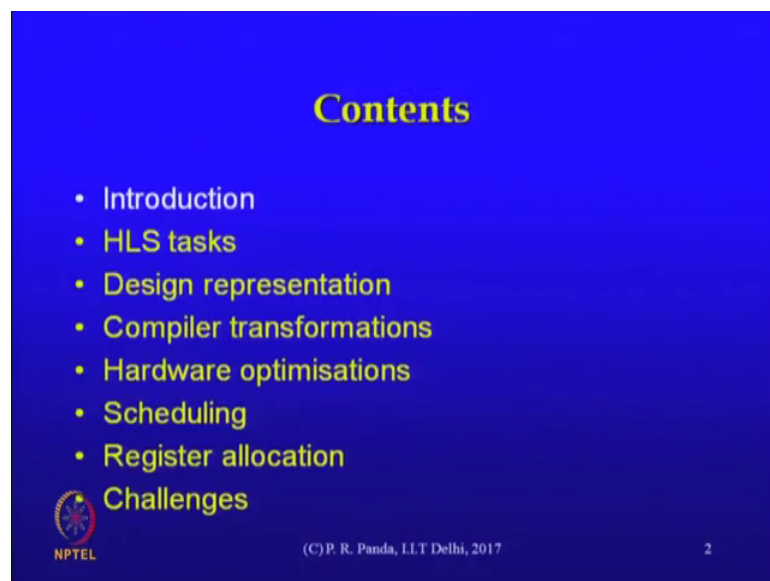


Synthesis of Digital Systems
Dr. Preethi Ranjan Panda
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 07
Introduction to High-level Synthesis

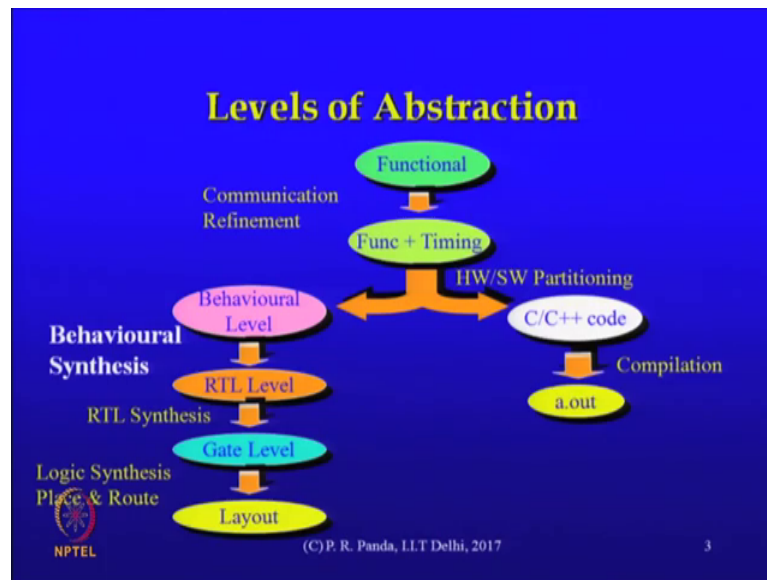
Let us go on with the next topic. We will spend a few weeks on behavioral synthesis or high level synthesis.

(Refer Slide Time: 00:27)



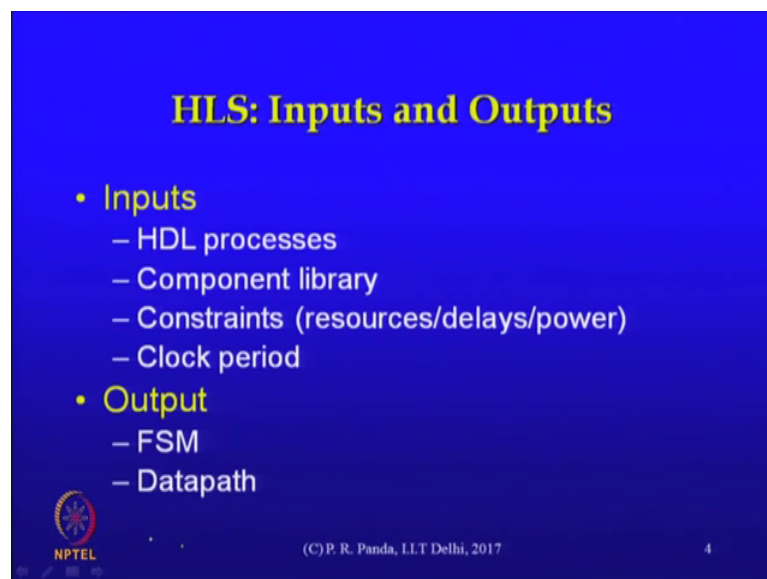
If you remember from our overall flow of the several levels of abstraction that we had of the design behavioral synthesis takes us from what we said was the behavioral level of abstraction we had defined it as one in which functionality is specified, but timing and other details may not be cleared, particularly the mapping of functionality two clock cycles are control steps has not yet happened.

(Refer Slide Time: 00:28)



So that is what the main job of behavioral synthesis is we convert from a level of abstraction where the time and clock cycle level detail is not quite clear to one in which it is explicitly clarified.

