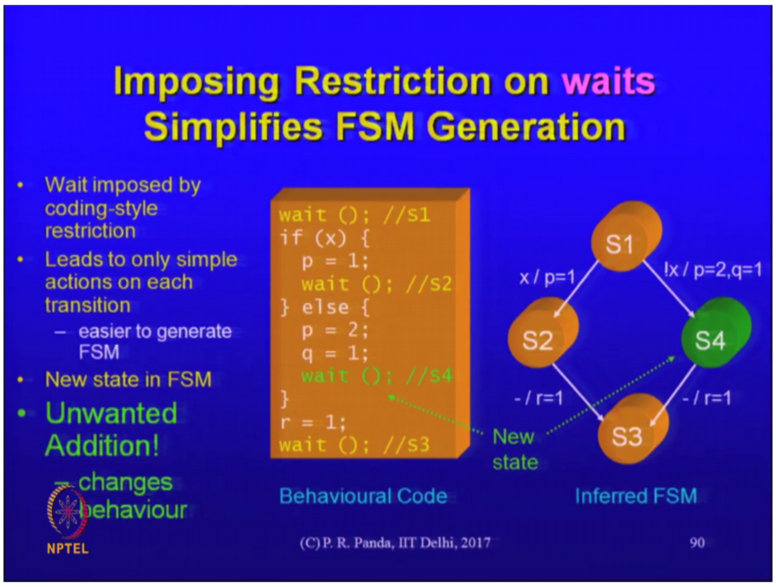
Synthesis of Digital Systems

Dr. Preeti Ranjan Panda
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 15
High Level Synthesis and Timing Issues

We were on this topic of handling I O and how that might impose some external constraints on the way we do synthesis. It might translate to some behavior such as some of the operations that we have in our design, may be attached to some clock cycles.

(Refer Slide Time: 00:53)



Imposing Restriction on waits Simplifies FSM Generation

- Wait imposed by coding-style restriction
- Leads to only simple actions on each transition
  - easier to generate FSM
- New state in FSM
- Unwanted Addition!
  - changes behaviour

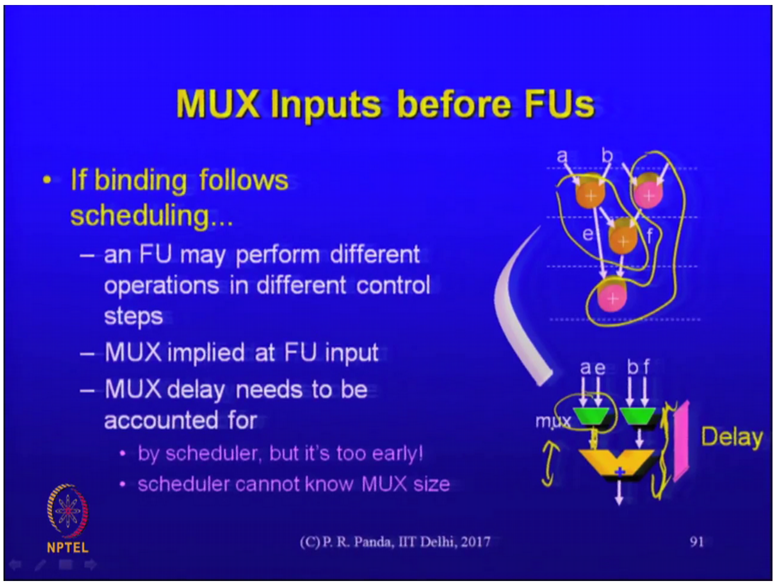
```
wait (); //s1
if (x) {
  p = 1;
  wait (); //s2
} else {
  p = 2;
  q = 1;
  wait (); //s4
}
r = 1;
wait (); //s3
```

(C) P. R. Panda, IIT Delhi, 2017 90

For other operations you have the flexibility, but some operations are attached. As you can see that can be handled in an overall methodology by fixing those schedules for some of the operations.

For the rest of the operations you still carry out the normal resource constrained or time constrained scheduling and so on. But these implications are there, which may limit some of the flexibilities that the scheduling tool has for example, the original flexibility we said which is that any operation can be scheduled anywhere that might be removed because of these constraints.

(Refer Slide Time: 01:33)



MUX Inputs before FUs

- If binding follows scheduling…
  - an FU may perform different operations in different control steps
  - MUX implied at FU input
  - MUX delay needs to be accounted for
    - by scheduler, but it's too early!
    - scheduler cannot know MUX size

a
b
e
f
a e
b f
mux
Delay

(C) P. R. Panda, IIT Delhi, 2017 91

But let us move on to another very important limitation, and what to do about it this has to do with timing. Let us see what are the timing approximations, we have made so far, in our formalisms.

We did the scheduling problem, we did the binding problem of various kinds, but the register allocation is what we looked at. But it is the scheduling that sometimes has to be refined in some ways, because of the timing approximations that were made what kind of approximations? First of all in the interest of simplification, we said we first do scheduling and then do function unit binding.

This sequence automatically imposes restrictions of nature that let us take look at. So, a nephew can perform different operations in different control units. So, here we have done the scheduling, and then there is a few binding step was taken which mapped these two additions into the same resource and there is other two additions in two right different resource. And these were done in sequence first we did the scheduling step and then we did the binding step.

This implies some kind of a muxing structure the moment you have a sharing of resources in the general case, this will lead to a mux at the inputs why would it lead to a mux because the operands may come from different places for that same resource in different clock cycles. So, in the general case if I have these two operations scheduled into two different clock cycles, I might have a 2 to 1 mux at the inputs of the adder this is one of the adders.

But for the other adders also I would have a similar sharing. So, mux is implied. Mux is implied means that there is a delay due to the mux right there is a combinational delay. And of course, we have to account for the delay, where do you account first what does it affect? If the delay now the combinational delay that I am concerned about is not limited to my adder delay, but it also has a mux that is connected in sequence, then the real delay of the combinational; this is the combinational path right from the mux I have a combinational path this is leading to the atom.

So, the delay would be the sum of these delays here that much has to fit into the clock period. But remember what was the assumption we made in the scheduling phase. As we were doing the scheduling, we were just checking whether the operation fit into the clock period or not. We looked at the added delay and we were checking how much did it measure compared to the clock period. If the delay was more than a clock period then we would say we will divide and take the ceiling and that makes an integral number of clock cycles. If it is much less than a clock cycle then in fact, we could accommodate multiple operations into the same clock cycle.

Either way we did not account of this implication of mux that is the result from the sharing why do not we account for it? Because we did not know it is too early during the scheduled phase to make that decision that decision in fact, sharing decision is being taken later on. So, the mux is inferred only later on, but that information needs to be provided to the scheduler in order for it to generate a correct scheduling.

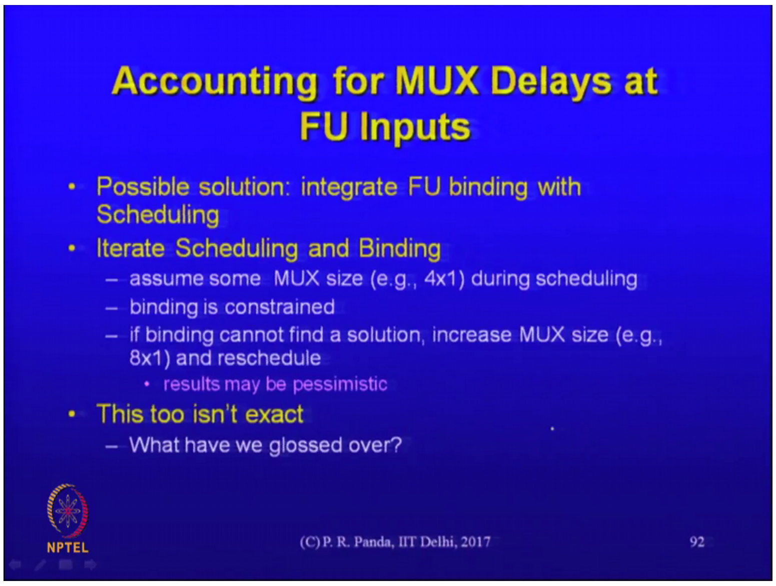
Otherwise you may have an incorrect schedule right if you can easily see that if this delay, delay of the adder was very close to the clock period then adding the mux will actually lead it to overflow into the next clock period. So, that is essentially an incorrect design that the scheduler would result in. This is an important restriction right, but when how did it arise and what can we do about it.

If we separate out the steps of the synthesis into one scheduling phase one binding phase function unit binding phase, that was done in the interest of simplifying the engineering somehow, but that leads to issues of this nature or what could we do about it?

