design right. So, that you try to go get the result as soon as possible right.

So, number of cycle which is less between this and your objective maybe improve the timing right. So, it is basically the high throughput, low latency and timing also has to improve; that means, the computational delay or that critical path length has to be reduced. So, that is the overall objective right.

(Refer Slide Time: 05:03)



Optimization Goals

- Delay: *clock_to_out delay + routing delay + setup time + clock skew*
- *Latency: 3*
- *Timing: Max(Comb1, Comb2, Comb3)*
- *Throughput=8/3bit/clock*

Optimization goal: High Throughput, Low latencies and Improve timing/Reduce critical path delay/higher frequency

So, let us take an example to understand this fact. So, suppose this is my circuit right. So, there is a computational unit here, the input is coming some computational circuits are here then there is a registers, then there is a another computational circuit and which process this data and output stored in this registers and again this computational unit and then this final results is stored in the register right.

And say for uniform it is. So, my inputs are all (Refer Time: 05:26) all the data for this is 8 bits. So, what is the throughput of this design days? So, once you give some input here, first we will discuss latency. So, it is. So, the first clock it will come here then the second clock this letter this process will come here and in the third clock this data will be processed and it will come at the output. So, in this 3 cycle right; so in the 3 cycle to give this input; so output will be generated for a given input right then will the latency is 3.

So, it is basically the number of logic this sequential register level from the input to output right and you can have multiple paths right. So, from your input there will be multiple path. So, some path maybe, say suppose if this path there are may be 2 register in this path there may be say 4 registers and there maybe 4 registers, this may have only one register. So, this latency will be determine the maximum number of registry in path from the input to output. So, this is your input and this is your output. So, for this path critical latency is 4 not 1 or 2 right. So, its the maximum number of registers there between the input and output for any path from the input to output ok.

So, for this circuits is there 3 level of registers the latency is 3 and what is the throughput. So, 8 bit are coming out in every for this 3 bit set. So, your throughput is throughput is something 8 bit per block right. So, basically what happened here? So, after 3 cycle your output will come, every cycle some output will come right. So, this is something wrong. So, your throughput is basically here 8 bit per cycle right why? Because once you give the first set of data here so, that will after 3 cycle the result will come right, but in the second clock you will give another set of input right

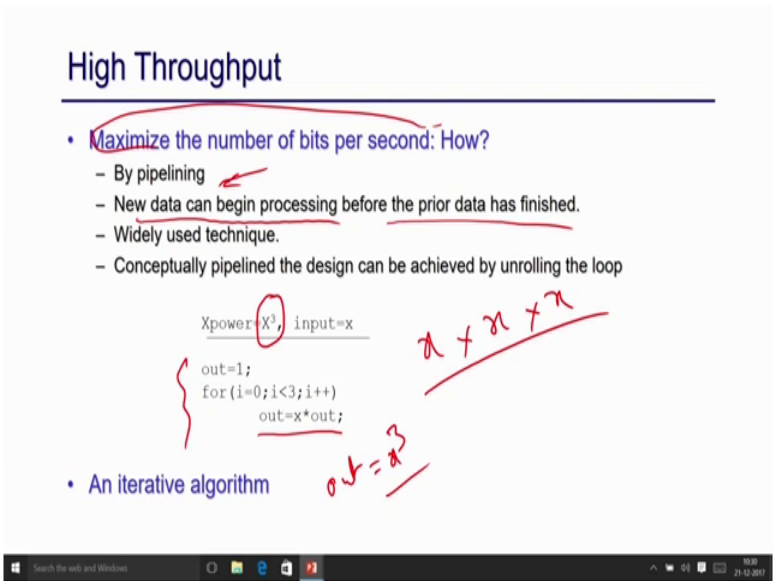
So, that will be also processed. So, suppose you give here say a and. So, result of a will come here, here after one clock right. So, if you just write this clock 1 clock. So, it will give a here. So, this is say register one. So, this is 1 2 3 right. So, you your register these are all 0, in clock 2 will come here and next input b will come right. So, in clock 3 clock 3, so a will come here b will be processed here in this registers this is register one this is register 2 this is register 3 and you will some will come right. So, 3 clock this result will be available in clock 4 again b will come here c will come here some new determine will come right.

So, after that every clock some output will be available right. So, in clock 4 this clock 5 u will come d will come and c will be available right. So, and after that every clock your every clock you will get some output right so; that means, your throughput is 8 bit per clock and the timing is what? So, I as I mentioned this is the maximum computational delay of any between 2 registers. So, here there are 3 set of computational delay. So, it will be a maximum of computational circuit one computational circuit 2 and. So, the delay maximum delay of these 3 circuit right.

So, for this circuit latency is 3 throughput is 8 bit per clock because its a parallel pipelined data path. So, after every cycle you will get some output for a data. So, it will be 8 bit per cycle and your timing is something, maximum of this computational delay ok. So, now, your objective is something you that you have to improve your. So, if you are talking about your speed of your design, you try to either improve your throughput. So, it will be either high throughput or low latency or your try to improve the timing; that means, reduce the critical path delay so; that means you are trying to improve the frequency right.

So, this is the objective for throughput. So, we are going to discuss of the how to improve your throughput, how to reduce your low latency, and how to improve your timing of your design all these 3 aspects we are going to discuss.

(Refer Slide Time: 09:38)



High Throughput

- Maximize the number of bits per second: How?
  - By pipelining
  - New data can begin processing before the prior data has finished.
  - Widely used technique.
  - Conceptually pipelined the design can be achieved by unrolling the loop

```
Xpower=X^3, input=x

out=1;
for(i=0;i<3;i++)
        out=x*out;
```

$x \times x \times x$

- An iterative algorithm

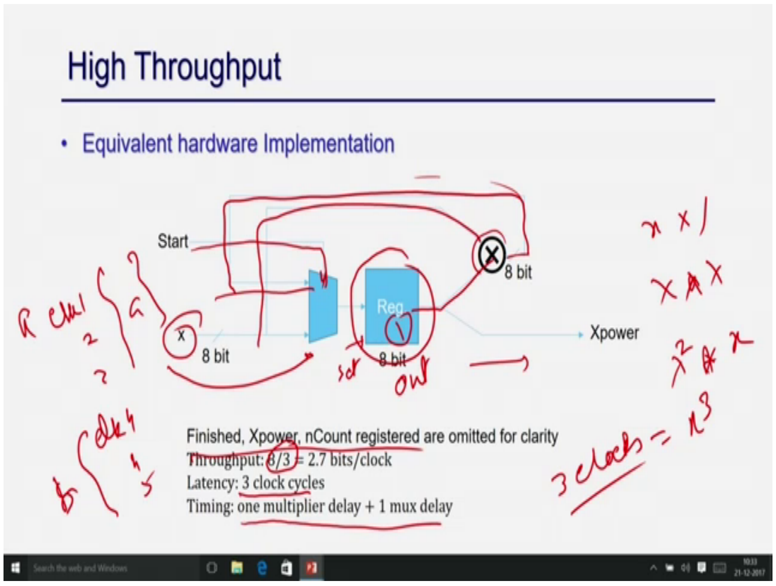
$out = x^3$

So, as I mentioned this high throughput is something in number of bit per coming out of the output, how to maximize this number of bit per seconds by pipelining right. So, as I mentioned here that. So, basically you have to process multiple data in parallel, what is happening here right. So, it is not that you give input a here and you can take 3 cycle and nothing is coming in, because if your circuit demands is such a its implemented this part is free, and you can actually process of some other part of the data.

So, it will just through the pipelining your throughput actually increases right. So, this is something happening here. So, basically to the pipelining your throughput will increase right. So, that is something I just discussed the new data can begin processing, before prior to data has finished right. So, before it has finished you can actually. So, this is the processing the new set of data right. So, let us take an example suppose you have to calculate x to the power 3 right. So, your input is x you have to calculate x power which is x cube right x into x right x into x into x.

So, you can write an iterative algorithm like this is c right. So, for i equal to 0 i less than 3, i plus plus out equal to x into out; so it will just calculate x x square and x x cube right after that 2 3 iterations you will get the results out equal to x cube, your out will be x cube right this is something in the iterative algorithm and its equivalent hardware will be like this.

(Refer Slide Time: 11:06)


High Throughput

- Equivalent hardware Implementation

Finished, Xpower, nCount registered are omitted for clarity
Throughput: 8/3 = 2.7 bits/clock
Latency: 3 clock cycles
Timing: one multiplier delay + 1 mux delay

So, you have this registers this is out and you. So, initially you just store this one your start signal comes, you store x here and you just multiply this x into out. So, initially this is set to one. So, this is as its set signal and it is set to one initially, and you just multiply and you store this back right you just store this back to the registers.

So, after start this x will come first little x then x square, because x into one initially it is going to happen in x in the first plugin x into 1 because x is going here right. So, x is going here and this is 1, then is the x next clock this is. So, that this is stored back to this.

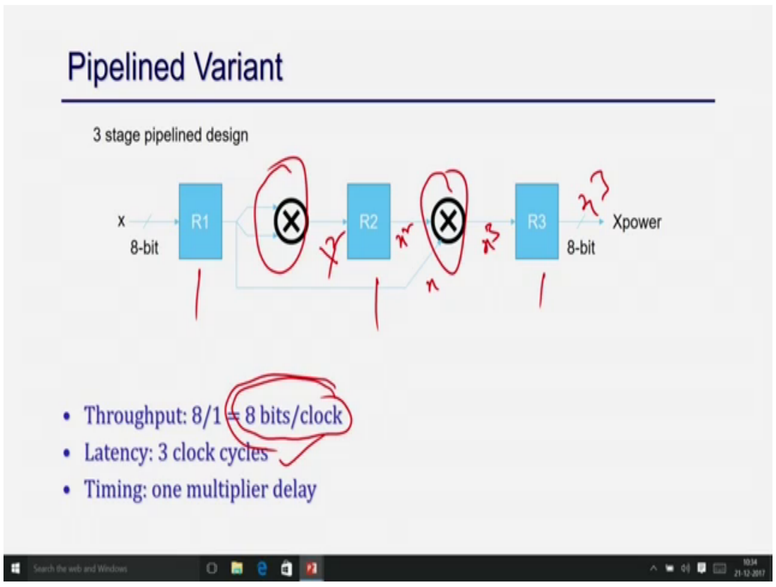
So, x into x right and then this is. So, it will become x square and then x square and then you multiply with nx and so, it will become x cube right. So, this is how this the iterative algorithm work and it needs 3 clock cycle right 3 clocks to compress right. So, this is something iterative algorithm and I will just group the other signal to make the circuit a little bit simpler.