(Refer Slide Time: 01:26)



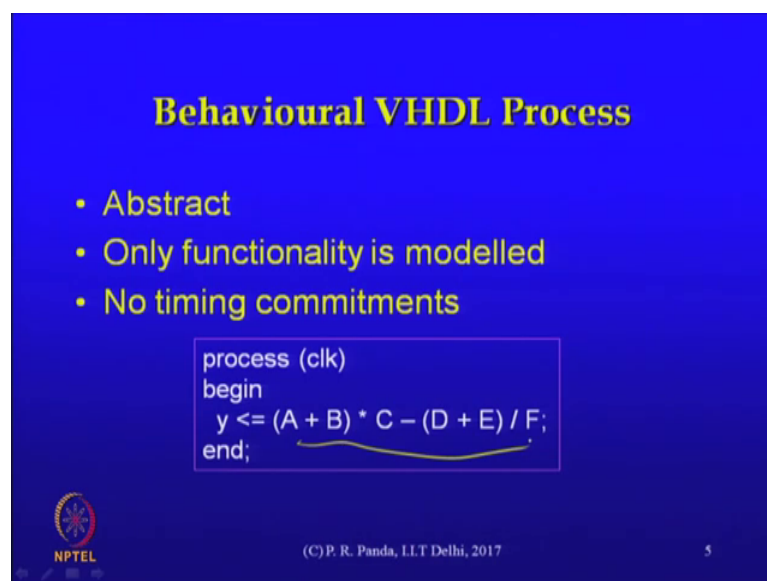
The inputs first of all it would be the set of HDL processes that we wrote. So, there is of course, an entity and architecture and a bunch of processes inside that architecture that HDL that we wrote is the main input to behavioral synthesis that is the starting point. But it has to start from there and ultimately realize the entire synthesis process has to realize

a circuit a digital circuit that implements that design that we have specified. So, how will it implement it? We have to provide a component library whose elements will constitute the circuit ultimately right. So, that component library needs to be there.

Beyond that there are several other things we may provide most important of which is a set of constraints so that will specify under what condition is this design occurring. So, these constraints come from where they may come from area budgets that you want to impose on the design it may come from timing requirements speed requirements of the design, power requirements and so on. These are externally imposed these are not inferred from the design itself like when you write a VHDL model it is not clear what the power budget is, so these constraints are externally imposed and they are also inputs to a synthesis tool.

Clock period is typically also imposed as a constraint although you could have a variation of the synthesis process in which you can ask the synthesis tool to come up with a solution and along with the circuit it also gives a suitable clock period for the design it could be. But usually the clock period is provided as an input. And what are the outputs? At this level of abstraction the output is not that final gate level netlist that we talked about remember that is a lower level of abstraction, but here there would be a data path what it looks like will get too soon and there would be a controller at least these two components are there in the output of the behavioral synthesis.

(Refer Slide Time: 03:48)



Behavioural VHDL Process

- Abstract
- Only functionality is modelled
- No timing commitments

```
process (clk)
begin
  y <= (A + B) * C - (D + E) / F;
end;
```

NPTEL (C) P. R. Panda, IIT Delhi, 2017 5

So, let us define a VHDL process that is behavioral or high level just for illustration we are modeling essentially only the functionality here. So, you can see that this is sort of a complex expression there is addition there is multiplication there is division that entire expression is not going to be completed in one clock cycle obviously. But there is still some value like we had argued earlier just like there is a value to 0 delay assignments there is a value to these kinds of processes where complex functionality is specified and we just have sensitivity to where clock.

No timing commitments means that we are not really saying when the addition should happen, when the multiplication should happen, when the division should happen and so on that should be the output of the behavioral synthesis process. But the input is just an abstract description in which you have the functionality, but not necessarily all the timing level details.

(Refer Slide Time: 04:54)

Refined Behavioural Process

- Abstract behaviour partitioned into clock cycles

```
process (clk)
begin
y <= (A + B) * C - (D + E) / F;
end;
```

→

```
process
begin
t <= A + B;
s <= D + E;
wait until rising_edge (clk);
p <= t * c;
q <= s / F;
wait until rising_edge (clk);
y <= p - q;
wait until rising_edge (clk);
end;
```

The slide includes the NPTEL logo and the text '(C) P. R. Panda, IIT Delhi, 2017' and the number '6'.

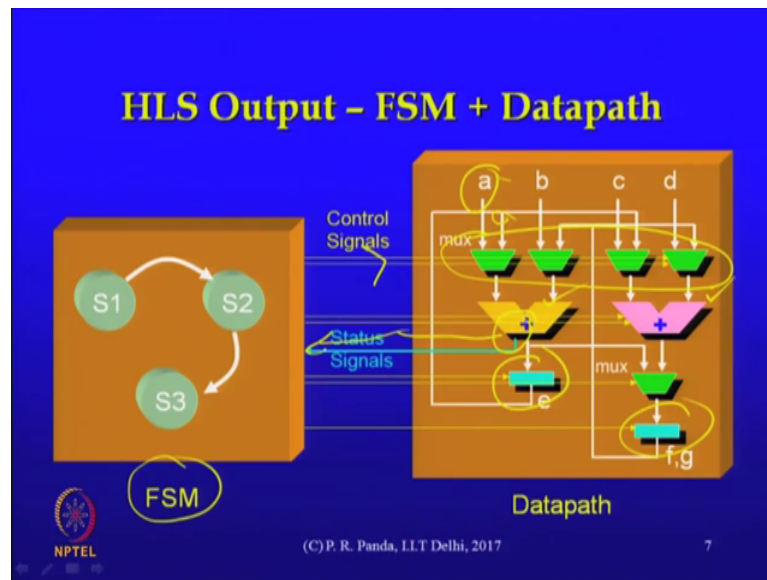
You could think of a refined behavioral process in which we actually explicitly make these decisions. So, there is an A plus B there is a D plus E right you can see that these two additions are independent of each other and could be performed simultaneously. If you had the resources then both of these could be executed in the same clock cycle that is what the refined VHDL model here looks like. This is actually the same thing functionally it is the same, but you can see that timing wise it is different what we have done is divided time into 3 clock cycles and in the first one we are doing that part of the

