



**Digital System Design**  
**Professor. Neeraj Goel**  
**Department of Computer Science Engineering**  
**Indian Institute of Technology, Ropar**  
**Lecture No. 57**  
**Pipelined adder design**

(Refer Slide Time: 00:18)





**Extension of multi-cycle adder design**

- Adding 64 bit numbers
  - Using 1 bit Full adder will take 64 cycles
  - Using 16 bit adder will take 4 cycles
  - Using 32 bit adder will take 2 cycles
  - Benefit?
- A pipelined design can increase throughput
  - Requires multiple adders
  - Reduce the clock period

Digital Logic Design Sequential Circuits



Single bit adder. Now what could be the other thing? So, if we want to extend this particular idea, then you see let us say, I want to add 64 bit numbers. So, what does that mean? It means that if I am using single bit or 1 bit full adder, how many cycles it will take 64 cycles. So, let us try to see that if I am taking, I am using single bit full adder I am taking 64 cycles, what is the overall latency? Let us say my 1 bit adder is determining this is the critical path. If this is the critical path, what would be the overall latency of one addition?

Overall latency the addition would be defined by 64 into delay because of one full adder plus setup time of register plus delay of a register, latency of a register. So, because of latency of the register itself the, when we see the overall latency of 64 bit addition, here, that part could be dominating, it would be non-negligible, it could be significant part of my latency.

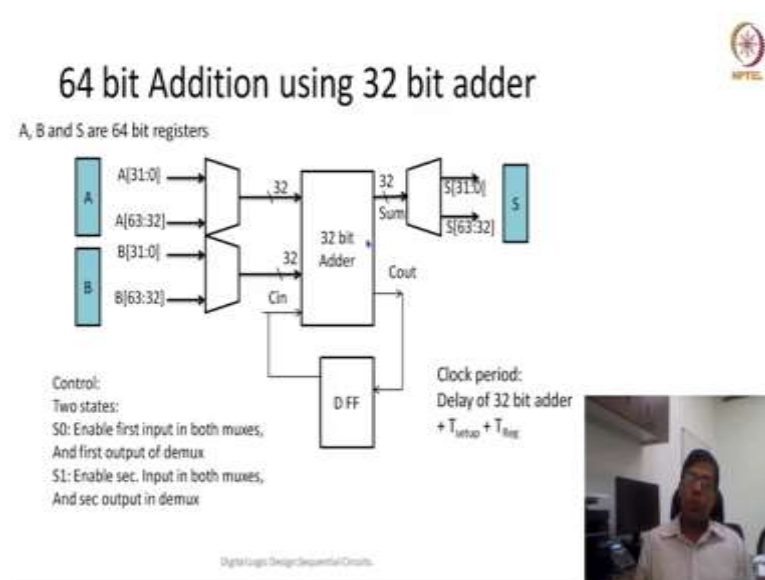
So, the advantage what we were trying to get from dividing things into very smaller pieces may not be there. So, that is why it could be the other, these things are more practical, so if we use instead of 1 bit adder, we use 16 bit adder or a 32 bit adder. So, for example, if we are using 16 bit adder, then it would take 4 cycles will first add first 16 bit the next 16 bit the next 16 bit, so this way to like take 4 cycles. And if we are using 32 bit adder, then it will take 2 cycles.

So, addition using this 32 bit and 16 bit adder is more practical, because there the fraction of time which is used by a register is also less, and the advantage, whatever advantage we want to get because of reduction in clock period can also be obtained. The other benefit, because of that could be also be that we can design a pipeline design.

What is this pipeline? So, let us try to understand what it is pipeline. So, when we would do this pipeline design, whatever it is, so when whenever we do this pipeline design, we would be able to reduce the clock cycle, but may also have the same throughput. So, what is this throughput that also we will try to understand in next couple of slides. So, the overall message here is that, we would be able to reduce the clock frequency because the size of my addition is less, along with that performance may not be impacted.

So, let us say my 32 bit edition is done using two cycles, but still what would, how good it would be that in my result is still available after every cycle. So, that means, I am able to reduce the clock frequency by half and also I am able to get my output every cycle. So, that means overall effectively I have reduced my delay by half. So, let us see with the example of this 32 bit adder. And if we would like to do 32 bit adder, 64 bit adder using 64 bit addition using 32 bit adder let us see how we go about it.

(Refer Slide Time: 03:57)



So, since I said so, basically, if we would like to add 62, 64 bit numbers, but the addition adder which we have available is only one 32 bit adder. So, what we can do is we can divide the whole number into two chunks, first 32 bits and second part of the 32 bits. So, let us say A and B are two numbers and we can say that first 32 bit is 0 to 31 and the second part is 32 to 63.