Student: (Refer Time: 06:24).

We could in some ways combine both the solutions to the scheduling and the function unit binding. It makes the solution a little messier because if you look at the algorithm for scheduling it was formulated in some way, and it was solved in some way.

(Refer Slide Time: 06:50)



Accounting for MUX Delays at FU Inputs

- Possible solution: integrate FU binding with Scheduling
- Iterate Scheduling and Binding
  - assume some MUX size (e.g., 4x1) during scheduling
  - binding is constrained
  - if binding cannot find a solution, increase MUX size (e.g., 8x1) and reschedule
    - results may be pessimistic
- This too isn't exact
  - What have we glossed over?

NPTEL (C) P. R. Panda, IIT Delhi, 2017 92

The problem formulation for FU binding was totally different, and that was solved in a different way. Now to do them together does complicate the solution you could do it, but it makes it more complex.

You could still retain the structure, the sequential structure of these steps doing scheduling and binding, but we could pass some information or enhance that problem statement in some ways for example, you assume first there is a fixed mux that is there, at the input of every function unit like a 4 to 1mux. I do not know as it is if the whether it is 4 to 1 or 2 to 1 mux is done is realized only much later.

But for the purposes of generating a correct schedule, we could assume that the delay of the function unit is not merely the combinational delay within the function unit, but also if you add the mux at the input. The problem with that is I do not know the width of the mux. So, let us assume that it is a 4 to 1 mux just so that we have a correct design, not necessarily optimally you can see that this kind of things give up on efficiency and optimality, nevertheless correctness is more important as we have seen earlier.

So, let us assume a fixed mark size let us assume that all the function units have such fixed muxes at all their inputs. So, that at least the analysis is easier in some ways. The way to analyze now would be that I just extend the delay of each of the function units and then do the regular schedule. But this could be an iterative process because the binding is now constrained. If I say there is a 4 to 1 mux at the input, now when I go to the binding stage, I should not allow more than 4 paths simultaneously of course, to the unit well not at any point in time there is only one path but I should not allow more than 4 connections to the input.

If I impose that kind of a constraint, it is possible that you actually do not find a solution binding, is not able to find a solution. If it does not then maybe I can increase it make it 8 to 1 or something. Just in the interest of speed there is no reason why after 4 to 1 you have to go to 8 to 1, but then in the interest of reducing the number of iterations you could do such a thing of course, this process will converge somewhere. If it converges at 8 to 1 you could further try to extend the search and maybe you find the best solution which is 6 to 1 you do not need more than as six level sharing.

But such as an iterative process may be needed to get around this, but the important thing is that this approximation is made as we treat scheduling as a different step and functionally binding as a different step, because the results of their scheduling of course, need to be correct from a timing point of view.

Student: (Refer Time: 09:59).

Wiring delay is also an issue, but we will talk about the wiring delay. This is an issue that comes up independent of whether there is a wiring delay. Even if wiring delays are all zero then too this problem shows up. So, in some ways this is a different problem. That could be solved if you have a tighter integration between the scheduling step and the few binding step, you do not need to have accurate wire delays which is a totally different sort of analysis from this kind of an analysis. This solution still works at a high level; you do not need anything more it is just that there is an interdependence between the two high level steps.

But that is a little messy if you want an exact solution, but you could approximate the process by introducing some fixed number of iteration between these two steps.

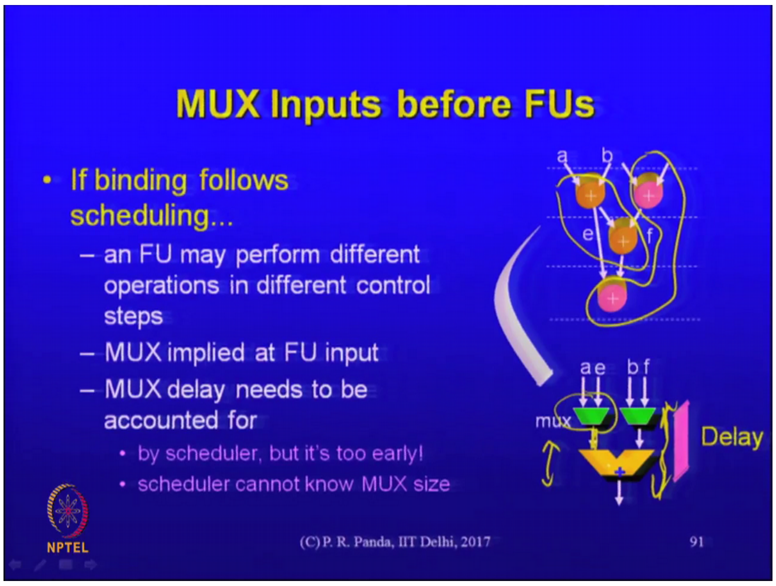
Student: (Refer Time: 10:57) we increase all the muxes (Refer Time: 10:58).

That is the pessimism we are talking about, because you do not know the sharing until much later. So, you put that mux at the input of every function unit, it is a pessimistic solution. You are trying to get a correct solution first, later on perhaps you could so, nobody is saying that you stop here with these iterations, because even if you start off with a 4 to 1, that could be a bad solution because the real sharing degree might only be 2 to 1 right. So, 4 to 1 would give you a correct result, but that is not necessarily the best one. You could nitrate further like you said you said, you could extend the iterative process to find the best mux as the mux best mux width or maybe different at the inputs of the different function unit.

So, as you can see if you have a large number of units, each unit actually has two inputs are and so on there is a very large number of combinations even here, what is the ideal mux width at the input of each of these this is a sort of a tricky problem. That what at least that introduces is a large number of iterations in the sequential scheduling and function unit bindings.

However this too is not exact in this assumption that that kind of sharing that I had here for those two adders, leads to this sort of mux what is the assumption we have made.

(Refer Slide Time: 12:30)



MUX Inputs before FUs

- If binding follows scheduling...
  - an FU may perform different operations in different control steps
  - MUX implied at FU input
  - MUX delay needs to be accounted for
    - by scheduler, but it's too early!
    - scheduler cannot know MUX size

(C) P. R. Panda, IIT Delhi, 2017

91

a
b
e
f
ae
bf
mux
Delay
NPTEL

If the FU is shared in two different clock cycles, what would be the extent of the mux width is the question.

Student: (Refer Time: 12:54) just think that we are just sensitive this a particular flow graph.

Yeah this is fun even for the children.

Student: (Refer Time: 13:02) two as you assumption like as you mux would not be used for any other inputs.

Even for this graph, let us just it ourselves to just that orange adder just one adder we said that this is the implied design is this always the implication or in other words is a 2 to 1 mux always inferred from such a schedule and binding decision could I have a three to one mux, just for these for the two way sharing of the same resource in to clock cycles that is my design.

Student: (Refer Time: 13:52).

That is not that yeah it is a seriously posed question, but that is not a trick question this is a 2 to 1 mux. Obviously, there are only two operations two is the level of sharing could you throw out the mux completely for some circumstance or even one of those muxes, but in general the question is just because a resource has been shared across n operations in the schedule, does it mean that I will have an n to 1 mux at its input. This affects the quality of our exploration because if I knew that that a smaller mux width would be ok then I should not necessarily assume that n to 1 mux.

Remember this design methodology if I say that I make I fixed the mux width, how will I fix the mux width? It could be based on the degree of a sharing right. So, what degree of sharing can be large for the same adder, but does not mean that I need if it is if 10 different operations widely separated across clock cycles. If they are sharing the same adder, will I need a 10 to one mux are not necessarily.

Student: Replace (Refer Time: 15:23).

Replace what with a resistor; can the mux be replaced with the register. Register we have not even talked about registers are involved in the process where are they involved.

Student: (Refer Time: 15:40) to store e.

For example to store all these intermediate signals you do need a register, that should lead to resolving this question of whether I really need it 2 to 1 mux or an n to 1 mux. So, the width of the mux is determined by what?

Student: (Refer Time: 16:00) cycles we will be sharing.

Across how many cycles, I have shared the function unit yes.

Student: For here first I am sharing for cycle and then the second cycle.

Right.

Student: So, in first cycle I am giving a and b.

Right.

Student: Then in second cycle I am giving b and a. So, let us say there was another level which is third cycle where I was using this mux then would have been 3 is to 1.

Yes. So, the generalization is if I have n different uses, I need an n to 1 mux.

Student: So, if the functional unit is used across n cycle.