computation that is highlighted here. We are waiting until the rising edge of the clock the rising edge is just a simple function we know how to implement. The rising edge right this is just a function that is doing the same thing we said clock event and clock equals 1. So, such a thing would be what would be there in that function called rising edge.

Then I have the multiplication and the division. So, since this addition is complete I now, need to do these two, I now have the operands for doing that multiplication and the division right in the second clock cycle of course. This is still an abstraction you may not finish a division in one clock cycle, but this is just an illustration. And finally, after this is done the results are stored. So, these intermediate results are stored in t and s. So, we can that is my t and that is my s after doing the multiplication the results are stored in p and q and finally, I have that subtraction with the result right so that is the multiplication and that is the division. So, this is what is going on here. This is a little more detailed what you have on the right side is a little more detailed version of the same functionality in which some synthesis decisions have been taken. What are those? These are scheduling decisions these are the mapping between individual operations and the clock cycles.

So, that is at an elementary level what is going on here at least that is one of the major decisions that has taken place in the synthesis and the output of the HLS looks something like this there is an FSM and there is a data path that data path would consist of those components that we pick up from the library that of course, should be able to implement those operations that are there in the HDL.

(Refer Slide Time: 07:25)



Then there is a controller that should make the decisions of which of these operations should happen in which clock cycle the output should be at least this much it can be more. So, you do not necessarily have to have only one data path and one FSM there could be multiple data paths and FSMs and so on, but this is what the picture might look like of the output of the high level synthesis process.

So, I have a bunch of these are arithmetic circuits there are multiplexers here there are registers to store intermediate data. There are control signals that go from the controller into the data path what is their job their job would be if they are going into the muxes these are the select input of the muxes that they are going into they would tell us whether this operand should be passed into the adder or the other operand should be passed into the other. So, such control goes from the FSM into the data path there may be some status signals that go in the other way from these computations that we have we may send some signals out back into the FSM. So, there may be signal flow in both directions between the controller and the data path that is what the output is likely to look like of the HLS process this is just a high level picture let us go into some detail and.

Student: Sir, in the in the system there will be created as files FSM.

In the system will they be created as files. You can think of this as an HDL model the input was an HDL, but the output is also another HDL, but now, it is a lower level of abstraction you can think of the FSM is still VHDL. But lower level of abstraction

maybe the FSM is one entity and the data path is a different entity right or it could be the whole thing is one entity, but the FSM is one process and the data path is more structural. So, you have instantiation statements composing the data path part of it, but as of now, the FSM as you can see is still little symbolic as of now, right this is not a structure the data path part of it is structure, but the FSM part of it is still symbolic, but you can think of that as two different entities one of which is the FSM and the other is a data path.

But that was just a very high level picture. Let us go into one more level of detail in high level synthesis.

(Refer Slide Time: 10:20)

Resource Library

- **Function Units: Adders, Multipliers, Comparators, ALUs**
 - Number ✓
 - Type: differing area/delay ✓
- **Memory**
 - Number and size
 - Port number and types
- **Information**
 - Function
 - Area
 - Delay
 - Power Dissipation

2 ADD: RCA
1 ADD: CLA
1 MULT ✓
2 MEM: DP ✓

Resource Library

NPTEL (C) P. R. Panda, IIT Delhi, 2017 9

What is that resource library? We said that the HDL is an input is a primary input for the synthesis tool to start off, but the resource library is another input why is this another fundamental input because the output is going to be composed of elements that are picked from this resource library. So that library would look like this I have a bunch of these components maybe they are added RCA as it is a ripple carry adder this is a carry ripple adder indicating that the library may have multiple components for what is essentially the same operation in the HDL right. You just have a plus in the HDL, but that could be realized in terms of different architectures in the implementation it could be a ripple carry adder it could be carry (Refer Time: 11:17) adder, these different implementations have different properties therefore, the synthesis tools job it is to decide

which one to pick up right. Multiplication, a memory and so on, these could be a bunch of resource library elements that you are taking up from.

These numbers here could tell us for example, how many such resources are there sometimes we may have a constraint on what resources are there and how many of each resources we are able to use. So, it could be that we also have this information of how many resources of each type are available for us.

Our function units would be adders, multipliers, comparators, ALUs and so on I would have perhaps the number of them, but also important is the distinguishing features between those components would be visible to the user of that library the synthesis tool is the user of their library. It could be that I have a memory and that memory would be characterized in terms of what parameters how many memory elements are there that could be 1, but the size of the memory the width of the memory, how many ports are there and so on those are some of the basic features of the memory modules that needs to be exported to a synthesis tool so that the appropriate decisions could be taken.