These are since it is a 3 cycle you need some finished signal right. So, in this some has to be some finished signal, that my data is ready now or you need the counter that n count which is actually can counting this 3 times 3 iterations. So, those are, but those will be added, but the basic idea of this design is that I can actually do it iteratively and I can do it in 3 cycle right. So, this is something and what will be the throughput of this design? Since I am actually and this is something indeed 3 cycle to process one data right. If you just give me one x it need 3 clock then it will finish then we can only give the next set of the data right.

So, if you just give a here then clock 1 2 and 3 will just process a right and then you have to clock 4 you have to give b contract to the previous design where I give this things in all parallel right. So, in every clock I am giving a new input a b c d e f, but right this design I have to give you a and for 3 cycle because this will 3 cycle to compute a cube, then you give b here and clock 4 5 and 6 to compute b cube then you give this right. So, this basically in every 3 cycle you are going to produce one data right

So, your throughput will be 8 by 3 right because I assume my when all data per 8 bits and latency is also 3 clocks cycle because this registers though even its a register one, but effective you can see that my data is processed 3 times. So, it is going through this same registers 3 times. So, the latency is 3 clocks in the cycles right and the delay of this register things is maximum one multiplier followed by this mux. So, the maximum timing is determined by the delay of one mux and the delay of a multiplier right. So, this is something if we just implement a iterative version of this algorithm similar to the c code you will get this right. But as I have discussed the pipeline version in the previous cycle you can have a pipeline version of this design, where I actually used 2 multipliers similar to mux based my previous design this.

(Refer Slide Time: 14:24)



Pipelined Variant

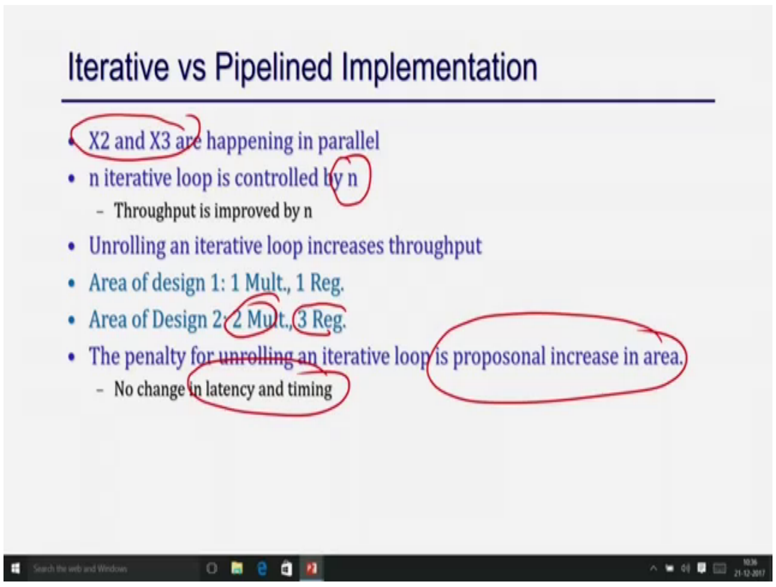
3 stage pipelined design

- Throughput: 8/1 = 8 bits/clock
- Latency: 3 clock cycles
- Timing: one multiplier delay

So, this is now a multiplier right. So, you can think about this is my multiplier right. So, this computation is just not a multiplier. So, you can just. So, this is that version. So, I have this x and just multiplying this. So, here it will be always coming x square, here this is x square and this will be x. So, this it will be x cube. So, x will be the output assuming the same thing, but I am just making pipeline for the variant. So, I just the pipeline the whole design, and here you can see that as I mentioned earlier also after first 3 clock after 3 clock onlt the first data will come, but after every cycle you will get one on. So, your throughput will be 8 bit per clock, your latency is 3 three set of registers. So, it does not change the latency, and the delay of this is something one multiplier right.

So, the delay of one multiplier, because maybe here also is a multiplier, here also multiplier, the delay of this particular circuit will be one multiplier delay of one multiplier right. So, what happened here? From this 2.7 bit per second per clock to I make it 8 bit per clock. So, I put the throughput of the design by just pipelining. So, pipelining is kind of a technique which actually improve the throughput of a design right; so if we just do this, but here you can understand. So, what is happening here is that x square and x cube calculation is coming in parallel for a given input right.

(Refer Slide Time: 15:31)



Iterative vs Pipelined Implementation

- X2 and X3 are happening in parallel
- n iterative loop is controlled by n
  - Throughput is improved by n
- Unrolling an iterative loop increases throughput
- Area of design 1: 1 Mult., 1 Reg.
- Area of Design 2: 2 Mult., 3 Reg.
- The penalty for ~~unrolling an~~ iterative loop is proposonal increase in area.
  - No change in latency and timing

So, this is the important factor, but the resource utilization resource increases right. So, earlier this is one multiplier, I am now using 2 multiplier right. So, basically the number of. So, as earlier I was using one multiplier one register now I have 3 registers and 2 multipliers.

So, the resource is going in proportion to the it will increase in proportion with the this 2 factor right the rolling factor pipeline factor. So, we will be determined by this. So, if you are throughput improves by say n times your area going to be increased by n times also right we will focus on it to n times some n times, but most likely there is no change in latency and timing right.

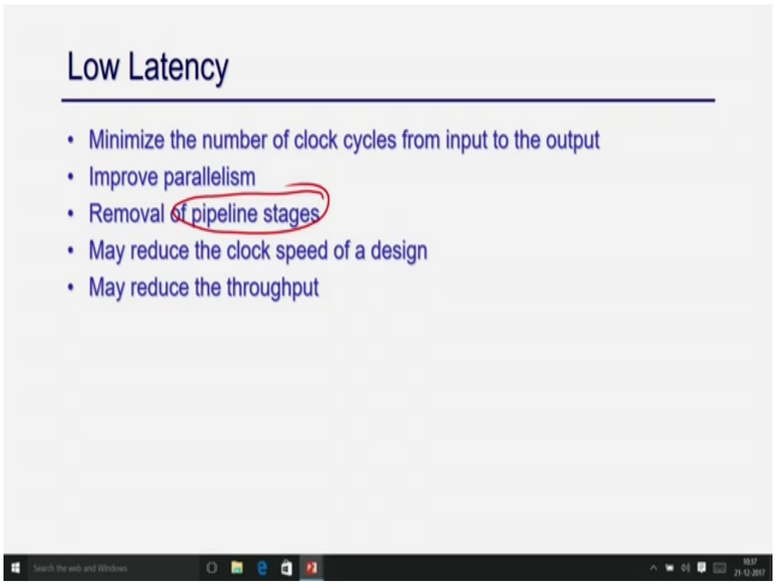
So, this is mostly (Refer Time: 16:14), but so, basically understand that if you want to improve your I mean throughput of your design, your area will also increase, but if you do not care about your area because your design is small even if it takes more area it does not matter for you because your design is small. So, in that case you can if you want to improve your throughput this pipelining can be a approach, which is actually you can apply to your design right.

So, this is one you have to make this, but you have to keep in mind if your design is very very big and area increment is not something is acceptable for your design then probably, you cannot just apply this kind of technique. So, it all depends on your target architecture, target design, target design size requirement and right.

So, you cannot actually do all this you cannot improve your throughput, you cannot improve your timing and improve your latency all access. So, may improve your latencies in the same the same time, either you have improve the throughput in the expense of latency or expense of area or if you are going to reduce the latency in terms expense of timing you are decreasing the timing or you are decreasing the throughput or if you want to improve your timing with the expense of little bit of extra area and throughput.

Right. So, this is something is always there. So, there is a trade off and it depends on your design choice which one you are going to choose ok. So, we are discuss about this high throughput its a pipeline increments. So, you can achieve that.

(Refer Slide Time: 17:41)

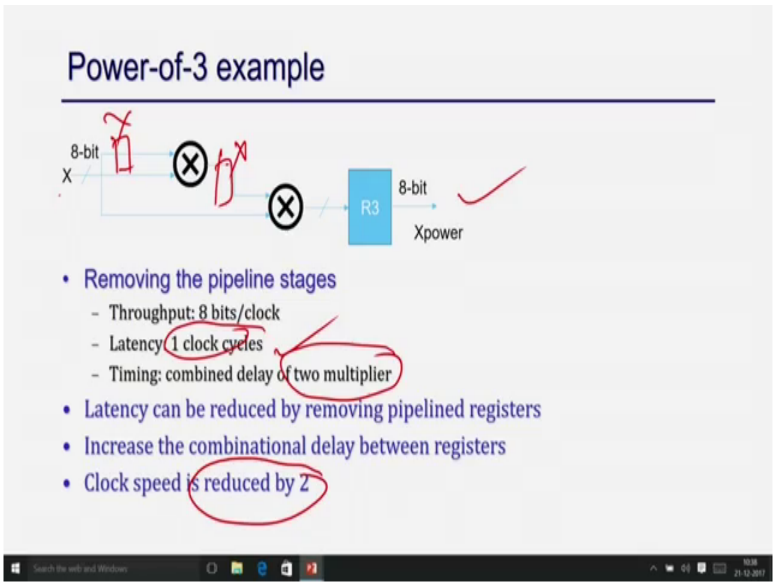


Low Latency

- Minimize the number of clock cycles from input to the output
- Improve parallelism
- Removal of pipeline stages
- May reduce the clock speed of a design
- May reduce the throughput

Now, I am going to talk about the low latency one, so when your design. So, low latency means what? So, you try to reduce the number of register level in your design ok. So, it is basically just removing some register right pipelined register, you just remove some parallel pipe you remove some register from this pipeline stages right. So, this is something you just do.

(Refer Slide Time: 18:02)



Power-of-3 example

- Removing the pipeline stages
  - Throughput: 8 bits/clock
  - Latency 1 clock cycles
  - Timing: combined delay of two multiplier
- Latency can be reduced by removing pipelined registers
- Increase the combinational delay between registers
- Clock speed is reduced by 2

So, for example, in the previous design, I have a register here I have a register here. So, I will just remove this register and I will just remove this register if you just see this figure, I just removed this register, I can remove this register, then all these things are working parallel right.