Yeah so, that is our design it the same functional unit is use that is being shared. So, the extent of sharing in the schedule is n, but does that mean that I have an n to 1 mux. Clearly a max of an n to 1 mux is what I need, but could I do with a smaller width or what actually determines.

Student: (Refer Time: 17:05) the width looking at lifelines of this register so.

Yeah.

Student: How many resistors we need right. So, I look at if I have multiple registers which are available across this lifeline where I was sharing my access, then probably I can reduce it depending upon my register.

Depending upon there is a dependence upon the register, what is that dependence.

Student: So, if you could be have a register coming into the adder and the register value could be updated.

Yeah.

Student: On a time by time, then muxes could be.

Could be reduced. So, let us clearly identify this each of these inputs comes from where.

Student: Comes from a register.

Comes from a register. So, if I have 3 or 5 such edges that are coming in to that same operation right a some time e some other time and so on. Do I need five paths in the data path, they all come from registers. So, do I need five paths from five different registers to that mux I do not. I could do with fewer because.

Student: (Refer Time: 18:23) memory that (Refer Time: 18:24) register we can read only when.

The model here is that every one of these is going into some register, if it is crossing a clock cycle boundary then it is going into a register. But nobody is telling you that they are all distinct registers.

Student: (Refer Time: 18:40).

A and e could be assigned to the same register, if that were the case then that mux would not be needed. So, the number of paths you need into a particular function unit, particular operand of a function unit depends on not just the degree of sharing, but actually depends on the number of independent distinct paths that are there from different registers.

So, if some of these inputs are actually mapped to the same register, then you could have a few have a smaller width at the mux.

Student: (Refer Time: 19:21) clock cycle.

Right at our assumption of register allocation of course, is that in different clock cycles that the same physical register may store different data. So, this is actually an example of that, if I have one register that register could store a in that cycle and in fact, in the other cycle here at the end of the first cycle it could store e. So, actually I do not need that mux if a and we e were to map to the same register say in physical register.

Then in the data path you need only one path from that register, output of that register back into this adder, you do not need multiple registers. That of course, complicates the problem further, because register allocation is the third step right. So, that is just pointing out that if you want to be really aggressive about determining what the width should be, even the function unit binding step is not good enough. In fact, when you have a large number of operations like 10 or 20, it is very likely that the same register could possibly be sharing multiple of those 10 different operands and in fact, the number of paths might be much lesser than the degree of sharing of the function units yes.

Student: (Refer Time: 20:46). So, when we were solved using this graph as an.

Yeah.

Student: (Refer Time: 20:50) to solve that (Refer Time: 20:52).

Yes.

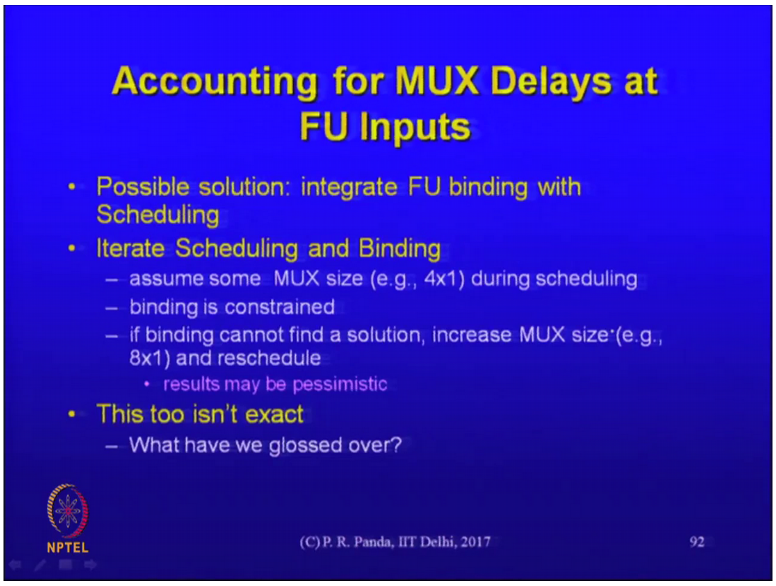
Student: (Refer Time: 20:53) at that point of time you were looking at the life line of (Refer Time: 20:54).

Yes.

Student: (Refer Time: 20:59). So, this is what I need to do.

Student: So, I need to see that. So, far is (Refer Time: 21:05).

(Refer Slide Time: 21:10)



Accounting for MUX Delays at FU Inputs

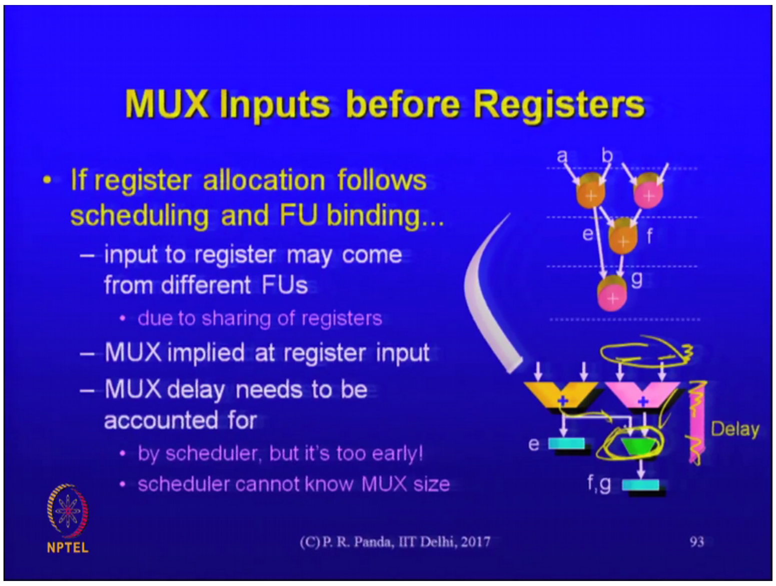
- Possible solution: integrate FU binding with Scheduling
- Iterate Scheduling and Binding
  - assume some MUX size (e.g., 4x1) during scheduling
  - binding is constrained
  - if binding cannot find a solution, increase MUX size (e.g., 8x1) and reschedule
    - results may be pessimistic
- This too isn't exact
  - What have we glossed over?

(C) P. R. Panda, IIT Delhi, 2017 92

Well what I need to do is solve all the three problems together. Now we already said that scheduling and fu binding have to be done together, now we are saying register allocation also has to be done together for me to really solve this problem and get my mux bits. So, anyway that is just to point out, but that is reality the number of paths that is there is not merely the extent to which I am sharing their FUs, it is the number of independent paths that show up all these inputs are coming from registers in the general case.

So, it does depend further on the kind of register sharing. So, from this you can say that you can see that there is an impact, I could actually use this to influence my register allocation itself, I should share the registers in a way that the number of these paths is minimized right. In addition to minimizing the registers that could show up as a further parameter, that influences the register allocation.

(Refer Slide Time: 22:16)


MUX Inputs before Registers

- If register allocation follows scheduling and FU binding...
  - input to register may come from different FUs
    - due to sharing of registers
  - MUX implied at register input
  - MUX delay needs to be accounted for
    - by scheduler, but it's too early!
    - scheduler cannot know MUX size

a
b
e
f
g
Delay
e
f,g
NPTEL

(C) P. R. Panda, IIT Delhi, 2017

93

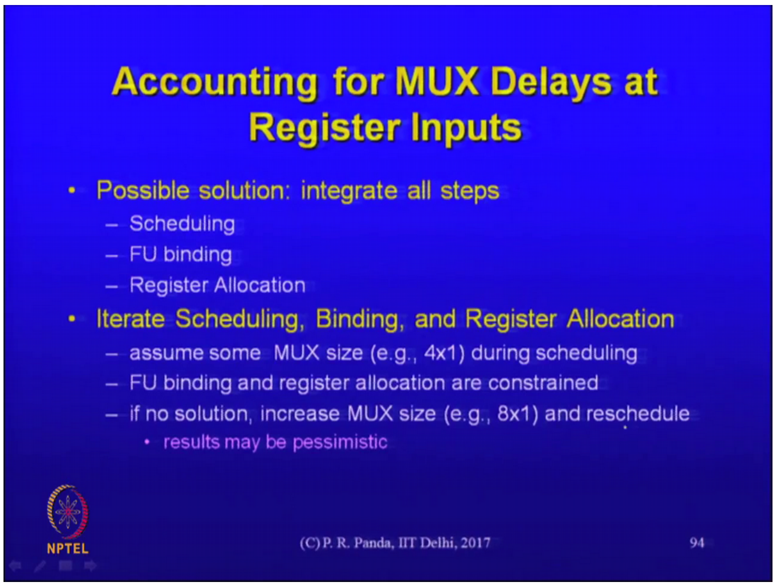
Let us look at the register side. Now, if register allocation follows scheduling and FU binding, we already actually argued that it need not and you are already giving up some optimality some efficiency at a timing level by following this sequence, but nevertheless I have sharing of registers and the input to a register may come from different function units what does that? That implies that there is a mux that may be inferred at the input of a register. If I say that f and g are both sharing the same register, then since f is coming from some function unit this one, g is coming from a different unit.