Beyond that there would be other typical information that I would store along with my library elements that would be first of all what function is it implementing. So, if it is adder or a multiplication or some ALU and so on that has to be encoded in a way that the tool is able to understand and make the association between operations in the HDL and the appropriate components in the library.

Area of the component, delay of the component, power dissipation and so on these are typical properties of those components that would be there as part of the library. Yeah.

Student: Since we are doing a high level synthesis (Refer Time: 13:29).

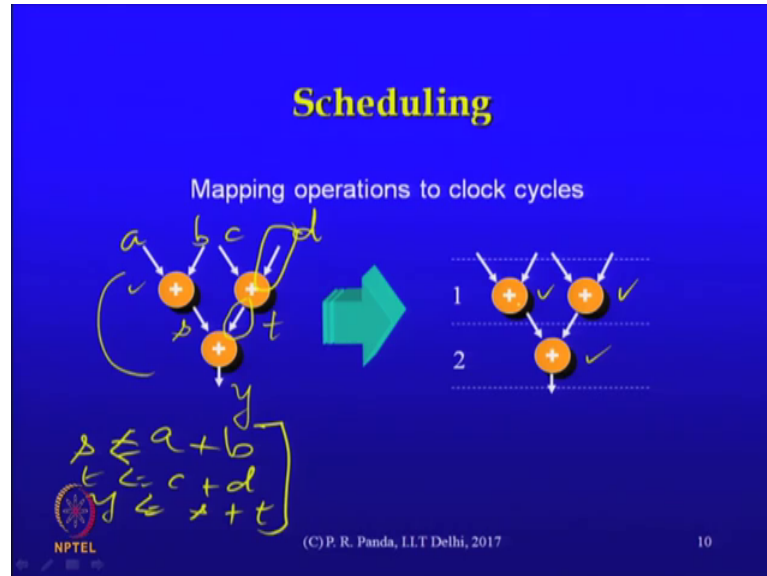
Yeah.

Student: The area delay which will be differing across multiple resources would they motional or they be not to a technology.

So, are the area delays technology is specific the resource library is typically technology specific yeah. This is still only part of the delays remember there are other parts that are not necessarily visible at this time, but components are characterized for the maximum

delay from any input to any output that is the kind of information that would be available here when you pick up elements from a library.

(Refer Slide Time: 14:10)



Let us define one very elementary operation that in some ways is the starting point of the high level synthesis that is called scheduling. You have operations we have indicated a graph over here which in some ways violates the definition of a graph remember you cannot have hanging edges every edge needs to connect two nodes, but the meaning should be obvious hopefully here what is being shown here is just a part of a larger graph. But there are some operations and that structure what does it mean the node represents some operation that is there in our HDL you can always try to map these pictures that we are seeing with HDL that you would write we do not necessarily point out the HDL all the time, but sometimes it is important to keep that in mind.

Where did they come from, where did this graph come from? It came from an HDL that we originally specified. What does the edge represent? It represents a dependency between the operations. You may have these operands as a b c and d alright and you may have an s and t and 1 and your statement in the HDL from which such a structure might have been inferred or the set of statements could be that you have s equals a plus b and t equals c plus d y equals s plus d. So, some such set of statements might have been there from which we derived that graph. But often we will skip the HDL part of it and will

directly show the graph and show the operations and the algorithms that are working on the graph. But remember they all came from an HDL that we ourselves wrote.

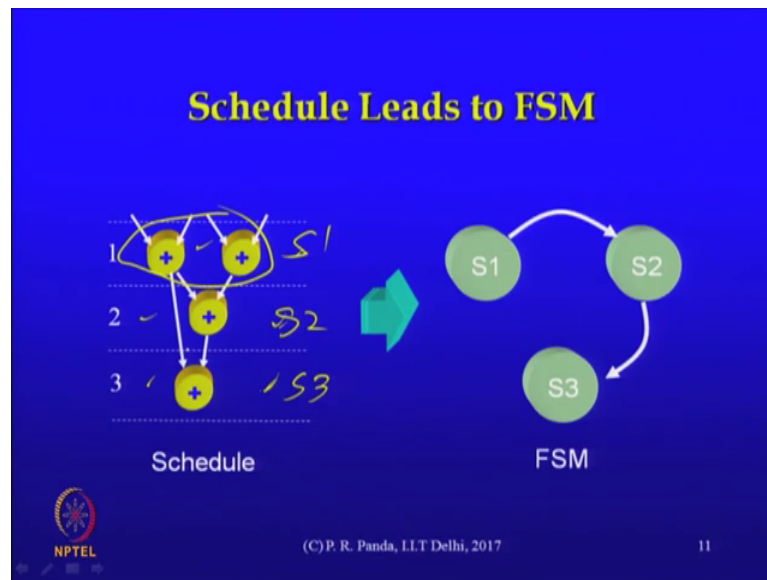
What is scheduling then? The scheduling is taking these operations and associating a clock cycle with them associating a timing with those operations, we did not specify any timing in these assignments right these were zero delay assignments that we specified, but of course, finally, these things have to be mapped into clock cycles all the operations do not necessarily fit into the same clock cycle. Whether a new clock cycle is necessary or not how many clock cycles are necessary this could be part of the scheduling tools decision making process it is an important part of the high level synthesis tasks.