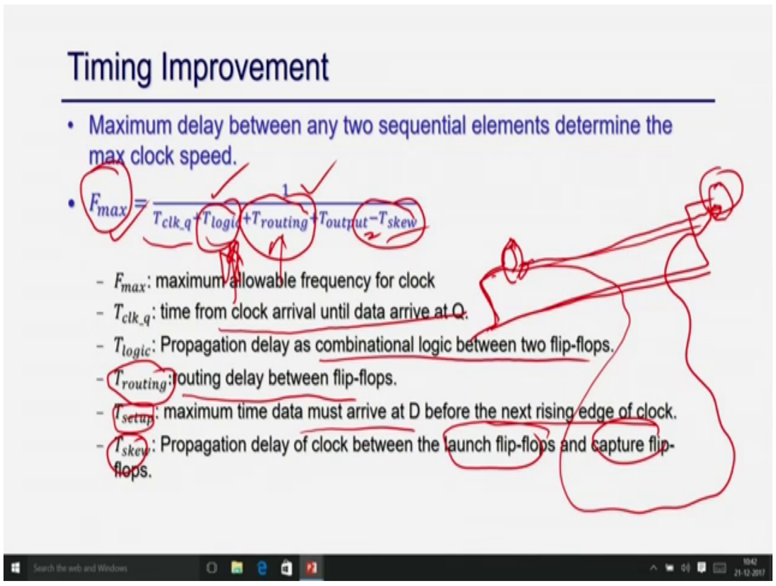
So, this is what I have done. So, here all this things still my throughput is 8 bit per second because every clock I am going to get output, latency is reduced from 3 clock to 1 clock because now if you want to put in 1 clock, whole things are done in the same clock and the output will be available in 1 clock itself right, but your computational delay increase now 2 multiplier.

Earlier it was only one multiplier because there is a pipeline stages here, now I have 2 multiplier in the path. So, you just improve in decrease your timing by a factor of 2. Earlier if you were choosing 100 megahertz see now 50 megahertz because you are now there are 2 multipliers in series and multiplier usually a very complex design. So, your clock period will be reduced by 2 right.

So, this is something is going to happen. Again its suppose if your primary objective is to get it latency in reduce the latency, then probably this is acceptable. But most of the time this improving timing is the important factor, most of the design people try to design or want to improve the timing or improve the frequency, because to achieve more frequency rather than this throughput and latency.

So, above this 3 these 3 factor this high throughput low latency and improve timing is the most important factor, and people put maximum attention in improving the timing. So, we will discuss that timing improvement now.

(Refer Slide Time: 19:38)



Timing Improvement

- Maximum delay between any two sequential elements determine the max clock speed.
- $F_{max} = \frac{1}{T_{clk\_q} + T_{logic} + T_{routing} + T_{output} - T_{skew}}$
  - F_{max}: maximum allowable frequency for clock
  - T_{clk_q}: time from clock arrival until data arrive at Q.
  - T_{logic}: Propagation delay as combinational logic between two flip-flops.
  - $T_{routing}$: routing delay between flip-flops.
  - T_{setup}: maximum time data must arrive at D before the next rising edge of clock.
  - T_{skew}: Propagation delay of clock between the launch flip-flops and capture flip-flops.

So, as I mentioned this clock period or the frequency is determined by the primarily this computational delay of that as I mentioned here, that computational delay of that that register right. So, that is something is determined by this right, this computational delay. So, now we can understand that, but if you just consider the actual factor there are multiple other factors, first factor is something clock to q which is basically clock time from clock arrival until data arrival.

Since its in your design maybe that clock arrived, but data is still not computed right. So, you need some extra time. So, that is called clock arrival until data arrives right. So, that is something clock to q and then logic, logic is something the computational delay which is you all know about the computational delay which is the logic delay, computational logic delay between 2 flip flops and the routing.

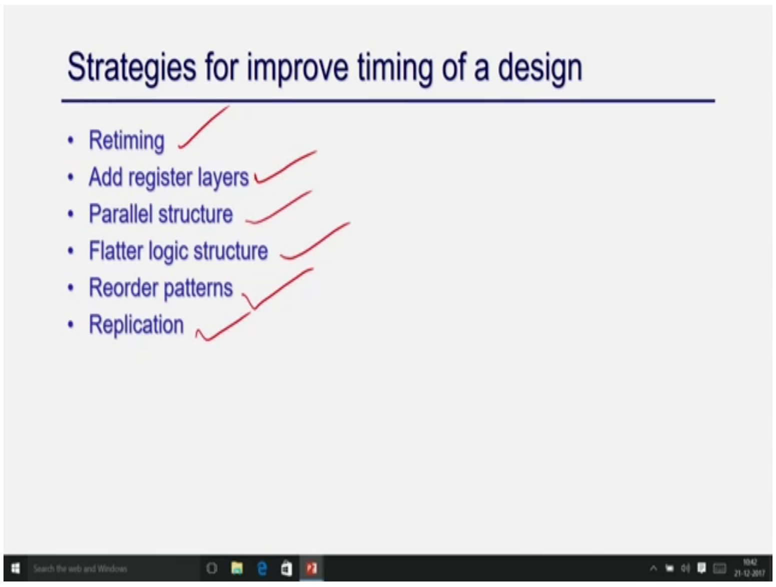
So, when you physically place those 2 registers right. So, if you have a register here and this is another register, they maybe placed very close by in the actual I C or they will be placed in very far right if it is placed very far then there is some time to remove this which actually have the routing delays because now you have to connect this 2 pin right here the output is going to this input this might go directly like this or it might be route like this right it might be route like this.

So, because of the routing I may be the routing path is very high or they maybe placed very high. So, the routing path automatically becomes very long. So, there are some time involves in routing also and also there are some setup time of the registers, which basically maximum time maximum time data must arrive at the before the next clock cycle. As you need some setup time. So, the data should be stable at the input of the register before the clock count so that this it can update in the next rising clock right. And then the Q is something the clock that is suppose the same clock is coming here to this register as well as this registers, but there are might be some delay right they may not come the same time.

So, the time this Q is something the progress in deliver till the launch flip flop this is the launch flip flop and this is the capture fill flip. So, the time difference between is coming here versus coming here. So, that is something is the extra time you actually get in your circuit right. So, which is which you that is why this is minus right. So, this is the actual the factor it will determine the frequency the maximum frequency to achieve in your design ok. So, you are talking about improving timing; that means, you have to achieve more frequency you try to reduce either logic or routing delay right.

So, other is mostly I mean there are other techniques to using Q and all those things, but primarily the primary factor we are going to talk about how to reduce the logic I mean computational circuit delay we can do some other kind of techniques so, that you will reduce that computational delay between 2 registers and so, that this factor is reduced; and also routing time we can do certain kind of optimization which we can improve the routing. So, we are going to discuss about how to improve this 2 factor. So, that if this factor going down so, this will be increase right. So, this is something we are going to achieve ok.

(Refer Slide Time: 23:01)

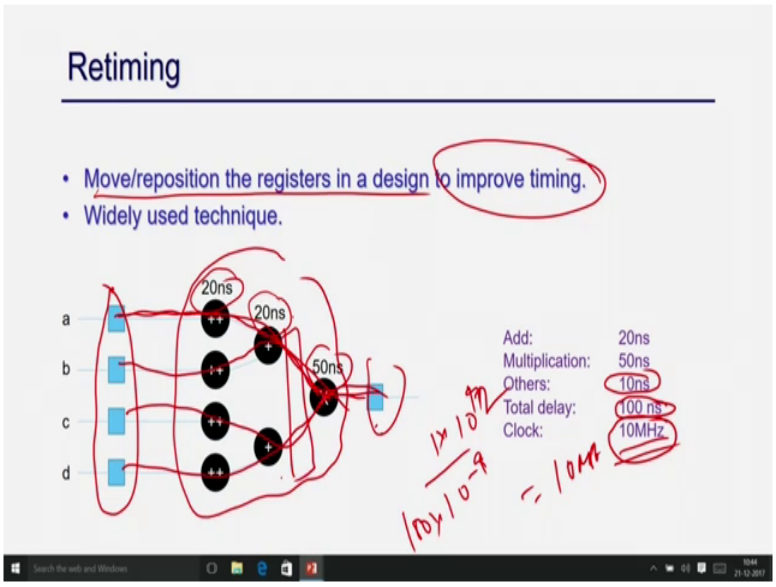


Strategies for improve timing of a design

- Retiming
- Add register layers
- Parallel structure
- Flatter logic structure
- Reorder patterns
- Replication

So, there are several techniques and very widely used techniques one of them is retiming, the next technique is add register layers then we can also do the paralyze some structures, flatter logic structures, reorder paths or replication right. So, these are the all kind of techniques that is are applying to your timing. So, we are going to discuss them separately ok.

(Refer Slide Time: 23:22)



Retiming

- Move/reposition the registers in a design to improve timing.
- Widely used technique.

Add:	20ns
Multiplication:	50ns
Others:	10ns
Total delay:	100 ns
Clock:	10MHz

$$\frac{1\times10^{?}}{100\times10^{-9}} = 10MH$$

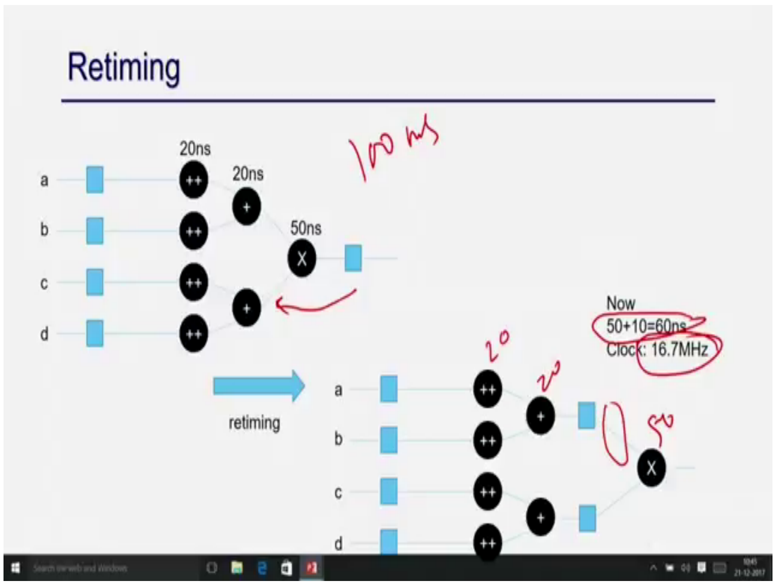
So, first we are going to talk about retiming. So, what is retiming? Its basically moving your registers in your design without changing the functionality of your design ok. So, you just move your or register in your design so, that the computational delay become less ok, but you have to keep it in mind that you should not change the functionality of your design.

So, it is basically move or reposition of your register in your design to improve timing without violating the functionality of your design. So, let us take an example here, suppose I have this bit logical. So, this is all my computational circuit between this register right these are the input set of registers this is the output set of registers. So, there are many paths, I have one path like this is 1 path, this is also 1 path, this is another path there are 4 paths right from the input to output and all path are involving the same set of registers. So, we can consider. So, there are all path are critical path right and if you assume by this addition is taking 20 nanoseconds 20 nanoseconds and multipliers you are 50 nanoseconds some of the other factor like this Q and this clock to q this setup time and all those things contributing 10 nanosecond.