I have a mux there right for in our design here for e I did not need a mux, because there was only one value stored in that other register anyway, but for this register I needed a mux. Now you can see that that mux also influences the critical path of the design. So, the two kinds of muxes; one is the mux that is inferred here at the inputs of the function units, but the other is this mux that is due to the register sharing. So, do you see that this also influences our scheduling decision, because now we are saying that I should have enough time for that mux that operation and this mux.

All of them could be in sequence and the combinational path that is formed through that sequence, all of that has to fit into the clock cycle. Again it is too early when do we solve this clearly it is the scheduler that is taking all this timing decisions and deciding that they should go here that should this operation should go here the other operation should go there and in fact, the problem is not as simple as that. There are uncertainties here at the inputs of the FU; there is a different level of uncertainty at the input of the register that it also needs to be aware of ok.

So, in advance the schedule you cannot know the mux size what can I do about this.

(Refer Slide Time: 24:30)



Accounting for MUX Delays at Register Inputs

- Possible solution: integrate all steps
  - Scheduling
  - FU binding
  - Register Allocation
- Iterate Scheduling, Binding, and Register Allocation
  - assume some MUX size (e.g., 4x1) during scheduling
  - FU binding and register allocation are constrained
  - if no solution, increase MUX size (e.g., 8x1) and reschedule
    - results may be pessimistic

NPTEL (C) P. R. Panda, IIT Delhi, 2017 94

I can integrate now everything, if I really want an optimal solution. Actually nobody is preventing us from that there are integer linear programming based formalisms for solving the high level synthesis problem, where in theory you can combine all of this into one problem to be solved. Again ILP cannot be solved optimally, but at least the formulation is there all the parameters that affect the problem they are all there. We did not talk about it our approach to these problems was a heuristic approach; we took these priority functions and a simple approximations, just in the interest of practicality.

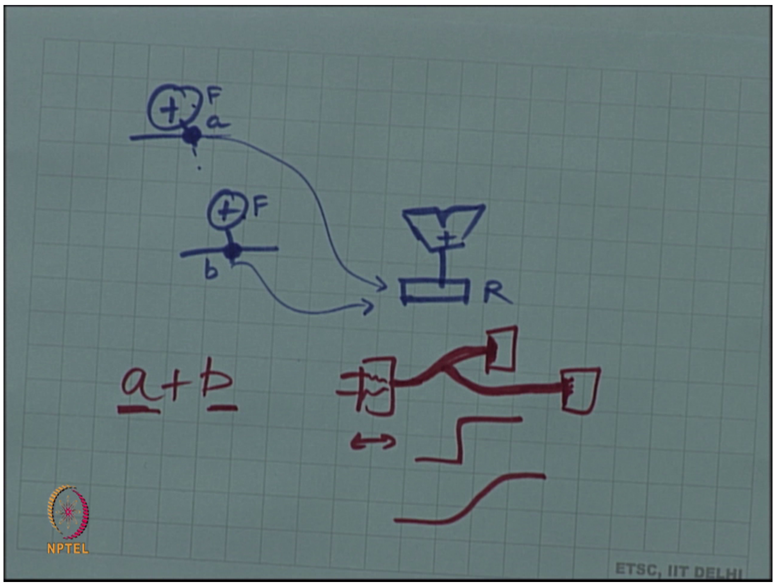
But it is not as though you cannot integrate, you can it is just that you, you can first integrate and then.

Student: (Refer Time: 25:30).

Adopt some heuristics to solve this. I could iterate otherwise scheduling binding and register allocation all in a sequence, but with the same extension that we looked at earlier like, you assume that there is a mux size fixed mux size during the scheduling maybe its 4 to 1 a FU binding register allocation both of these are constrained. So, register allocation is constrained means that there can be no more than 4 paths to that register. It is a little tricky it is not as though there are only 4 edges in our data flow graph that should go into the register.

Because the physical path comes about from individual FUs right, it could be that I have the same function unit, that is producing different results in different parts of the data flow graph and those are going into the same register, means that you need only one path you do not, that does not lead to a mux.

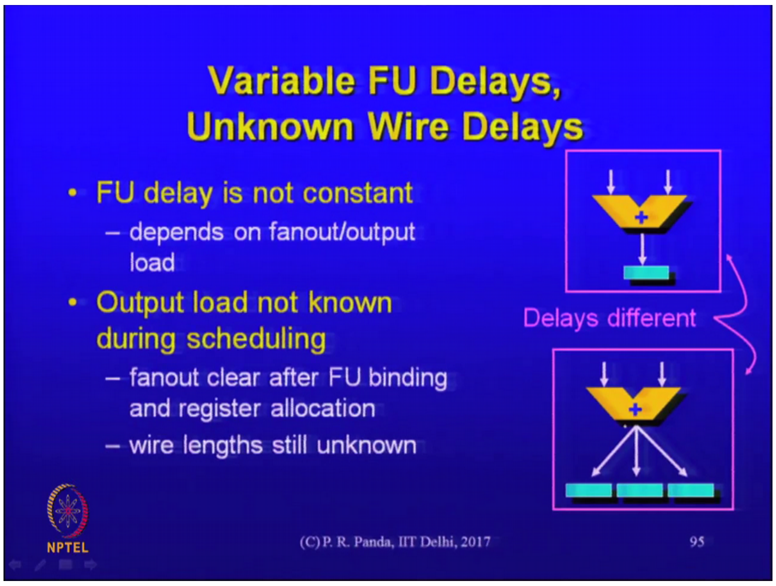
(Refer Slide Time: 26:27)



So, if I have one unit here, some other unit here and you decided that this and this both of them are mapped to the same register R. Question is does this lead to a mux; for the same reasons as earlier it does not necessarily lead to a mux, because both of these could be mapped to the same function unit if they were then I have this kind of a.

Of a design these are different parts of the schedule I have a and b, but because these two adders are shared in fact, there is only one physical path from the function unit into the register right. So, there also we should be careful not to overestimate the number of paths and therefore, the width of the mux that goes into the register.

(Refer Slide Time: 27:32)


Variable FU Delays, Unknown Wire Delays

- FU delay is not constant
  - depends on fanout/output load
- Output load not known during scheduling
  - fanout clear after FU binding and register allocation
  - wire lengths still unknown

Delays different
+
+
NPTEL
(C) P. R. Panda, IIT Delhi, 2017
95

So that is one kind of uncertainty of course, there is a classical uncertainty that we have pointed out different points of the course. In the interest of saving computation we have actually ignored all the wire delays, actually there beyond this kind of approximation regarding muxes, there are a couple of other things that happen. One is the function in a delays what determines the delay of a function unit?

Student: Combination (Refer Time: 28:09).

Yeah. So, it is implemented in terms of combinational logic, the critical path through that although we have not formally defined it, but it should be clear that the longest path from any input to any output is the critical path of the design that determines the delay. If I have an operation of that nature, how do I know what is the delay. Is it is the delay clear or there are uncertainties there with respect to how much the delay is. One thing is the delay of an adder is not specified and is not fixed, it depends on the width of the adder right. Of course, the 10 bit adder takes longer amount of time than a five bit adder. So, I actually should look at the width of a and b, and instantiate that adder that is necessary there is no need perhaps to instantiate a wider adder.

In the general case that adder may be shared across different function units, and actually I we glossed over this during the sharing formalism, but we do have to somehow influence the sharing in a way that we encourage the algorithm to share, if the two adders are approximately of the same width. If the operand widths are very different, then it is not worth sharing right if for the narrower operand you are wasting this combinational circuit. So, that is there the delay certainly is a function of the width, how do you know the width you have to actually look at the now the type definitions of a and b, and you can determine the width, but of course, you have shared that add operation across some different operations.

So, you somehow have to account for the max width of all of those operations that have been mapped onto this adder that is one thing. Other thing is it is a combinational circuit; combinational circuits delay depends on what factors? One is intrinsic to that combinational circuit there is a delay for example, you just take a and gate, what does its delay depend on you. So, it does a one standard cell that you have instantiated, it has been characterized for the longest delay from its from any of its inputs to any of its outputs, and we say that that is the delay the max of those paths internal paths that are there that constitutes the delay of the gate, but that is only part of the story delay depends on other factors right what are the factors.