So all that we had we have done here is on the right we have divided the operations into clock periods saying that these two operation these two nodes will be scheduled in two clock cycles 1 and that other node here the third node would be scheduled in two clock cycle 2. Of course, this is possible assuming some knowledge about a resource constraint. So, in order to realize this it is essential that we should have two adders; these are all adder circuits that are necessary in order to perform both of those operations in parallel. So, same clock cycle means that the operations are being performed in parallel and therefore, you need access to two adders in order to realize that circuit. So, somehow that resource constraint was available to the scheduler that is why it took that decision. But what if it had 3 adders could it have scheduled all the 3 into the same clock cycle.

Student: (Refer Time: 18:03).

If the clock period was wide enough maybe all of them could have been scheduled into the same clock cycle right the whole thing would have been just one big combinational circuit and as long as the delay fits in the clock period it is ok. But if not as is the example here where let us assume that the delay of one adder is the clock period that we are given then the other operation has to follow the first two operations. There is a dependency that is inferred from the specification itself that cannot be violated you cannot schedule the third operation in parallel with the first two here to wait for the first two operations to conclude why do we do this schedule.

(Refer Slide Time: 18:43)



Remember our objective in high level synthesis is to go closer and closer to the lower level of abstraction right. So, remember this timing was left out in the abstract description that was the input and we are trying to add more and more timing level detail into the design.

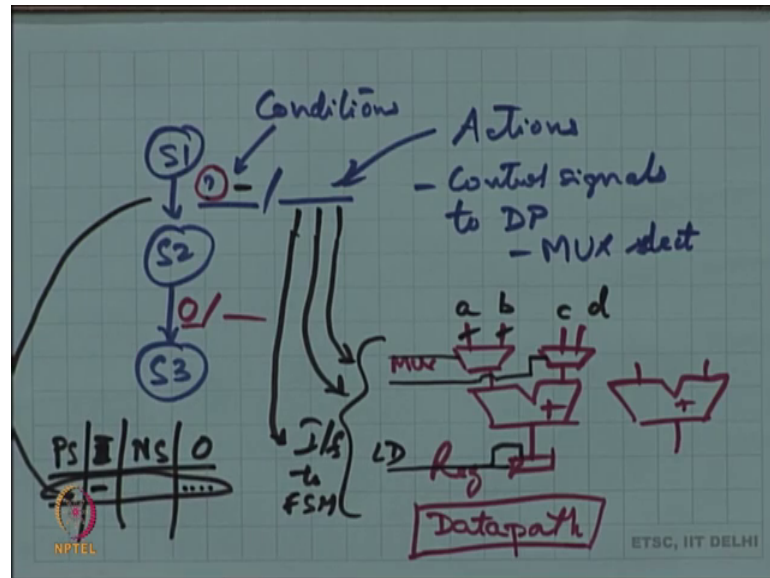
Schedule as you can see if I had that graph here and I took this decision of these two operations in the first cycle this in the second and this in the third because of various reasons all of which are not indicated here, but it is primarily dependent on the nature of the dependencies in the HDL itself. But also what resources are available delay of the individual adders and the clock period that was given to us using all of those as the input the scheduler decided on such a schedule.

So, we have this information that in the first clock cycle. So, I need only 3 clock cycles to implement my design in the first clock cycle I should enable these two operations in the second clock cycle I will enable that operation and in the third clock cycle I will enable this operation. This is essentially an important piece of information from which the FSM can be composed that what is the FSM anyway. So, you can see that in an FSM what kind of information goes. We can associate with the state or with transitions some actions right. So, there are conditions and actions associated with these transitions.

So, let us say for now, that the state s one corresponds to clock cycle 1 state S 2 corresponds to clock cycle two and s 3 corresponds to clock cycle 3. So, such a schedule

leads in an immediate way to a finite state machine which of course, we said is one of the main outputs of the synthesis process. This is not the entire FSM right, so let us draw that FSM here I have S 1 and S 2 and S 3.

(Refer Slide Time: 20:51)



That is the FSM that is inferred from this schedule in the picture. This tells us about the structure of the FSM, but some information is missing words missing.

Student: In the transition condition.

Yeah. So, there is associated with this transition there is a condition. What else is there in the FSM? You have states, you have these transitions, you have the conditions. So, I need to specify these conditions and there are actions. What would the actions be for the FSM?

Student: (Refer Time: 21:34).

So, these are the conditions I need those conditions, but there are also actions. I can associate actions with the transitions or with the states themselves depending on whether I choose a mealy style machine or a more style machine. So, there are actions here right. What actions are we talking about here in this example?

Student: (Refer Time: 22:06).

What is the output of one state?

Student: The result of the calculations (Refer Time: 22:12).

Result of the calculation. The calculations themselves are not taking place in the finite state machine so that is the go back let us go back to that that picture of an FSM and a data path. The computation part of what we have specified with the if there are addition, there are multiplications and so on those are part of the data path. The FSM is only controlling the data path it is the controller right. So, the calculations that we have given the computation part of HDL they go into the data path, how do you do the calculations here there are additions right. So, those additions that we have in this diagram they would be actually occurring in the data path part of it.

So, what is there in the controller? The controller has this key information of when the additions should happen right, but the additions themselves happen in the data path. So, given that is the model what would be there in the actions controller.