So, similarly B can also be divided into two parts. So, although, it is available as a register, so either now there are two possibilities here. If it is a 64 bit register now there has to be a mechanism so that out of the 64 bit register I can read either this part of upper half of the register as well as the lower half the position or what we can do is this A could be a mix of two registers which are two 32 bit register, so that these two different parts could be read individually.

So, both are equal possibilities, and in practically also both are feasible. So, having one 64 bit register and reading two different parts of the 64 bit registers is also equally appealing as well as equally possible than having two 32 bit registers and considering them as one 64 bit register.

So, somehow, I have to provide a mechanism so that only either the upper part or the lower part will be added, will be given as an input to my 32 bit adder. So, I have put a mux here. So, this mux there will be one select line. With this select line will say that whether upper half or lower half, this is the lower half this is the upper half, so whether upper half or whether the lower half, which one should we selected.

So, there would be two muxes here. And similarly, at the output because it is only 32 bit output, so that output has to go to a S register, which is again a 64 bit register. So, here we have to see that there is a D mux here D multiplexer. So, this D multiplexer will decide whether the upper part or whether the lower part of this of this sum, this S register would be written.

So, now, because we have to find, which part would be written so there will be a D mux here. So, if D mux means that whether this upper part or the lower part, and we also have to select in the S accordingly that which part do we written. So, since, after first edition my Cout has to go to a Cin of the second half of the edition, so This D flip flop is still required. What about the control? Control would be here we can see that we can we would have two states. Let us say S0 state and S1 state.

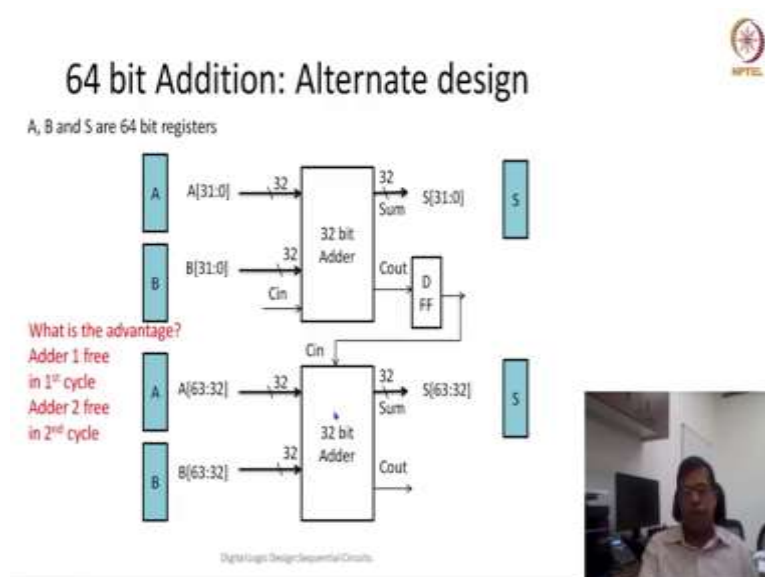
So, whenever we are there in S0 state, then upper half. So, this lower half would be selected A 30 0 to 31 B, 0 to 31 would be selected and the output would be given to S 0 to 31. So, when S1 is there, then we will enable the second part. So, the second input or the mux would be selected and the second output of the D mux would be selected. So, after second state we can also make this clear the D flip flop so that can also be done.

So, so, this is one particular type of design where we this this addition is done using two clock cycles. So, if we say what is the clock period, clock period would be the delay of 32 bit adder plus  $T$  set of time plus  $T$  write time. Now, if we compare the latency of this 64 bit additional. Latency means, let us say this 32 bit adder is in the critical path then what would be the latency? Latency would be twice the delay of 32 bit adder plus twice setup time plus twice the register time.

So, let us say these two are negligible in comparison to delay of 32 bit adder then we can say that the latency is essentially twice the 32 bit adder delay. And if we would like to compare this particular design with a ripple carry design of my 64 bit edition then we can see that this design is almost as good as 64 bit ripple carry adder with half the area. So, my area is reduced to half because I am using only 32 bit adder not the 64 bit error. So, area would be clearly the half, and my delay is almost the same.

So, this is a very good design trade off that if we would like to reduce the adder area by within the same cost then this sequential design which is a multiple cycle design could be quite effective.

(Refer Slide Time: 09:49)



Now, let us see the another trade off point. So, let us try to use 64 or basically two instances of 32 bit adder, and this is the alternate design. So, here, no muxes are used. No multiplexers are used, but the design is something very similar. So, we are using two 32 bit adders only there is no mux here so that means only the upper lower half of A and B would be added here and the upper half of A and B would be added in the second adder.