Student: (Refer Time: 30:54).

Number of inputs is known or not known.

Student: (Refer Time: 30:55).

Yeah if I tell you it is a two input and gate, then you certainly know this and in fact, it is a one standard cell that has been instantiated. Of course, the standard cell designer has characterized that and has given to us as it has populated the library and it has told us that this is the delay. So, presumably it is the max delay of any of these paths, in general those paths could have different delays it depends on the circuit of course, but that has been characterized. So, it depends on one more thing.

Student: (Refer Time: 31:21).

Yeah.

Student: (Refer Time: 31:22).

It is instantiated as part of a bigger design right what does it depend on?

Student: (Refer Time: 31:28).

Ports are available or not.

Student: (Refer Time: 31:35).

What do you mean ports are available?

Student: (Refer Time: 31:37) operation (Refer Time: 31:38).

Yeah.

Student: So, if the inputs are not available like it comes from the other path (Refer Time: 31:47).

Yeah this is one of the factors, the rate at which the inputs are rising and falling do contribute to the delay propagation delay of the of the gate itself. It does right when you when you characterize this, this is in terms of you are making some assumption about how sharply the input is rising. If the input instead of rising in this way is actually rising in this way then of course, the delay is longer right that is one thing, but on the output side it is a function of the load what you are connecting the and gate to of course, determines if the load is greater than the then the delay is greater.

So, those are external factors there is an intrinsic factor regarding what is happening inside your circuit, but the delay most definitely is a function of the fan out of that design; that too is an approximation that we have made so, far we did not care about what it is connected to in computing the delay. This is a little messy as you can see.

Student: (Refer Time: 32:54).

Maximum what is the max load when the library has been characterized, when the adder has been characterized independently, there was an assumption that was made about the maximum fan out right. But did we take care of it in the scheduling resource allocation binding nothing right we did not take care of it. So in fact, that delay is a complicated equation. It is a function of the width, but it is also a function of what it is connected to some of which is clearer only in later stages in the interest of simplifying the steps we have made assumptions along the way that we somehow have to remove.

Generally that may lead to an iterative solution which may mean that you make certain assumptions, and it may turn out at the end that does not work. When do you know this anyway at later stages you do know, but anyway one of these one of the approximations is that the fan out was ignored that is one. And of course, the wire lengths are still an unknown even at the end, the fan out is still a logical entity you can count how many gates it is connected to and so on.

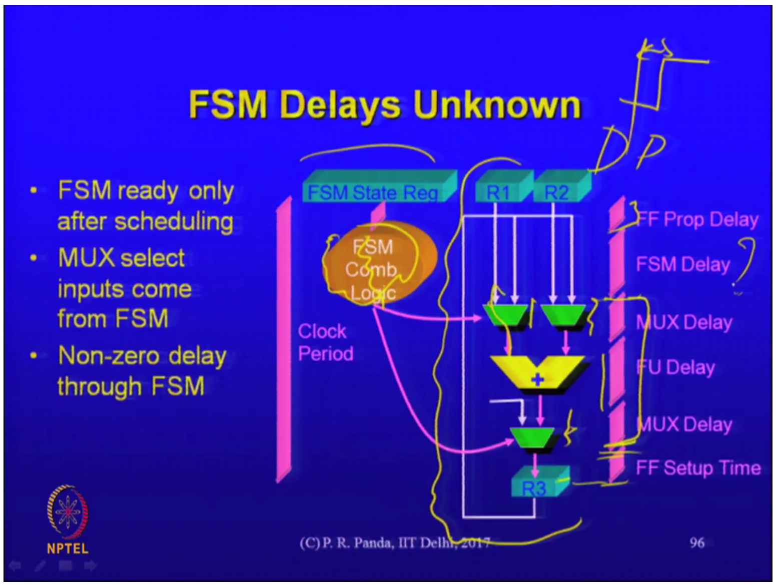
And determine and based on that you can provide some information, how do you provide that information you know what it is connected to; at some stage you do know what it is connected right its connected to this, its also connected to this and for these individual cells or whatever it is connected to after you know these connections, you do know the load that this offers to anything that connects to it right. And you can add up all of those capacitance values and that will tell you the load.

So, that is at least a better picture a more refined picture of the load than earlier, but reality of course, is that all of this also adds to the load in some way, that is a function of the geometry, which is realized much later. Which introduces a very fine dependence between all of this high level, and logic synthesis levels of abstraction, and physical design which takes place later. So, it is the same circular problem that we are talking about, which is ideally it would have been great if we knew what those delays were, but in fact, the way the design methodology works we do not know those delays. Now you can say that placement and routing should also be part of the scheduling resource allocation and register allocation and all of these steps.

So, in the general case you can see why this methodology is a little messy, there are these weaknesses that are there, which leads in the practical case to an iteration in the design flow. That you make some assumptions, it is not that you throw out all of this just because there are these deficiencies; at various levels you make different assumptions. One of the primary assumptions that is made is that out of the clock period, you exclude some fraction of that clock period as not being available to any of these high level steps placement and routing often does that. So, you do your timing analysis on the netlist the gate level netlist that is still very approximate right.

Because you do not know the geometry of the wires, but the solution around that could be that you leave 25 percent of the clock cycle for delays to be populated later on when we know when we have further knowledge of the geometry. That is all right it is not the best solution, but at least it is some solution, that helps avoid an infinite loop in the process right.

(Refer Slide Time: 36:59)



FSM Delays Unknown

- FSM ready only after scheduling
- MUX select inputs come from FSM
- Non-zero delay through FSM

There is one final delay related issue that we should be knowing, because we have an idea of the kind of output that high level synthesis is ultimately generating right. There is this data path on the data path side, there is this you have registers, you have some muxes you have function units you have ok another bunch of registers.

We said that there are these muxes that are inferred at later stages after the scheduling, but this is what the data path looks like and this is what the FSM looks like. And I put them together in this way that we already saw and we defined the problem itself. So, let us look at it from a timing point of view of course, the correct sequential design should finish all its combinational work within a clock period.

So, that is my clock period let us see what all are the different delay elements that are introduced in this process, and which all could possibly form the critical path of the design. There is an FSM here itself there is some logic, it you read the state register depending on which state you are in you would enable or disable certain function units you where you send appropriate select signals to the muxes. So, of course, that does not happen in zero time, you have to spend some time here to actually generate those signals to the select inputs of the muxes.

So, there is some combinational path delay here, we will call that the FSM delay. Actually even before that there is a flip flop propagation delay. So, at the rising edge of the clock, it takes a while for the whatever is there at the d input of the flip flop to be from get propagated to q right, then only you can do anything else. The q value is visible to the combinational logic, after a certain propagation delay that is also not zero right it of course, there is some non zero delay there. So, we start off with that, then there is an FSM delay, we have to go through that FSM does control what is going to happen in this cycle.

So, I have to do that first before any computation can take place. In fact, to decide which is the path that actually leads to the computation, I have to first do my FSM. So, that does come first, then there is this mux at the input of the effuse that we already saw, then there is the FU delay. Later on I have the mux at the input of the register, that goes into the register, but as you know the flip flops here are associated with what is called a set up time right. Before the rising edge of the clock I should have all my data ready. So, that the appropriate set up time that the flip flop needs is satisfied otherwise you will not latch the input correctly.

So, these are the delay elements that I must be aware of in the general case. Some of these are fixed; you can say that the flip flop has been characterized already for its propagation delay and for its set up time. So, that much is fixed. So, if it is fixed then it is easy we just subtract that much time from the clock period do not make it available to the scheduler ultimately the scheduler needs to know whether I have enough delay available in the clock period or not, but this is not hard to fix, but whatever the clock period for the design is, you subtract the appropriate amounts. So, that is what we are left with. Mux delay FU delay. So, this part is what we already looked at right it is complicated, but some handle we have around that what about this FSM delay has to be accounted for and subtracted appropriately from the time that is available for the data path operations how do I get that.

Student: So, FSM is synthesized at the later stage only (Refer Time: 41:09).

When does the FSM come in?