Student: (Refer Time: 23:19).

What are those control signals?

Student: Special.

Yes, we already saw that. Remember the same adder might be doing different operations in different clock cycles; I can reuse that same adder to do different things in different clock cycles. So, the inputs may come from different sources for the same adder where they come from that can be controlled by the finite state machine. It is the FSM that knows whether we are currently in cycle 1 or cycle 2 or cycle 3 and it can send the appropriate select signals to the muxes. So, these actions would essentially be control signals to the data path. So, and what controls signals these are essentially these mux select signals and similarly if there are other elements like registers then there also we may need to send some signals of when to load a register.

Fine actions should be clear what about the conditions. In this example where we have that graph, this is the graph that got synthesized into an FSM and a data path. What condition would go into the FSM?

Student: Status signal (Refer Time: 24:47).

Status signal. What status signal?

Student: Computation.

We talked about this particular example. You can visualize what hardware we are talking about on the data path side right for this particular example what kind of hardware would it be two adders. Why two adders? Because we have two editions being simultaneously scheduled into the same clock cycle I need two adders to realize this. However some of those adders could be reused in the second clock cycle and the third clock cycles I do not need more than two adders I could have, but I do not need more than two. What else would be there in addition to the adders?

Student: (Refer Time: 25:27) registers.

I have an adder and that adder that I have two adders, but the inputs come from somewhere and go somewhere, but important thing is that adder may be shared across multiple clock cycles. So, actually mu inputs to one adder may come from multiple sources that implies what else is there in the circuit other than the adder itself there may be a mux right. So, if that is the adder that is being shared then I need muxes that have their inputs coming from different sources right. How many inputs there would be to that mux it depends on how many distinct places you are getting your data for as far as that particular adder is concerned so that is certainly there. What else might be there in addition to the arithmetic components and the muxes?

Student: Register.

There could be registers why do you need registers.

Student : (Refer Time: 26:31).

Values need to be stored since the output in produced in one clock cycle is being used in another clock cycle you would need register in this fine. So that is the kind of data path we are talking about. There are other things the picture is not complete yet you have to take some other decisions before the picture is complete, but there is an example of what the data path looks like.

Fine, so for this kind of a data path the control signal should be clear. Each of these muxes need to be controlled and they are controlled through the select lines which come from the FSM so that is one interface between the FSM and data path. Is there anything else that needs to be sent from the FSM, that register also needs a control signal in general there is a load not the clock, there is a yeah there is a load signal to the register. So, I put it there that load signal tells the register when it should load new data into the register because you might not be interested in loading new data every clock cycle when it is relevant that is when you would load the register. So, such a signal would also come from the FSM. This forms the sort of the interface between the data path and the FSM and clearly these are the signals that can be produced as the actions of the FSM.

So that part is clear what the actions are and what do they translate to, they translate to control signals of the data path fine. That still leaves us with the condition, yeah. So, any guesses what should be what should we have in the condition.

Student: Attention of the operation by the data path.

Student: It must be (Refer Time: 28:57) given as a feedback.

This is the condition that tells us when we should proceed from state S 1 to state S 2 similarly here to there is a condition right right. So, what is the condition under which.

Student: (Refer Time: 29:13).

We should transition from S 1 to S 2.

Student: It must be blocks in that a and added a in the clock (Refer Time: 29:20).

Of course for this circuit to work for this schedule to work it is essential that, what is the relationship between the adder delay and the clock period?

Student: (Refer Time: 29:30).

Adder delay has to be less than the clock period otherwise this does not work. So, both of those pieces of information what is the adder delay and what is the clock period must have been known to the scheduler otherwise you would generate an illegal schedule that the fact that we came up with the schedule means that we are aware and have taken that decision after being aware of these delays which is we according to the methodology we

said that the clock period is an input to the synthesis process. And the library elements the details of the library elements are also visible to us as part of the synthesis process therefore, those that the respective delays are known. Brings us back to the question what should that condition be.

Student: Clock, change in clock.

A change in clock does translate to us proceeding from one state to the other that is the association between a finite state machine and clock. But what would you put there on the condition.

Student: Changing (Refer Time: 30:34).

Student: (Refer Time: :) this.

For me.

Student: Carry this.

If carry exists what does that tell us a carry from the adder which is it would be in the reverse direction. So, these were inputs the control signals were inputs from the FSM to the data path how would the FSM know that there was a carry on an adder. It is the reverse information you have to send that status information back from the data path into the FSM, but the functionally what does it tell us if a carry exists then.

Students: Depends on the algorithm we are proceeding the operation is like carry exists go to (Refer Time: 31:18).

But is that what is the meaning here we go from S 1 to S 2. So, does it do you think there is a dependence on carry, the addition is done which resulted in maybe a carry and perhaps there is no carry that depends on the operands and then we go in the next clock cycle to S 2.

So, what is the, what condition do I put.

Student: (Refer Time: 31:42).

Student: The results of that.

The presence of a valid output on the adder. How do we know whether the adder output is valid or not?

Student: (Refer Time: 31:54) change in the output value from the previous value.

Student: No, no.

Student: have a (Refer Time: 31:58).

Change in output value from the previous value does that. If there is no change what does that mean we are still working?

Student: Yes, sir.

Can is the change in output on the, of the adder enough information to tell us that the computation is done.