So, total delay of this critical path is 100 nanoseconds right. So, you have 100 nanoseconds. So, you are going to achieve 1 by 100 nanosecond means 10 to the power minus 9 into 10 to this is 1 second. So, this is something you are going to achieve 10 megahertz right. So, this is something this is you can this is 10 to the power minus 9 into 100 right. So, this is the 100 10 to the power and this is 11. So, if you are going to this is 12 right sorry is 12. So, you are going to get a 10 megahertz right.

So, in this circuit you are going to get 10 megahertz, because there are total delay of this circuit is 100 nanoseconds ok. So, now, we can how we can improve this. So, you can actually move this registers here right. So, instead of placing this register here, you can place this register here. So, then your circuit performance will increase right. So, if you just think about this. So, this is my actual circuit.

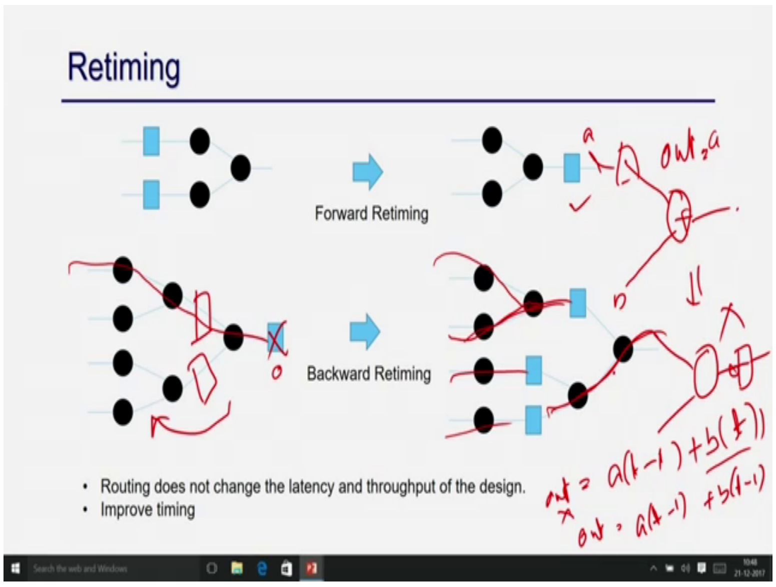
(Refer Slide Time: 25:21)



The maximum delay of the circuit is 100 nanosecond here what I did? I just moved this register here right. So, I just moved this register to here. So, now, what will be the maximum delay? So, this is something 20, this is 20, this is 50 and there are other delay. So, maximum will be 60 right 50 plus some other factor. So, there are 60 right. So, you are going to achieve more clock right 16.7 megahertz.

So, this is something is called retiming. So, you just move your register in your design so, that you can break the critical path right. So, you have to you try to reduce the critical path length and. So, that is something and again you have to keep in mind that you are not going to change the functionality of your design. So, there are certain factors, how we can move.

(Refer Slide Time: 26:05)


Retiming

- Routing does not change the latency and throughput of the design.
- Improve timing

So, if you can move registers if both the input there is a register then you can move it right. So, for example, if you have this if there are registers here and here then only you can move it to output right. So, if you can do this then only you can move these registers to this output right. So, if there is only one here this is not there then you cannot move this right.

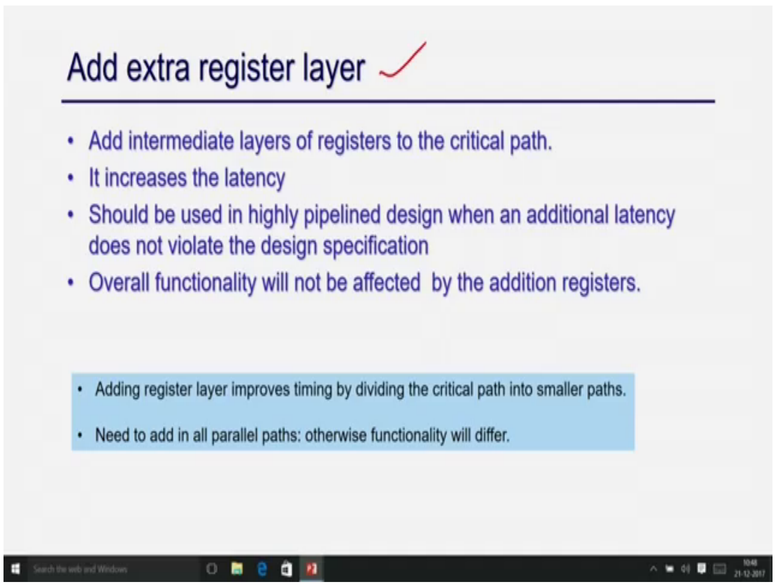
Then that is a function will change right. If you have only one register here then you cannot move it right because then the functionality will change because earlier what is happening? Both input is delayed by one and then calculation is happening. So, say output was like a say suppose this is a and this is b and there is a register here. So, earlier this is out equal to a t minus 1 whatever is coming in t minus 1 is going to this is this is plus and b t minus 1 and if you just and if you just move it here what is happening here. So, you can say let me just consider this example, I have only one register here and. So, this is what happening.

Then you are actually calculating this right a t minus 1 whatever the t minus 1 time data comes, you are adding with the current time b b t right t is the present time clock right, but in this circuit is happening. So, in this case out is calculating like out equal to a t minus 1 plus b t minus 1 right because there is a register here. So, whatever the computation is coming it will be delayed by 1. So, this 2 are not same right. So, if you have register in one path you cannot just move it to output. You can only move if there are register in both the input, you can then only move it to the output. Similarly if there is a output here it has to be move in the both the input right.

So, you cannot just move into one of the register. So, you have to move it. So, it will come here and here you remove it and then you can again move this to here right. So, you can move this. So, that is what I have shown. So, that is moved here. So, and this is this is again moved to both the inputs. So, this is the retiming circuit. And you can understand the earlier the delay of these 3 units now I have only 2 units right. So, all maximum path length is 2 units. So, here is 1 units, there are 2 units here, this is 1 unit this is 1 unit this is 2 units. So, maximum delay will be 2 unit of this particular module.

So, this is what we call retiming and retime is very widely used technique to improve your timing. So, this is and this can be automated we are going to discuss how to automate this whole process to improve your clock period of your design. So, this we have a separate module to discuss that. So, we will we will go into detail of retiming in that particular module.

(Refer Slide Time: 28:46)



Add extra register layer

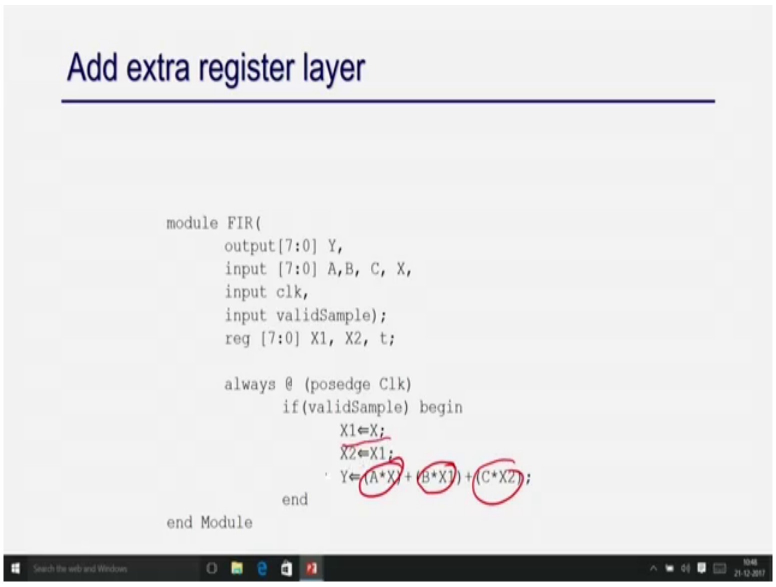
- Add intermediate layers of registers to the critical path.
- It increases the latency
- Should be used in highly pipelined design when an additional latency does not violate the design specification
- Overall functionality will not be affected by the addition registers.

- Adding register layer improves timing by dividing the critical path into smaller paths.
- Need to add in all parallel paths: otherwise functionality will differ.

So, I will move into the next technique which is called adding extra register layer. So, retiming is something you are not adding extra registers. What you are doing you are just movie your registers, in your design to improve reduce the critical path length.

So, here what we are doing? We are instead of moving we are adding some extra register layer right. So, if you just consider this design.

(Refer Slide Time: 29:10)

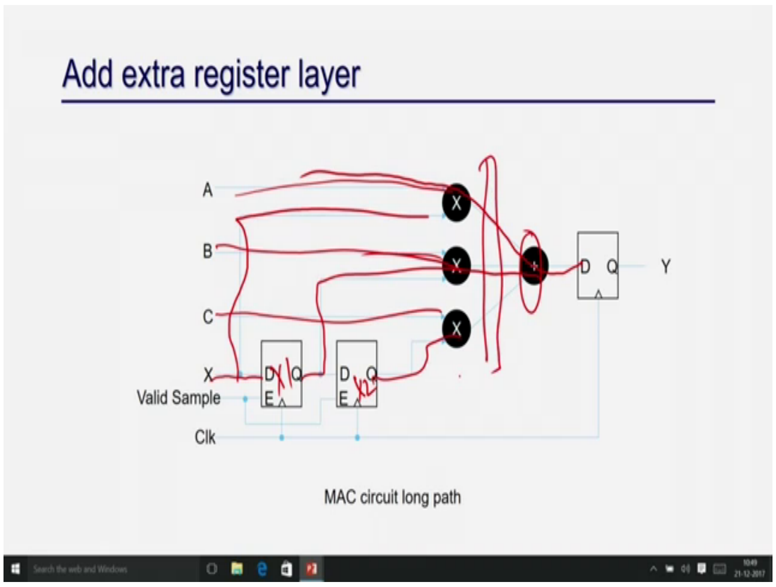


Add extra register layer

```
module FIR(
        output[7:0] Y,
        input [7:0] A,B, C, X,
        input clk,
        input validSample);
        reg [7:0] X1, X2, t;

        always @ (posedge Clk)
                if(validSample) begin
                        X1<=X;
                        X2<=X1;
                        Y<=(A*X)+(B*X1)+(C*X2);
                end
end Module
```

So, suppose I have this f I r design what I calculating this A X into B X 1 into C X 1 where this X just shifting to X 1 and X 2 right. So, this is something which is happening here.