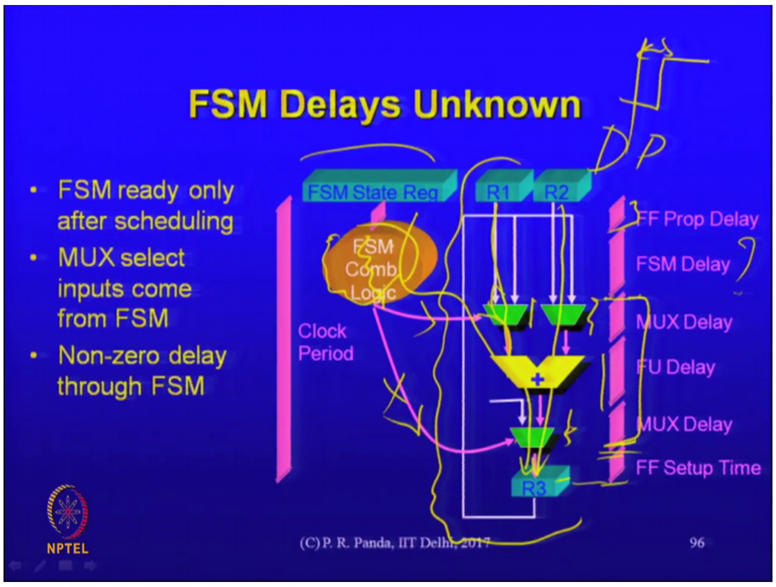
Student: After scheduling.

After scheduling you know the FSM, there is not all the end not the entire FSM this the structure of the FSM you know in terms of the states there, but there is something that is not known at the end of scheduling what is it?

Student: (Refer Time: 41:31).

The control signals are not known right do you have a schedule.

(Refer Slide Time: 41:53)



FSM Delays Unknown

- FSM ready only after scheduling
- MUX select inputs come from FSM
- Non-zero delay through FSM

(C) P. R. Panda, IIT Delhi, 2017

96

That does result in whatever the paths are it structurally in a 4 state FSM that looks like this. But the states are not everything right in the FSM, I have states and transitions. In fact, that turn this is a linear FSM we already know what the transitions are and in fact, if there are conditionals then we also know what those conditional are. So, that part is clear what about the output part of the FSM do I know this at the end of schedule I do not know why do not I know what is unknown.

Student: (Refer Time: 42:17) binding (Refer Time: 42:18).

In fact, whether the mux are not known it is unknown at this time you do not know whether to select whether to send a select signal to the mux because you do not even know that the mux is going to be inferred mux gets inferred later on in the form of fu binding register allocation and so on, until that time you actually do not know what signal to send from the FSM right. So, you have a partial FSM at the end of the scheduling but in fact, that is not all the complete FSM is inferred at the end of all of our steps the sharing steps are done and therefore, the muxes are inferred we know what are the muxes.

Now that we know the muxes, these muxes need to have the appropriate selector signals and we can construct the complete FSM at a later stage. But in fact, we want to provide this as an input to the scheduler also. This too has to somehow go through a similar design methodology. So, you could do something smarter with respect to estimating the delay, its not as though you know nothing you do know something and based on what is known here and resource constraints and so on.

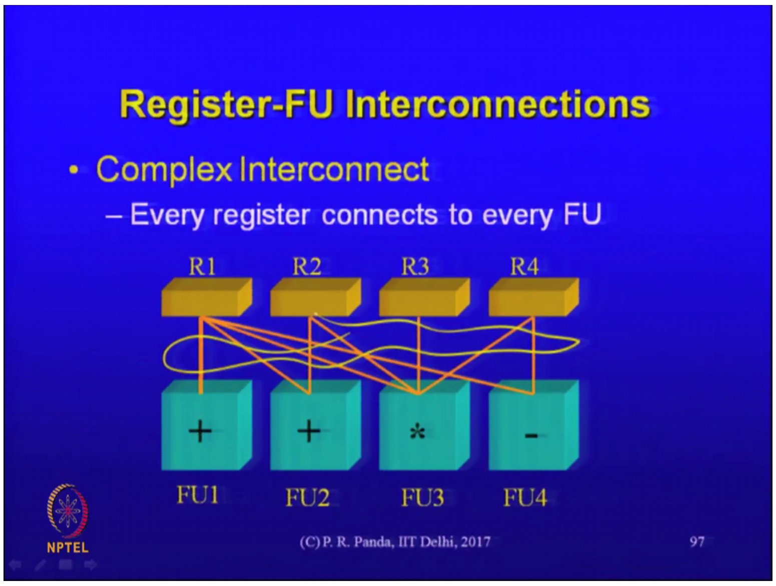
You could build a model actually we have done some work in this area, where from a high level inputs that are available to these synthesis tool even at the scheduler level, how do you actually estimate an account for the for the FSM delay. So, that you can appropriately adjust how much time is really available to the scheduler, but it is a very interesting problem to solve. You would like to know all of those delays and account for them as early as possible, but the methodology is such that you cannot possibly have all that information.

So, hopefully that was clear, what all are the delay elements that affect our clock period. There are other paths what I have highlighted here is one path possibly the more important path, but what other paths are there that we should be worried about. Notice that this signal control signal is going here, but it is also going here. This one may be a little safe because this is the one that appears early right this one in fact; this signal may be ready as soon as that signal is also ready.

So, does not affect the critical path that much. So, that one might not affect too much, but this one might, but that is not the only path as you can see this is one path that is the path that we talked.

But this is also independently a different path right whether the longer path is through the select line or from here directly through the inputs, that is not clear you have to look at the circuit to see which one is the longer path. If it happens that the path through the function unit is actually longer, then it is alright then you could actually ignore the FSM completely, but in the general case you do need to worry about this ok.

(Refer Slide Time: 45:32)



Register-FU Interconnections

- Complex Interconnect
  - Every register connects to every FU

NPTEL

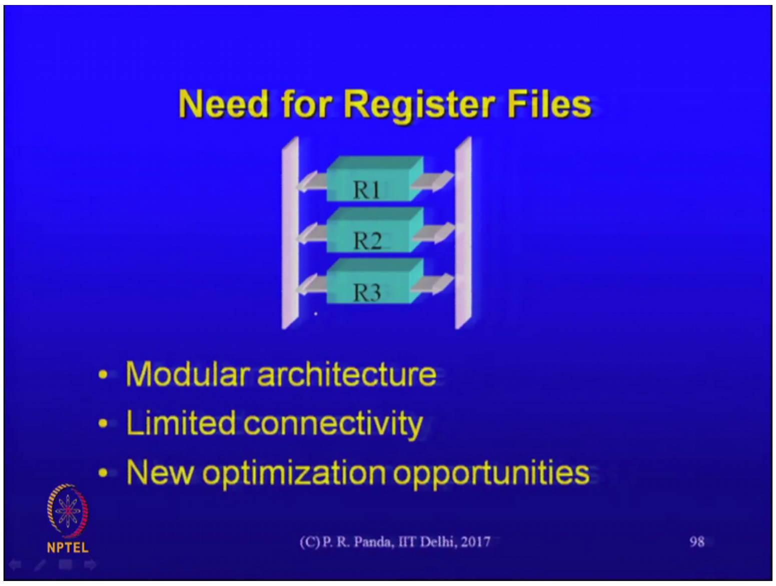
(C) P. R. Panda, IIT Delhi, 2017

97

Let us move on to a few other architectural extensions that we might have as a consequence of some of the other issues that show up. One issue is the impact on the later physical design steps. So, the way we have formulated these problems, these are all discrete registers that get inferred R1, R2, R3, R 4 you may have a large number registers large number of function units, and in general there could be a very messy interconnect structure like that, which again we said that we need to do something about this wind awareness of the delays due to those interconnect structure is important for us.

But it is hopeless to try and anticipate all the delays at the level that makes a difference which is during the scheduling. So, we can think of different ways around this, those ways include restricting the architectural flexibility in some way. We did talk about this when we talked about interconnect for example, you have introduced some buses or and so on to make it a little easier to predict the effect of these things on physical design easier to predict.