Student: No.

No, it might not who knows maybe if the output is the same for two successive editions because the operands were the same operands did not change its still a valid addition it is just that the output not change. I cannot rely on either the carry or the output the some output of the adder. Yet it is an incomplete FSM if I do not specify a condition right.

Student: We have a separate status in a like in a processors status (Refer Time: 32:49) said in the addition (Refer Time: 32:51).

What status signal? This is just, this circuit is so simple that you should be able to complete it.

Student: (Refer Time: 32:59).

What status signal do I rely on?

Student: Sir, rising, rising edge of the clock.

On the rising edge of the clock right. But do we put the clock here as a condition have you written an FSM. You have seen an FSM diagram right do we put the clock there. Clock is implicitly a part of the finite state machine whether it is a state table or whether

it is the diagram we do not put clock explicitly there it is assumed that the system of course, is running on one clock and all these registers and state registers of the flip flop and so on they are working off that clock right maybe it is on the rising edge or of the falling edge, but that is external we do not necessarily put that clock edge as a condition.

Student: Inputs.

Inputs, which input?

Student: Inputs to the adder.

Inputs to the adder are these right.

Student: Right depending on the which input we are taking using mux the statement (Refer Time: 34:03).

What are these inputs anyway I had a b c d if you relate them back to the HDL they are variables maybe they are 32 bit integers or something like that. Is there a point of providing all those 32 bits to the finite state machine, this is just a 3 state machine which would be realized using how many flip flops just two flip flops are needed. So, what is the point of sending the all those 32 bits as inputs to the finite state machine. Remember FSM is not going to do computation addition is going to happen in the data path.

Student: (Refer Time: 34:42) registers.

The register has the output of the addition stored there what would go from the register to indicate to the FSM that you should move on to the next state.

Student: Sir, does the clock (Refer Time: 35:00) actually by default (Refer Time: 35:01) is available.

Yes.

Student: For this transition (Refer Time: 35:05).

Yeah. So, what do I put I the FSM is incomplete without a condition.

Student: By all.

They specified always true means 1.

Student: Yeah.

What variable is 1. So, if there are inputs to the FSM right remember the state table I have present state, next present state inputs, next state and outputs these are my state table. Conditions are realized in terms of the inputs to the FSM and actions are in terms of outputs of the FSM. So, you are right that I should proceed from state S 1 state S 2 irrespective of what else is happening yeah which means that what do I give here if the inputs I should have do not care.

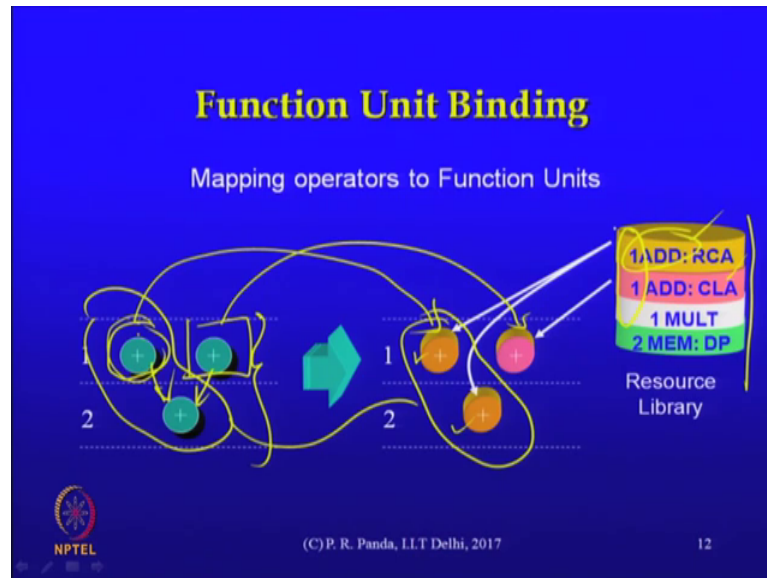
What are the other inputs anyway? Outputs are clear these would be the select signals of the muxes, inputs as of now, we did not need it actually, but there may be some status signal the carryout for example, might be useful as of now, it is not useful right. The fact that the scheduler took the decision of scheduling an operation in a clock cycle means that it is guaranteed that irrespective of what the operand value is the operation will finish in one clock cycle. You do not need anything from the data path to tell you that you should move forward into the next clock cycle there is nothing I need.

Specifically the values on the sum or the outputs of the adder do not tell us that the operation has finished or not finished, but in fact, the scheduler has made that decision, it was made accessible this key information of what is the max delay through the component the adder component that was made accessible to the scheduler. So, there is nothing else that is needed at least for this simple FSM that we have inferred from that schedule in fact, you do not need any input from the data path and therefore, whatever the inputs are as of now, it this would be just I do not care. That transition would correspond to a transition corresponds to a row inside the state table right so that would correspond to it do not care. So, this is a do not care actions are these. So, hopefully we understand how to complete the FSM.

So, two things one is that at the end of the schedule you have the structure of the FSM it is in the complete FSM, but you know for example, how many states are there and then you have to complete it though we are saying that we need these values as of now, I do not know what those values I have to do some other things to send the appropriate two control signals here. So, the outputs are not yet decided, but if the FSM is linear in this way then the conditions are all do not care, you always go from one to the next. You can

see that this is what kind of an FSM it is like a counter like you are just going from one state to the next state without any a counter is an example of such an FSM the clock is an input, but there is no other input unless there may be a reset kind of input, but otherwise from state 1 you go to state 2 you go to state 3, but that is not all like we said a few other things are missing let us complete those.