(Refer Slide Time: 29:21)



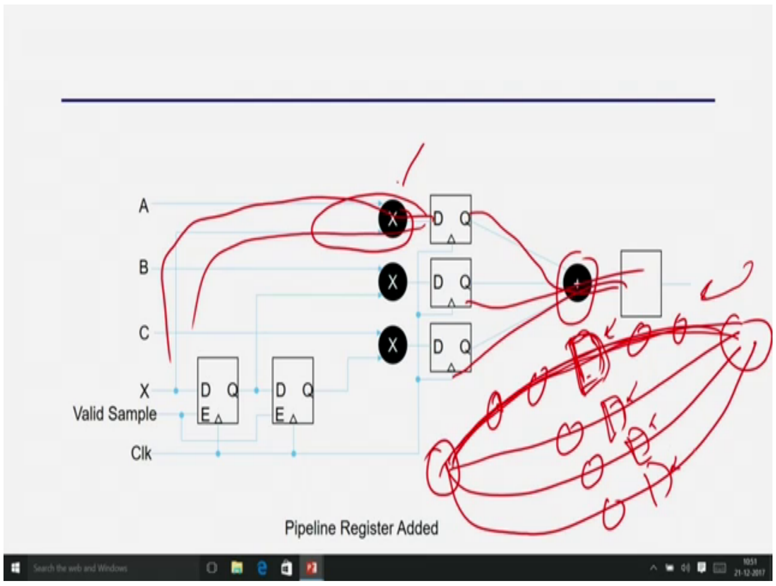
Add extra register layer

MAC circuit long path

So, this is the circuit. So, this is my X 1 this is my X 1 and this is my X 2 and this X 1 is coming here it is storing here. So, I am calculating A into X here, where this B into X 1 here and then C into X 2 here right, but these 3 are all in parallel and then I am just adding them up right.

So, this is what I am doing here. So, but the critical path length is what I have a multiply adder and a multiplier that. So, all path is has one multiplier and one adder right. So, the critical path length is something is like this. So, what I can do I can just add just one.

(Refer Slide Time: 30:04)



Pipeline Register Added

Extra layer here. So, that is what I have done here. So, that the critical path is now the layd of the maximum of one multiplier right. So, this has one this path has only one register adder, but this paths have one multiplier and usually multiplier is more complex than adder. So, the critical path will be determined by the delay of this multiplier.

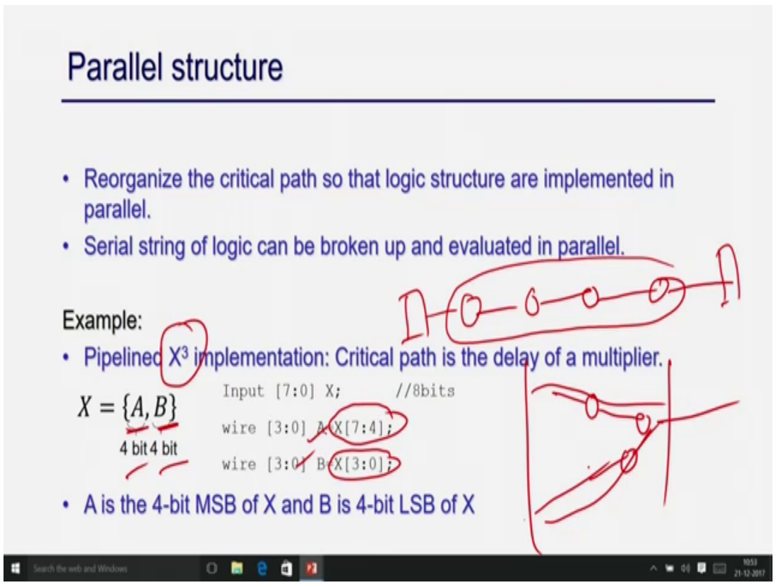
So, you can understand here that if you just add register your latency will be improved by it will be improved by increased by 1 right. So, but that has to been taken care of. So, its not that if you if this is something is actually changing your functionality, but you have to synchronize your design with the other component, that your data will come after one cycle earlier if it is coming after 5 cycle now it is going to take 6 cycle to computer this thing because I am just adding one extra layer right. So, you have to just take and take care of that particular factor. So, that you can actually synchronize your whole other you can design synchronize this part of the design, with the other component of your design ok.

So, this is something you can do, but again there are other factors that has to been taken care right. So, what I just say that adding register layer to improve the timing by specifically dividing the critical path into smaller path, but your you have to basically it is improve the latency right improve the latency so; obviously, it is going to increase because you are adding it to your design, and is basically its not be affected if you have to just, but you have to be careful about this. So, when you are adding a layer, you should add in all parallel forces that is what I am mention.

So, if you just add here not here then its a problem right. So, suppose if you are say there are 4 parallel path from the input to outputs right. So, this is my input and this is the output and you can add one register here because this is very critical if you want to add a register here you have to add in all parallel path even if this is say there are 4 multiplier in this right. So, if you assume that there are 4 multiplied here, and this path are simpler I mean I have say one multiplier right. So, you just I have to add latency and I will just add in between and I am done then this is wrong because now that then there is a if you just add one latency here, then this data will be delayed by one, but this will all come in as earlier right

So, then the problem what will happen this is not synchronized. So, the idea is that when you are going to add extra latency, it has to be in the all parallel path. Even if it is not required for this path because you need only this path, but you have to add in parallel path to keep your functionality intact of your design. So, this is the factor that has to be taken care otherwise your function it will be differed right. So, this is something the factor that has to be considered and also as I mentioned that whenever you are reading this extra layer you have to take care of this latency it is increased by one. So, have to if your block is synchronized with other block that has to be taken care right. So, this is these are the 2 factor you have to understand, but using adding register you can actually improve your timing right.

(Refer Slide Time: 33:05)



Parallel structure

- Reorganize the critical path so that logic structure are implemented in parallel.
- Serial string of logic can be broken up and evaluated in parallel.

Example:

- Pipelined X^3 implementation: Critical path is the delay of a multiplier.

$X = \{\underbrace{A}_{4\text{ bit}},\underbrace{B}_{4\text{ bit}}\}$

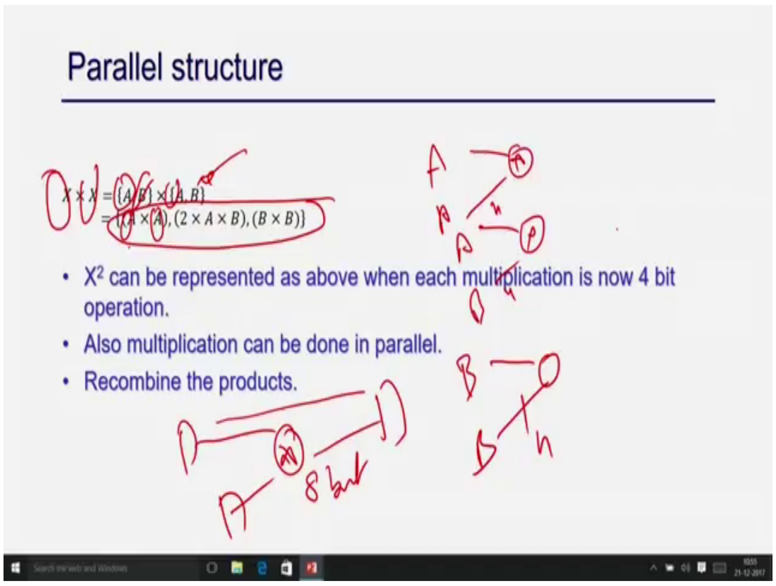
```
Input [7:0] X;      //8bits
wire [3:0] A=X[7:4];
wire [3:0] B=X[3:0];
```

- A is the 4-bit MSB of X and B is 4-bit LSB of X

So, that something is one of a one technique which you can consider ok. So, next the topic is like parallel structure. The parallel structure is something if you are computing something in sequentially right. So, that will be added up right you to just starting from there are say 4 modules and you just do this component. So, you are. So, this is one registers and this is one registers. So, the computation the computational delay of this whole 4 unit will be added up to and which will determine your clock speed right, but you can in certain time you can change your design as a whole, its not like that you can do something here you can, but you can change you are a logic little bit so that you can actually paralyze these things.

So, for example, you can just maybe you can do something like this here. So, that the delayed will become maximum say 4 now ok. You mean sorry earlier it was four. So, suppose we can do something so, that your delay will become now 2 maximum 2 right. So, you can do certain kind of optimizations, but this is you are your design choice and this is the expertise of the design right its not possible in all places, but certain time you can do that. So, for example, if you just take the same x s cube power 3 example when your input is X and now we can think about. So, my X will actually put in 8 bit, but we can think of what that X is has 2 components, this is the 4 m s bit 4 bit and LSB 4 bits right and we can do and we can store this MSB 4 bit into register to A and lower LSB into register B and we can do this sum multiplication separately right.

(Refer Slide Time: 34:36)



Parallel structure

$$X \times X = \{A, B\} \times \{A, B\}$$
$$= \{(A \times A), (2 \times A \times B), (B \times B)\}$$

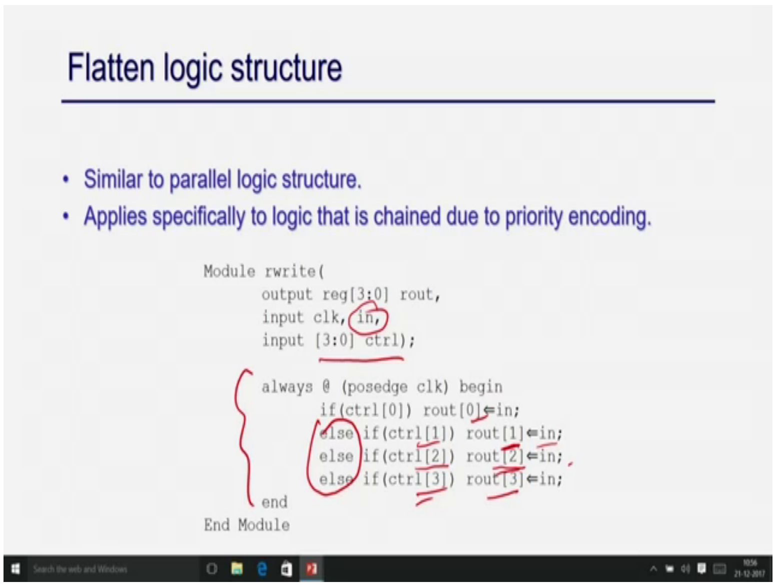
- X^2 can be represented as above when each multiplication is now 4 bit operation.
- Also multiplication can be done in parallel.
- Recombine the products.