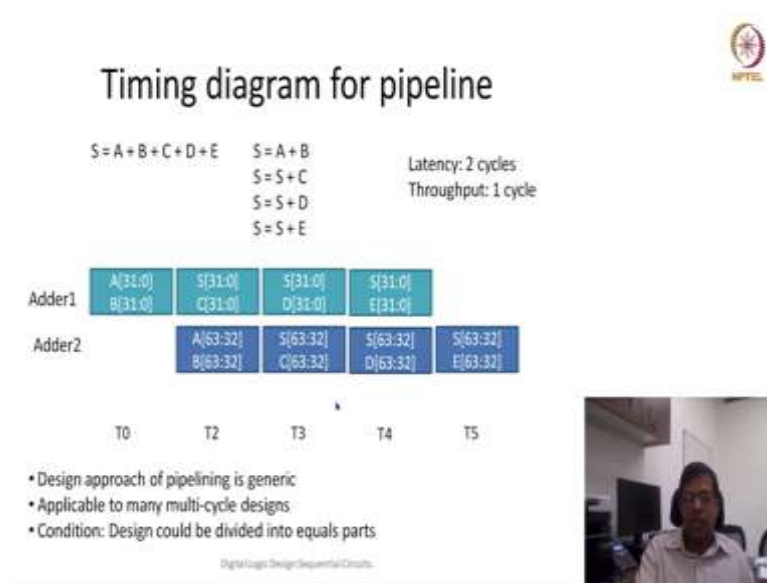
So, the output here, we have to understand that is again a clock design. So, basically in cycle 0 this is being performed in cycle 1 this is being performed. So, the result here would be stored in S lower half the S, here result would be stored in the upper half of the S. So, the output here is a stored using a D flip flop, and it would be available to this second adder for rest half of the cycle.

So, if we see this kind of a design then you will say what is the benefit? Effectively, the area is seen as 64 bit ripple carry adder with the additional one flip flop. These registers are we assumed that it would be still required in both the cases. The other interesting observation here is that my adder first adder this adder is free, is occupied in zeroth cycle and this is free in the zeroth cycle. And in the first cycle this adder is free, and this is occupied.

So, only one adder is working at one point of time. So, is it an advantage? It looks like it is a disadvantage that we have some resources but they are laying idle. There is no utilization of that resource. How can we say that this is an advantage? So, to make this thing into an advantage, what we can do is that as soon as the first edition is done, our first part of the addition is done we can use this 32 bit adder to add some other number if they are there.

So, if we want to add some other numbers then this 32 bit adder, first adder can be used to add that number. And in the next clock cycle it can give the result here. So, this carry out would be used here and then we can that the second part of that computation could be done. I can understand it is not clear. So, let us try to understand it with an example.

(Refer Slide Time: 12:44)



So, let say, I want to add these five numbers  $S$  equal to  $A$  plus  $B$  plus  $C$  plus  $D$  plus  $E$ . Now because we have adder, which take only two inputs, so this addition can also be written as  $S$  equal to  $A$  plus  $B$  then  $S$  equal to  $S$  plus  $C$  then  $S$  equal to  $S$  plus  $D$  then  $S$  equal to  $S$  plus  $E$ . So, some of you may question that why we are not using a tree.

So, if we are using three addition then it would require multiple adder, but let us say for currently we have only one adder, which is there with us only single adder is there, so then we would be able to do it in 4 different steps. So, when we doing things in 4 different steps, so, let us say what would happen?

Let us say we have these adder 1 and adder 2 they are represented here, and now in cycle 0,  $T_0$ ,  $A$  and  $B$  lower half of  $A$  and  $B$  is added. Now, in cycle number 1, upper half of  $A$  and  $B$  is added. So, by the, this is  $T_1$  not  $T_2$  this is  $T_1$ . So, in this  $T_1$  this is added and along with that, because we said it is this this particular adder 1 is free, this adder 1 is free. So, here what we can do is, we can start computing  $S$  plus  $C$ . So, you see that at the end of this clock cycle, my  $S_0$  to  $31$  is ready, it can be used, it can be because it is saved in a register so it is available to be used. So,  $S_0$  to  $31$  is available and we also start adding the  $C$  number lower half of the  $C$  number.

Now in the next cycle, this computation, the second part of this computation would be done. So, when we are doing the second part of the computation  $S_{32}$  to  $63$  would be given as input, and the upper half of the  $C$  would be given as a input. So, you see by the end, this  $60$   $32$  to  $64$ ,  $32$  to  $63$  will also be available, so, which could be used in this  $T_3$  cycle. So, now, again, my adder 1 is free, because adder 1 is free, and I have already computed  $S_0$  to  $31$ . So, what I can do is, I can start adding the third, start doing the third addition.

So, when I start doing the third addition then I will take the lower half of the  $D$  and also the lower half of my  $S$  and then the as a in the next clock cycle by the next clock cycle. So, this this would be available. And because this is already available, now, adder 1 is again free, so, we can start the fourth edition, which is  $S$  plus  $E$ . So, lower half of this fourth edition we have started.