(Refer Slide Time: 46:48)



Need for Register Files

- Modular architecture
- Limited connectivity
- New optimization opportunities

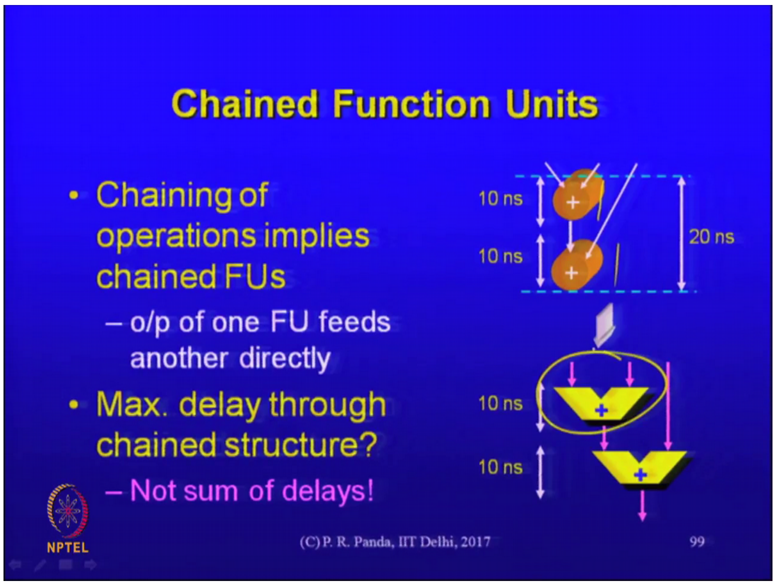
(C) P. R. Panda, IIT Delhi, 2017

98

Some of those may influence the way we organize our data path elements themselves, one good example is you could actually put together many of these registers into one unit called a register file. If you actually need access to all of them at every clock cycle then its a different matter, but there will be large number of registers well their usage maybe not all simultaneous for the most part. Then you could actually group them into a register file kind of an architecture, this is more modular and could be generated automatically in some way through a memory compiler kind of tool. There is a trade off here this does help predictability it does help the physical design part it helps layout and so on.

But what you are giving up in the process is of course, the extent of connectivity, there is a limited connectivity. So, its limited by the number of ports that are there in the register you could put a large number of registers there, but you might not be able to afford as many ports the ports might be limited. If ports are limited then it does go back and influence some of our other steps, because all the data cannot be simultaneously accessed, but the other way to look at it is that such an architectural feature would actually be incorporated into some of our high level synthesis steps right these are optimization opportunities, that when you have a large number of data path elements that need some kind of organization.

(Refer Slide Time: 48:23)



Chained Function Units

- Chaining of operations implies chained FUs
  - o/p of one FU feeds another directly
- Max. delay through chained structure?
  - Not sum of delays!

10 ns
10 ns
20 ns
10 ns
10 ns
NPTEL
(C) P. R. Panda, IIT Delhi, 2017
99

So, that is one let us conclude this discussion with another very interesting topic something that influences the execution of high level synthesis in a very fundamental way. I have the flexibility to chain operations into the same clock cycle right. So, this is an adder, this is a maybe it is a copy its of the same adder, but we are introducing both of them into the same clock cycle just because there is enough time. Now how do you implement this we already saw that the way to implement this is just have the output of one adder feed as the input to the next adder. So, that is clear, but question I want to pose this what is the max delay through this structure. And there is a big hint there that tells you what the expected answer is not.

Student: (Refer Time: 49:23).

There are these other effects that we looked at set up time there is of course, the assumption here that wire delays are 0 and so on, but this is an orthogonal problem all of those are there independent of that just from a logical point of view, from the combinational logic critical path point of view the question can be separately answered what is the maximum delay you expect through the structure.

To help you interpret the question and to indicate that there is no trick curse or something here. What I mean of course, is that 20 nanoseconds of course, are enough to realize this, but could it be less than that even though the path that is defined through each of these individual structures is 10 nanoseconds .

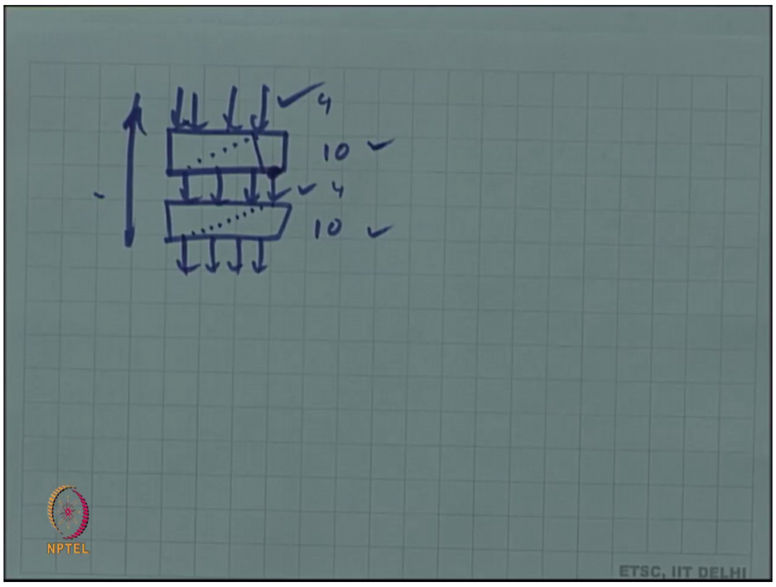
Student: (Refer Time: 50:27) if I know that second input is 0 or something.

If you know.

Student: Second adder input is redundant, but that will not be case that is something which (Refer Time: 50:43).

No you do not know that we are talking about this is a circuit question that is being asked there is nothing funny that is going on here. You can see what is the general question this is a specific instance, but the general question should not be hard to answer. You are stuck at the twenty nanosecond here because we have used this example all along and you are wondering where did we go wrong yeah it did not go wrong anyway, but let us say I have some piece of combinational.

(Refer Slide Time: 51:11)



Logic this is combinational logic right and there is a combinational and I have some other piece of combinational logic right.

There are inputs here, that are passed on to the cascaded input for simplicity here we have assumed that it is another copy of the same design, in general these are or are the same or are not the same its all right our argument can be the same. And there are a bunch of outputs let me put 4 in that is my design. So, if I know that the delay through this is 10, if I know that the delay through this is 10 what is the delay through this whole structure is it necessarily 10 plus 10.

Student: Upper bound is.

Upper bound is 20great, but as I said I am not challenging your arithmetic abilities, this is computer arithmetic abilities that we had talking about.

Student: So, there is no dead logic right.

There is no dead logic if it was dead I would throw it out of it this is a library element somebody, a respectable designer has designed it and of course, he has not filled it with blank space or some such thing. So, I am taking an off the shelf component which is an adder, and I am cascading these two. But in general this is a function and two functions I have cascaded I am asking the delay through the combined cascaded structure.

Student: So, this 10 nanosecond is the maximum delay, which is given by the person who is characterize it.

(Refer Time: 53:09). So, there is no reason to disbelieve him once he said that.

Student: (Refer Time: 53:17).

Yes.

Student: So, by definition.

It is the worst.

Student: (Refer Time: 53:25) critical path, and then we can complete in less than that (Refer Time: 53:32).

So, tell me what should.

Student: I mean this is the 10 nanosecond is the worst case (Refer Time: 53:40) less than 10 nanoseconds and then second term combinational logic will get the output.

So, are you saying that for some operands for some data values you will get the result in less than 20 nanoseconds, but that is all right, but this bigger unit now needs to be characterized by this is. Now your design you have to characterize it when you sell this to a customer, and the characterization cannot be that for some particular combinations of data that I am not going to tell you, the delay is much longer that is not acceptable.

Indeed it is worst case that is an is an understood thing, but question is if 10 is the worst case for this guy what is the worst case for this? That is not to say that this argument has no merit, if you could show that the worst case that could come here cannot also occur simultaneously here because of the nature of the logic, then you could argue that my worst case for this combined structure, theoretically never leads to both the delays being 10 nanoseconds.

If you could argue that then it is its fine independently that is the different analysis, but my question here was more from a structural point of view really. But the arguments are similar, this has to do with this general observation that if you have two cascaded elements we have worst case of one verse worst case of the other the worst case of the cascaded unit together is not the sum of the worst cases, it is always almost always pessimistic. But you can see that in this example it is in this simple example not yet in that adder example because there we did not go into the detail, but this example you can see give a general worst case path for this function unit.

Student: (Refer Time: 55:48) I had cascaded same thing (Refer Time: 55:52). So, for the first output (Refer Time: 55:53), but once the outputs are available ready it is more over pipeline effect can be (Refer Time: 56:07).

That is a different kind of an architectural element, there is no pipeline here because this is a of course, this is a combinational circuit. You are getting closer and closer to what I would ideally like to extract from you, and not want to just pause the just give up.

Student: (Refer Time: 56:27).

You are absolutely right, but I want you to complete the argument, but argue this theoretical argument can be easily completed. I said that the worst case path in this element is 10, given a generic how many paths are there in this design from any input to any output how many parts are there, which could be possible candidates for the critical part; is that also something that needs analysis? There are 4 inputs and there are 4 outputs how many parts are there all we do its a black box. So, any of those could be the critical path right. So, there are sixteen paths and just let us identify give me some path that could be in general the critical path.

Student: (Refer Time: 57:042).