So, that detail I am not going into detail we can computation refer some other places, but the idea is that now your X into X is nothing, but this 2 component and you can do this multiplications of the smaller unit, this is all 4 bits right you can do and in combined these results to get this results X square right. So, this is possible. So, the basic idea is just try to understand give you the notion here is that instead of doing this 8 to 8 bit multiplication, I can now do 4 bit multiplication and all these 3 A into A, now this A into A and this A intoB, this is A, this is A B, and this is B B this can be done in parallel and these are all 4 bits; earlier what I am doing here.

So, the example is source. So, earlier what I am doing here. So, you are just doing a multiplication here, which is 8 bit right though there is you can feel I have only one multiplier, but the delay will be determined with the computation computational delay of this multiplier, but now what I am doing instead of this 8 bit multiplication, I am doing 4 bit multiplication. So, all are 4 bits which will be the computational delay; obviously, less compared to this 8 bit and I am doing all this thing parallel and I just combine these 3 results to determine the output result right.

So, this is something can be done, this is just to give you the idea that in certain time complex and can be splitted into smaller component, and smaller operations and combining this smaller operation results can generate the actual output and where this smaller operations can be done in parallel right. So, this is the objective of giving this example without going into detail right. So, this is kind of we can paralyze your structure to improve your timing this is one more technique.

(Refer Slide Time: 36:15)



Flatten logic structure

- Similar to parallel logic structure.
- Applies specifically to logic that is chained due to priority encoding.

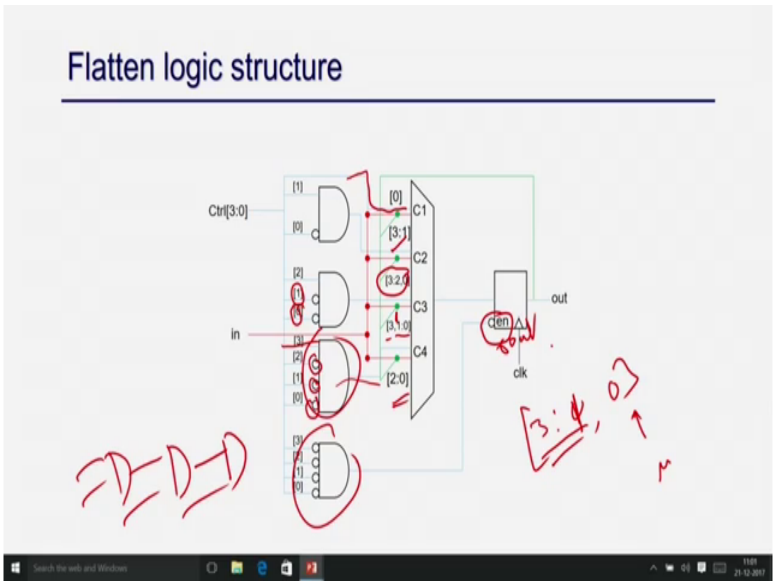
```
Module rwrite(
        output reg[3:0] rout,
        input clk, in,
        input [3:0] ctrl);

        always @ (posedge clk) begin
                if(ctrl[0]) rout[0]<=in;
                else if(ctrl[1]) rout[1]<=in;
                else if(ctrl[2]) rout[2]<=in;
                else if(ctrl[3]) rout[3]<=in;
        end
End Module
```

The next technique I am going to discuss about is the flatten logic structure. Again some time we write something without understanding how it will be finally, implemented in hardware.

Let us take an example like this where you just write key if control. So, you have input say control 4 bit control, and your actually setting some register bit based on this set the control bit 0 then you are going to you, you put in one beat right. So, you are you are storing this input to the 0 bit of the route right. If it is one then you are going to update this input the bit 1 of the output similarly if it is control bit 2 is 1, then you are going to update the route 2 the third bit of the register and similarly this it is allowed 3 v other bit will be remain as it is it right

So, if we just write this is very obvious this else if you are writing like this right, but if you just think about what is this equivalent hardware implementation you just look into this.

(Refer Slide Time: 37:17)



Flatten logic structure

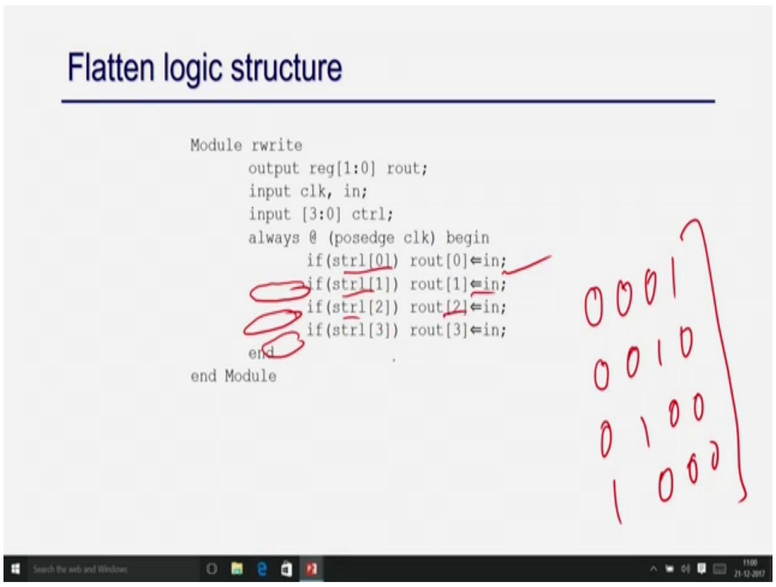
So, if your control bit is this is 0 bit. So, if it is 0 this is the 0 bit you are going to. So, you are taking this your register this is r out is going here, I am taking this 3 to 1 bit and the 0 bit from here and combining this 3 to 1 bit here 3 to one and combining this 0, I will update right. So, this we will basically I am taking this input directly this is the input the raid is input. So, input is coming. So, input is the 0 bit this is your v and I am taking this 3 to 1 as it is and that will determine this result.

So, this is what I am doing here also right. So, this I am just updating the 0 bit other bit as it is. Now if this else is true that if else if; that means, that time the control 0 is 0 right this is 0, then only this else will be true. So, I have to though now I add a and get where I have to check this control 0 is 0 that is the negation of is 1 and I am going to check the control bit 1. If it is true then I am going to take I am going to update the 1 bit. So, I am going to take 3 to 2, I am going to take 3 to 2 at 0 bit and the 1 bit will be updated from this right. So, this is the input will be the 1 bit I am combining this to get the results right.

So, similarly when this both is 0 control 0 is 0, control 1 is 0 then I am going to take out this control 2 right. So, if this is the case, then I will now check control 0 bit is 0 control 1 bit is 0 and control 2 bit is 1, then only I am going to update this third bit. So, I am going to take this 0 to 1 and 3 from this register, and the second bit 2 is going to updated by the input. Similarly when I have to check when this third bit, this third bit then I have to check this 0 is control bit 0 is 0 1 bit also 0 say second bit is (Refer Time: 39:005) and the third bit is 1. So, I need big adder to just to realize this right because its all possible computation is coming and then I am just going to update the third bit.

So, 0 to 2 I am going to take from the r out and third bit I am going to take this as the input right and there will be fourth case as well. That all of them are 0 right then you have to just copy paste you are not going to if this is the become the enable of the design right. So, if this is the case then I am not going to update this in data right. So, if it this is something, in the implementation of the circuit in hardware right and you can understand these are the bit adder is happening here right. So, we usually when you are writing code we do not consider about this putting this else, but it may be the case that in your design your control can maximum say kind of decoder right.

(Refer Slide Time: 39:55)



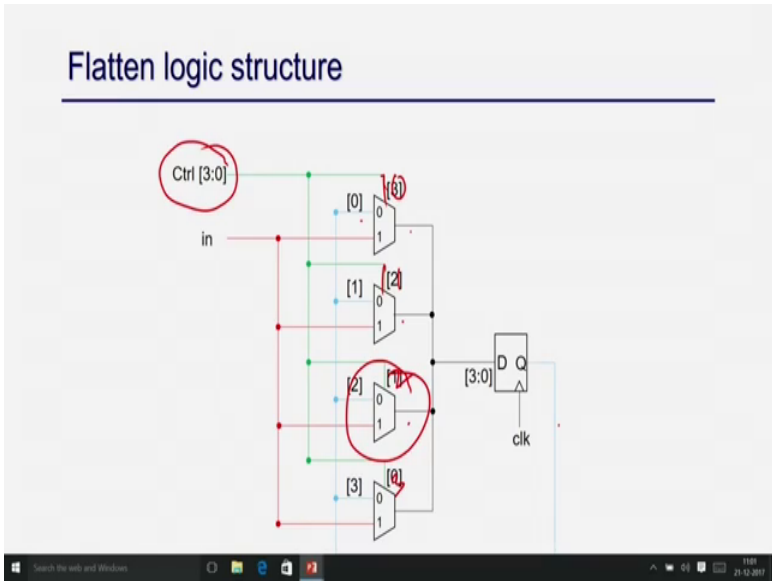
Flatten logic structure

```
Module rwrite
        output reg[1:0] rout;
        input clk, in;
        input [3:0] ctrl;
        always @ (posedge clk) begin
                if(strl[0]) rout[0]<=in;
                if(strl[1]) rout[1]<=in;
                if(strl[2]) rout[2]<=in;
                if(strl[3]) rout[3]<=in;
        end
end Module
```

So, only one bit will be the 1 right among this control 0 1 and 3, it can have only this right. So, this control 0 0 0 1 or 0 0 1 0 0 1 0 0 or 1 0 0 0; if you know this in this case even if it this is your design. So, your control bit can be its a kind of a decoder only maximum at most one on the bit is one then if you write this is your actually if you write this code this is actually generating a lot of redundant extra adders rights. So, these are all big adders unnecessary these are unnecessary because you know that your design is actually a maximum one of them it is 1 and you can rewrite code is like this right if there is no else here you can see that there is no else here, and that is a big improvement in your design.

So, if this control bit is 0, then I am going to update 0 bit it if is 1, then I am going to update 1 bit if it is 2, I am going to update 2 bit and if it is right then the equivalent circuit will be like this right.