Now, by the time  $T_5$  will come, this competition would be done. So, how much time this whole competition has taken? H much time this whole competition has taken 5 clock periods. So, there were 4 additions, which were done. And total amount of time taken was 5 clock cycles. So, can we say that one addition is done on an average in 1 clock cycle? This is what

we call throughput. So, if we have  $n$  number of tasks, how much tasks, how much amount of time we take to do  $n$  task is called throughput.

So, let us take a quick analogy. So let us say you were given 5 different assignments. One for digital logic design, one for computer architecture, one for mathematics, one for electronics, so let us say 4 different assignments and everything was given one weeks of time. So, one possibility is that you are doing all of them one by one and then finishing it off. The other thing could be that because you were spending some time parallelly whenever your brain was free or when you have put your computer on work and some simulation was going on till the time your computer was busy or computer was doing some computation by the time you were doing some writing job.

So, this kind of thing is called parallelism. So, this way because or multitasking, so your mind is a one but it is doing multiple tasks, multiple tasks at same time. Although, you can take now, if you are able to finish all these four assignments in 4 days so that means the overall latency is going to be 4, but throughput is going to be on an average you are saying that you have finished one assignment per day, correct. So, this throughput is also an important measure for performance. And here we clearly see that throughput for such kind of an adders are quite good. This is called pipelining.

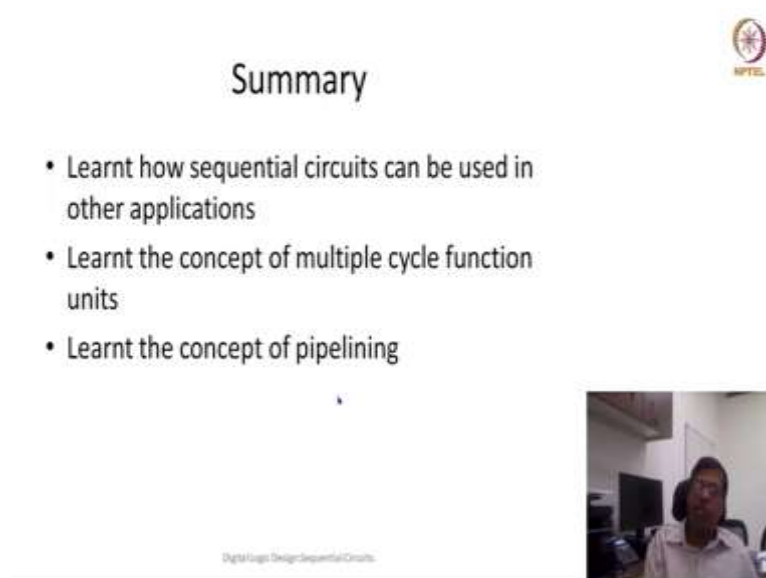
This is quite a popular process in industry more in the manufacturing, so they call it pipelining. So, basically, let us say you are manufacturing cars so one person would be designing tire another would be designing chassis another would be designing body. So, as soon as one is done, he will pass it on to other than whenever second is done he will pass it on to other. So, this way, this kind of a pipelining goes there.

Now, this pipeline is quite a generic and we see in in our daily life also. So for example, when you go to a gol gappe wala then also he will give it to one then he will give it to second then he will get it to third. So, by the time one will finish then he will give it again. So, this is a very generic concept could be applicable in many real life applications and also applicable in many other digital design circuits also. So, it is more applicable when the design is multi-cycle.

So, if it is multiple cycles then you can see that in one cycle we do this in the second cycle we do this, and then we can by the time second cycle is done the some new operant can come and can start operating. The only condition here is or the best possible, best possibility when pipelining will give best benefit when all the tasks are equal in terms of time.

So, here this addition and this addition both were taking same amount of time. So, it was almost perfect that the total throughput is 1 cycle and total latency is also, latency is almost double, but throughput is actually the throughput is also double. So, basically we are able to perform 1 task in 1 cycle or 1 addition in 1 cycle.

(Refer Slide Time: 20:35)



Summary

- Learnt how sequential circuits can be used in other applications
- Learnt the concept of multiple cycle function units
- Learnt the concept of pipelining

NPTEL

Digital Logic Design-Sequential Circuits

So, with this I would like to close today's lecture. So, in today's lecture, we have seen a couple of new interesting concepts. So, one thing we have seen that, how sequential circuits can be used in other applications like additions.

Now, addition we have done as a combinational design. Today, we have seen the same addition in a sequential design. And when we did that in a sequential design, it became a multiple cycle design. And in this multiple cycle design, we also have to design our own state machine where we clearly define that what would we done in stage 0 stage 1, stage 2stage 3 etcetera.

And also we have seen that how pipelining could be quite an effective in in digital circuits where it can help us in increasing the performance by increasing the throughput while keeping the latency almost the same. With this I would like to close. Thank you very much