Like this let us this is actually not bad from added works like this right the least significant full ladder, influences ultimately the carry out or the sum of the MSB sum and therefore, that could well be it. In this other design similarly that is the critical path is the same adder right does I tell you what that did that is you are severely overestimating the delays by adding them up, where is the critical path of the combined circuit.

Student: (Refer Time: 58:20).

Right the second order can begin adding as soon as the first.

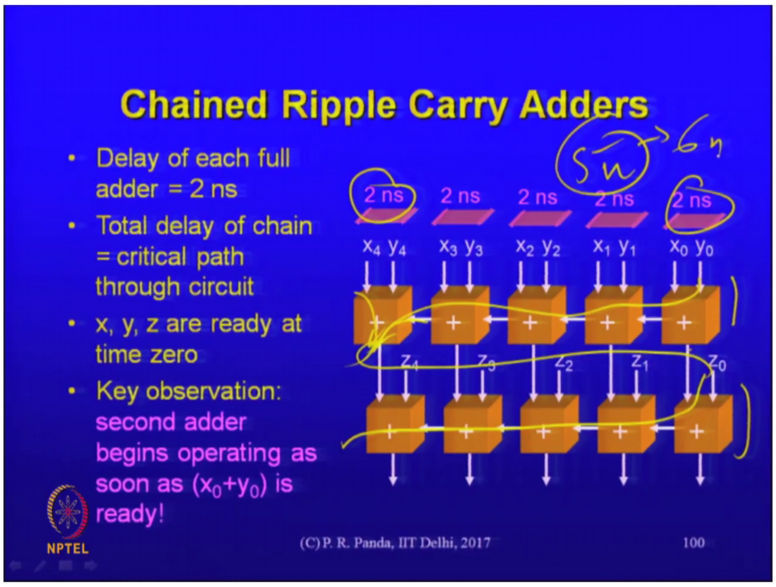
Student: (Refer Time: 58:32).

Output the some bit is available.

Student: (Refer Time: 58:36).

It is marginally higher than a single adder. It does depend on the architecture, but at that level of abstraction where we drew this adder design its it seems as though there is no argument.

(Refer Slide Time: 58:51)



Chained Ripple Carry Adders

- Delay of each full adder = 2 ns
- Total delay of chain = critical path through circuit
- x, y, z are ready at time zero
- Key observation: second adder begins operating as soon as (x_0+y_0) is ready!

(C) P. R. Panda, IIT Delhi, 2017

100

But if you look at the implementation of the ripple carry adder, this is what the circuit is likely to be. This is the first edition, this is the second adder result of one that is the some bit these are the some bits right; those bits are going as inputs to the second adder. Critical path of the first adder is this, critical path of the second adder is this, critical path through the whole circuit is not the sum of the critical paths that would have been only if there was a path from here back to there.

Therefore in fact, for a ripple carry adder kind of cascaded circuit, you can easily see what the equation is for the critical path of the cascaded design. So, if each were to there is a full adder delay that. Full adder device is any input to any output here there is an approximation, because that input might be this delay might be different from this delay and in fact, what when we talk about critical path, we are worrying about the in general the carry in to carry out delay. That is the one that needs to be optimized actually and if you look at some of these smarter adder designs, they try to make sure that the carry into carry out is what is optimized. The data to some that path you do have time for that is not on the critical path of a wide adder, but is the c into c out which is on the critical path.

Student: (Refer Time: 60:26) operation which was of 10 nanoseconds in to multiple independent operations right.

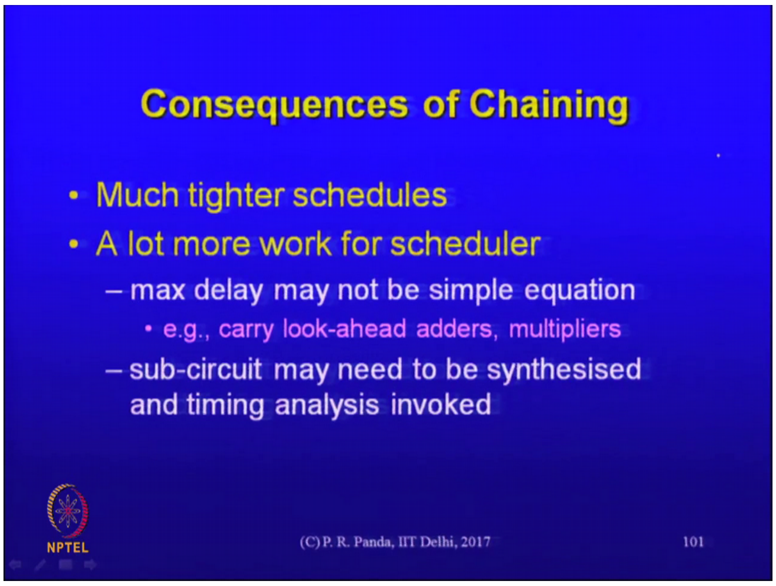
Yeah, in the analysis, as part of the analysis.

Student: (Refer Time: 60:40).

Yes yeah that is the of course, the reason it shows up as a problem is it may not be as easy. You can see in the ripple carry adder case, if this delay the delay of adder was 6 n where n is the full adder delay, then the delay of the ripple carry structure of the cascaded a structure is not 5 n it is 6 right that is all that is it. But this is simple only for this kind of a structure. So, two implications one is that this knowledge leads to much tighter schedules, that you could actually cascade a number of additions all in the same clock cycle later. So, 10 plus 10 was way.

Pessimistic you do not have to do that and of course, these principles are exploited nicely in alternative adder structures like carry save adders that people use, but. So, this is one thing that is of course, something that you would like to do, this is nothing new these ideas are well known to designers. And if the high level synthesis tool has to work well has to produce good designs, it has to anticipate things like this otherwise the circuits will not be acceptable the delays will be way too much.

(Refer Slide Time: 62:04)



Consequences of Chaining

- Much tighter schedules
- A lot more work for scheduler
  - max delay may not be simple equation
    - e.g., carry look-ahead adders, multipliers
  - sub-circuit may need to be synthesised and timing analysis invoked

NPTEL (C) P. R. Panda, IIT Delhi, 2017 101

So, if you are aware then the schedules can be much more efficient of course, but this is also a little difficult problem it leads to a lot more work in general for the scheduler. Because the units that are being cascaded, could be anything these were simple units, but it could be that well if you are carry look ahead adders instead, you need a different analysis. If well there are five elements that are being chained one is a multiplier, one is some other unit. In general case this does lead to a lot of competition how do you do that? It could be that you could build statically a table that says, if this unit is cascaded with this unit then the result is this ok.

And the number of units in the library is small. So, hopefully the table is small, but this is just the starting point. Second thing is that you have 10 different kinds of units, it is not though the chaining is only two way right you may be chaining t units or 5 units and then that table does get larger. Not just that now if I begin to say that a I have a 4 bit adder or a 12 bit adder or 30 bit adder again this is a function of time.

So, there it might not be a neat function of n, where n is the bit width because you are dealing with different kinds of operations, you are cascading different kinds of operations a closed form solution might not be there closed form expression might not be there for computing the delays. However, designers are aware of such things. So, to match the quality synthesis tools might often, synthesize a sub circuit consisting of those cascaded units as part of a high level synthesis step. How optimal it is s a different matter you cannot afford those kind of times that an rtl designer has when he submits an rtl design to a synthesis rtl synthesis tool.

But even a behavioral synthesis tool in order to do a really good aggressive design with respect to timing, might as part of as a subroutine actually invoke some rtl synthesis tool or essentially it is a timing analysis that is being performed. So, circuits are these design components are first instantiated and the timing analyst is being performed, as part of an internal synthesis procedure.

This is a very expensive as you can see and often is a step that leads to significantly increased execution times of the behavior synthesis tools. On one hand these, these principles are not new these are known to the designer. So, if you are aware if you understand design, then you expect good quality designs do you expect that the synthesis tool should recognize that and give you designs like that. But it is not so, easily integral nicely into our formalism still those chaining as an extension to the basic scheduling was introduced as a step that is not so difficult right.

Theoretically it is still not so difficult, we remember how ever we deciding upon chaining we were just saying is there enough time slack time left in the clock period considering what the dependent operations, where the predecessors where, and if there was enough time we would insert it, but this question of is there enough time is a lot trickier it is not. So, easy to find out you may in the general case have to build that circuit and perform timing analysis on that to determine is there enough time or not let us stop here this concludes the high level synthesis discussion, we will move on to a little lower level of abstraction rtl and logic synthesis and later classes.