(Refer Slide Time: 40:45)



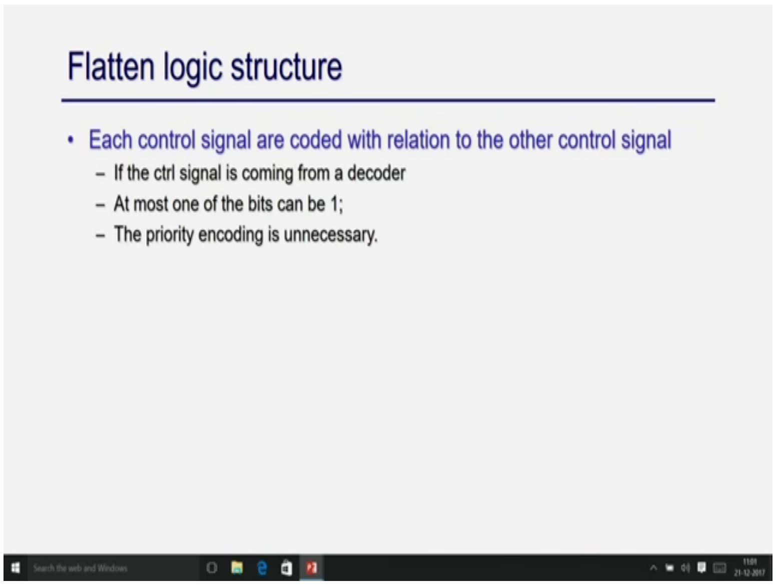
Flatten logic structure

This my this output is going here. So, this is the third bit, second bit, one bit and 7 bit and you this control is the signal of this right if this sorry. So, this should be 3, this is 2, this is 1, this is 0 ok. So, if your third bit is one then it is going to choose this one right. So, this it will automatically choose the third bit as one put here, and all other will be in the original bits right.

So, only maximum one of the bit of this register will be updated by the second because my know this my controller is this is decoder. So, either the 0 bit will be updated or the one bit will be updated your second bit will be updated or the third bit will be updated right. So, we can understand if you are you know that it is a decoder circuit and your circuit is not something that all of them in 1. Even if you write this code this is correct there is nothing problem, but its generating a complex resource and computational delay is also be great because you are now have this big mux versus this error delay will also come right because this adder is something 4 adders basically a 3 right.

So, this is a adder of 3 right. So, this all is something come into picture and your computational delay will computational delay will improve, but if you know that your circuit is like this you can actually flatten this and yours now the delay of this only one mux. So, it will be improved by a lot right. So, this is something kind of flattened logic structure you know that your circuit has specific type of behavior. So, you should write this kind of input instead of writing this if else right. So, that is something will generate complex hardware and your timing will be your timing will be can be affected by this kind of coding style ok.

(Refer Slide Time: 42:36)

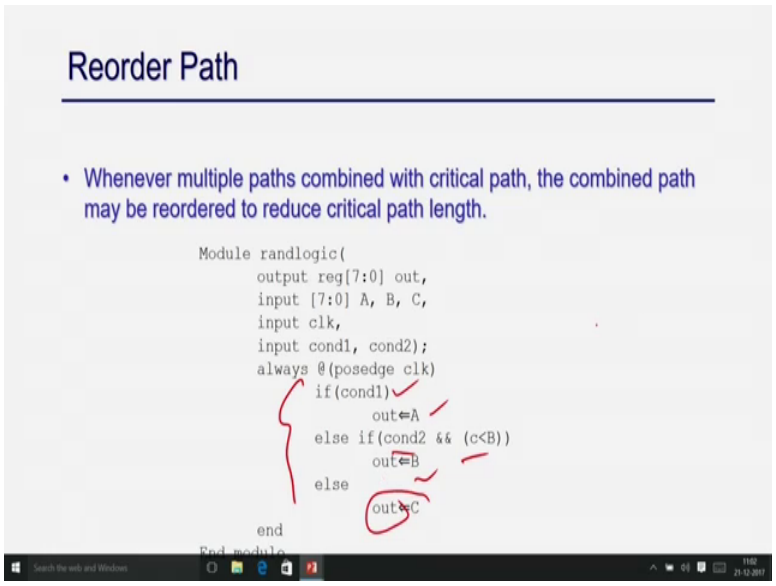


Flatten logic structure

- Each control signal are coded with relation to the other control signal
  - If the ctrl signal is coming from a decoder
  - At most one of the bits can be 1;
  - The priority encoding is unnecessary.

So, this is something you should always think about when you are actually designing an circuit, and based on the input parameters, we have to write our code judiciously. So, that u mean some are optimized hardware is not generated by the synthesis tools.

(Refer Slide Time: 42:52)



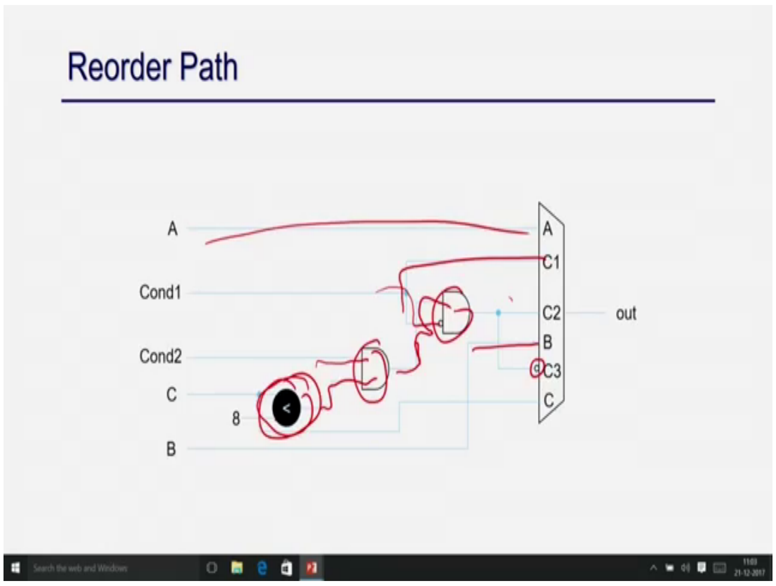
Reorder Path

- Whenever multiple paths combined with critical path, the combined path may be reordered to reduce critical path length.

```
Module randlogic(
        output reg[7:0] out,
        input [7:0] A, B, C,
        input clk,
        input cond1, cond2);
        always @(posedge clk)
            if(cond1)
                out<=A
            else if(cond2 && (c<B))
                out<=B
            else
                out<=C
        end
End module
```

So, now I am going to talk about another technique is again this is something designer choice is the reorder path. So, when your check is writing this if else kind of code some nested conditions that is actually generated in the hardware in long computational path right. So, if you just take this example if condition 1 I am choosing output equal to 1, if condition else if that is condition 1 is false and condition 2 is true and this is less than B then I am going to do this right else I am going to choose this.

(Refer Slide Time: 43:22)

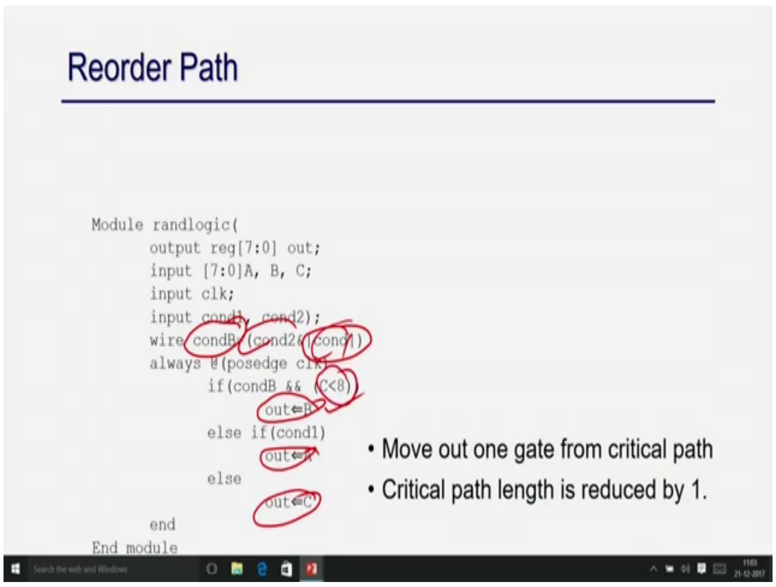


Reorder Path

So, if you just see the hardware. So, if condition 1 is true I am going to choose A. If condition 1 is false and condition 2 is true plus C less than 8, this is C less than and this through output and then this will then I am going to choose B right.

So, this will become the conditions and if this is not true then I am going to C you can see that the computational delay of this path comma involve these 3 gets right. So, this gets this gets and this gets. So, total delay of this path will be 3 and gets right, but you can actually rewrite your code a little bit different manner, because you have finally, you have to choose between ABC right. So, we can rewrite such a way that these 3 can be it will become parallel right. So, if we just. So, if we rewrite.

(Refer Slide Time: 44:05)



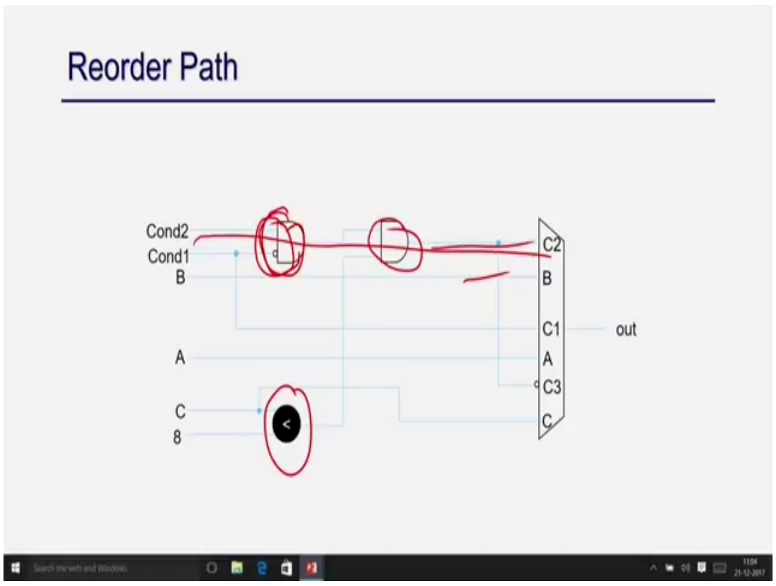
Reorder Path

```
Module randlogic(
        output reg[7:0] out;
        input [7:0]A, B, C;
        input clk;
        input cond1, cond2);
        wire condB=(cond2&!cond1)
        always @(posedge clk)
                if(condB && (C<8))
                        out⇐B
                else if(cond1)
                        out⇐A
                else
                        out⇐C
        end
End module
```

- Move out one gate from critical path
- Critical path length is reduced by 1.

What I have did? I just do this condition 1 into conditional 2 and these become condition B, and I am choosing B first. So, earlier I was using condition A first, then B and then C, I just do it now here, I just choose B first and then A and then C and what I did here actually I do this condition 2 and condition 1 versus this C less than eighteen parallel right.

(Refer Slide Time: 44:30)

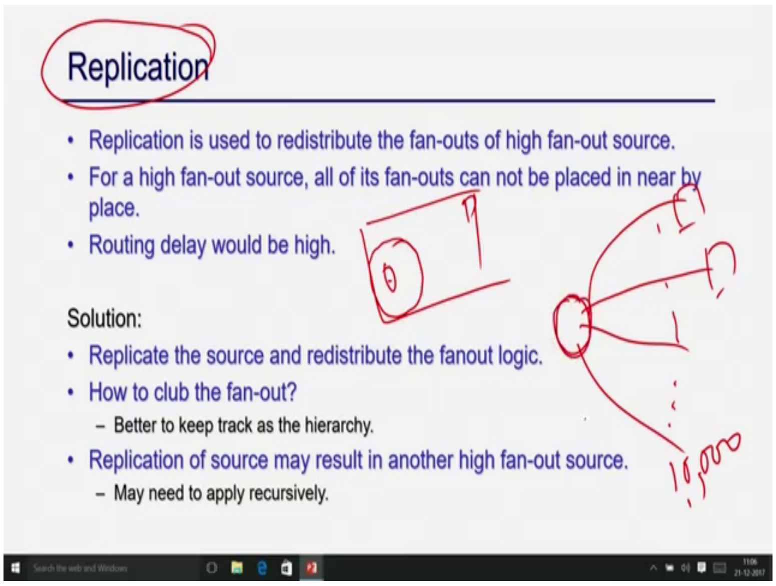


Reorder Path

So, if you just see the circuit, I am checking condition 1 condition 1 is not true of condition 2 and I am in parallel I am and checking C less than 8 and then I am combining these 2 right. So, this and this then I am choosing B. So, I am doing the same operation as this design, but I am my delay of now have maximum of 2 right and which is less than the previous circuit. So, this is all called the reorder path. So, its kind of a design that is why when you actually have a long if else structure, you should actually think about how we can reorder your this condition. So, that you can the design that critical path, that is generating will be less than the little less than the previous one right. So, as I showed in the example.

This is just a example, but if you just think about your complex logic structures some can be real big and you can actually paralyze this operation separate condition check in parallel so that you can reduce the length ok. This is what is all about reordering of paths.

(Refer Slide Time: 45:31)



Replication

- Replication is used to redistribute the fan-outs of high fan-out source.
- For a high fan-out source, all of its fan-outs can not be placed in near by place.
- Routing delay would be high.

Solution:

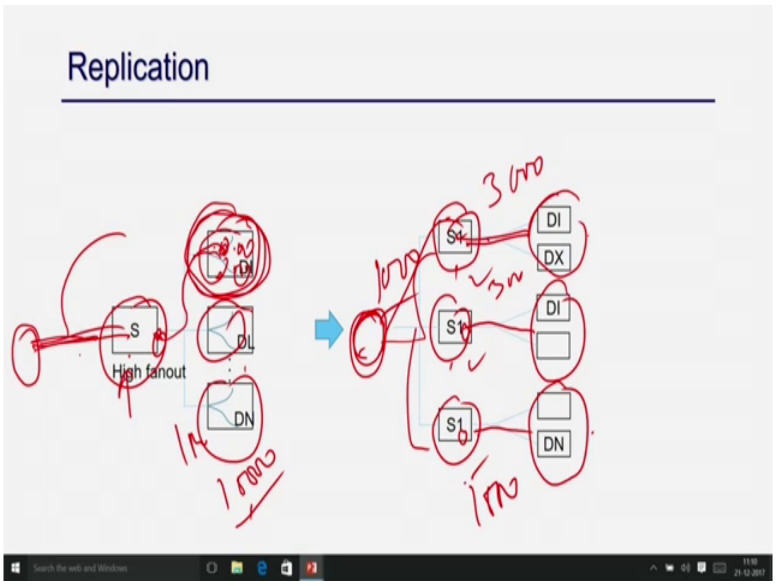
- Replicate the source and redistribute the fanout logic.
- How to club the fan-out?
  - Better to keep track as the hierarchy.
- Replication of source may result in another high fan-out source.
  - May need to apply recursively.

The last one I am going to talk about is the replication what is replication. So, this all the previous techniques I talked about is just to reduce that logic delay, that is the computational logic delay. The other factor that I talked about was that routing delay right. So, the replication is strategy that actually helps to improve this routing delay and what is this. So, if you just think about this high fan out note right. So, suppose from this note there is a 1000 fan outs, it is very common in design that when you go to logic level or when you this get level, and there is some source like clause and all which actually is very high fan out thing like this is other pin which actually its a very high fan outs right 10,000 10,0000 pins fan outs.

Now, you think about the placement finally, when you actually going to do the synthesis all this note will be places somewhere in the time right. So, if there is a high fan out note like this, it is not possible to place all of them nearby places. So, it is a source we cannot because the resource around this is limited, you cannot place all these 10,000 around this source note right some of them have to place in very far right and that will become the long path because there is other no choice right. So, some of the path definitely is going to be in very long and the routing delay of those path will be very high ok.

So, how we can actually improve those things and that is what is called replication what we can do, I can just replicate this source I can make it multiple source right.

(Refer Slide Time: 47:08)



Replication

So, what is I just discussed here. So, I have a this high fan out source and its going to very various designs is a 10,000, I can split this I can create multiple source zone right and what about the things input is coming I just copy to all of them. And I just split this fan outs to these sources right. So, I just split these fan outs there are multiple here I will split combine these 2 and make this fan out of this, I will combine this I just make the fan out of this one and this component I will just make the fan out of this.

So, maybe I just split a 3 and this 10,000 can be splitted into say 3,000 I just give a. So, this maybe say 10 we can copy 10 copies I just give 3, but we can actually get 10 copies and all of them have the less fan out now. And whenever there is a number of fan out is less the possibility of placing those fan out notes nearby is possible or high. So, then probably that during placement and routing your synthesis tool is able to place those components near to this place, and the routing path or the delay of that longest path is reduced and is which is effectively improve your timing ok.

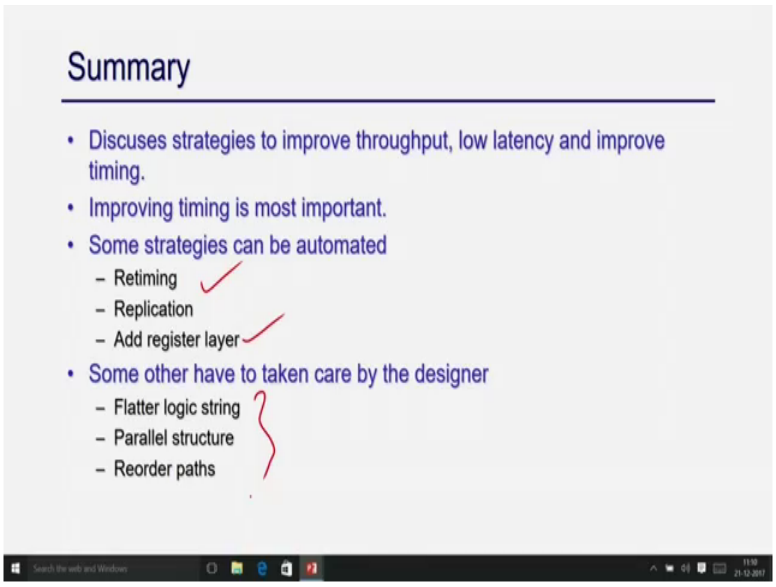
There is one factor here which you should when I have a discuss this how to club this fan out right. So, what I just talked about I just split these source into multiple source now ok. And now I will just say there are 10,000 fan outs how to club which 3,000 I should put as a output of this which 3,000 I should put as the output of this and which 3,000 I should put as the output of this source right here I should always maintain the hierarchy of your design. It is very obvious that whenever there is a hierarchy new design all these notes are little bit connected and the synthesis tool try to place those notes inside a particular hierarchy close by places.

So, if we just put one hierarchy to the same source, then there is a high chance that all these things should be placed in nearby places right. So, that is something very important as I will repeat again it is like if there are say a 100 component of this fan out is connected to this to this fan out right. Now if we connect all of them here to this source and then there is a high chance this all 100 will be placed nearby places. So, then this there will be no long routing, but if you just the way you are splitting these fan outs it maybe that this is place this component is connected to this, this component is connected to this, this component is connected to this, then your purpose is lost. Again this same hierarchies different component are connected to different sources, again they cannot be placed to near to also sources right. So, then some of the paths will be lost.

So, the idea is that when you are actually splitting this fan outs, you should keep track of the hierarchy. So, that all the fan outs in the same hierarchy you place them into the same source. So, that your place placement tool could not find difficult replacing the nearby places, and as a result of that you are routing delay will be improved right and you might ask me the same question again if you split this I mean create multiple source its not only 3 usually you do it you say 1000 right because there are 10,000 I want to make it simpler. So, I want to make it say instead of 3 I say 1000 source, then the source of this may be bigger fan out right high fan out note. Earlier I have only from this source I have only one fan out now this will be giving to all this fan outs. So, this become now 1000.

So, this something again we have to do this replication for this node as well. So, this is kind of a repetitive or say recursive way to do this right. I have to apply this recursively starting from. So, high fan out node to this source and then this source and so on finally, I find out some node which is not that big. So, this is something its kind of a we can apply during this logic synthesis, which actually improve the timing of your design ok

(Refer Slide Time: 51:11)



Summary

- Discuses strategies to improve throughput, low latency and improve timing.
- Improving timing is most important.
- Some strategies can be automated
  - Retiming ✓
  - Replication
  - Add register layer ✓
- Some other have to taken care by the designer
  - Flatter logic string
  - Parallel structure
  - Reorder paths

So, this is what is called replication ok. So, with this I am going to conclude as I discussed today we are discuss about several strategies which actually improves the either its the throughput of your design or reduce the latency of your design or primarily improve the timing of your design or you achieve higher frequencies ok. And some of the technique can be automated like replication, retiming, adding register layers, but some of the techniques like flattening logic structures or parallel structure or reorder paths is kind of a is (Refer Time: 51:41) with the designer right.

So, if you find some long path its choice of a designer one has to do it manually to improve the circuit performance, but some of them can be done automatically and its actually implemented which is implemented well in the any industry level synthesis tool. So, we are going to discuss one of them in coming classes on retiming ok. So,

Thank you.