(Refer Slide Time: 38:23)



There is the step of function unit binding start from the schedule that we came up with right and now, we have to take decision of how we will implement each of those nodes in the graph. We have implicitly assumed here that the input will be stored in the form of a graph what it looks like we will get to later on, but you can see that here just informally a node represents operations and edges draw them here, but these edges represent dependency structures that is important information for the scheduler because the scheduler needs to take this dependencies into account before deciding which one should go into which clock cycle.

But I may have a choice given one of those nodes. I have to now, look at the resource library and decide this one this node would be realized in terms of that component that node would be realized in terms of that other component this they carry look ahead. How to take that decision is a different matter so far we are just saying what is happening in each of the steps. Of course, we introduce schedule, but what is the algorithm for scheduling that we did not say that will come later on. What we are doing now, is quickly

going through the different passes of high level synthesis to understand what happens in each pass. How it works is a different question that we will get back to later.

So, this decision happens as part of the function unit binding step each of those nodes that are being scheduled, each of those operations that are being scheduled we have to map them to an appropriate component in the library that is the function unit binding. So, here the decision that we have made is that there are two adders one of this type and the other of this type and these two operations would be mapped on to the same adder that is what that color coding is indicating say matter here is the ripple carry adder and that other operations would be implemented on that other component. Why it is a different matter? There is an algorithm that would be deciding it, but this is the end result of the function unit binding step.

Student: Sir, there is only one adder or (Refer Time: 41:06) one add adder (Refer Time: 41:07).

Yes, but remember that I have scheduled some operation on that adder in the first clock cycle. In the second clock cycle I am reusing that adder to do something else that is alright right I still need only one component I am reusing it in time which is in space I cannot reuse if in the same clock cycle I need multiple operations to be performed I need them to be performed on different components, but across clock cycles I can reuse component.

Student: Sir.

Yeah.

Student: Is (Refer Time: 41:45) function blocks can we not just is just used to (Refer Time: 41:48) in the same clock cycle.

You can it is just that for this particular example we decided that I will permit the tool only one adder of this type and one adder of that type how many to use of each type is also a decision that the synthesis tool has to make. We will talk about the actual problem formulation for the synthesis tool, but we are just saying that as of now, this design was chosen why it was chosen is a different matter; obviously, obviously that is not the only way to realize their design.

Student: Excuse me sir.

Yeah.

Student: Now, like we are seen that synthesis outputs gives us a FSM and a data path.

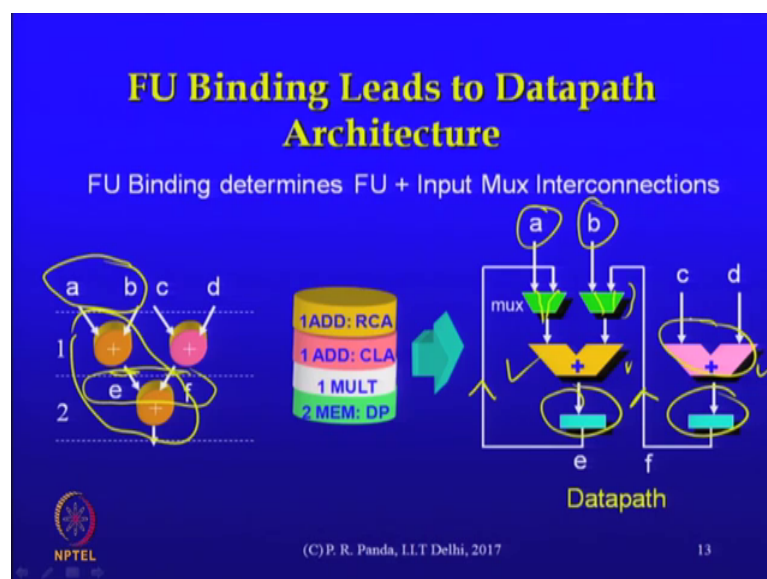
Yes.

Student: So, but some somewhere down the line we will have to convert this FSM also into some hardware.

Yes, indeed like we said the FSM is still symbolic that is not a circuit, but the data path part of it looks more like a netlist this is instantiation of components from the library and there is interconnection a lot of those parts are clear, but the FSM part of it you do need to take it through further synthesis steps. But that would be a little lower level we can put that in an RTL synthesis discussion. But yes there we do need to take that whether there is a diagram or a state table; however, it is that we have specified the FSM we need to we need another step that would take us from that symbolic specification into a netlist into a gate level netlist in terms of flip flops and AND gates, but that is of course, coming. We will be talking about how to synthesize an FSM later on of course.

But an important decision being taken here is which operation will be mapped to which particular component that is what is happening in function unit binding.

(Refer Slide Time: 43:38)



You can see that this takes us closer to the data path architecture the schedule step took us closer to the controller structure the FU binding step leads us to at least an outline of the data path architecture. What kind of outline? It tells us the following first of all these two resources are used right these two adders are used. Then remember there is a reuse of that adder being implied by that sharing of that adder across time and therefore, that may lead to a mux that is inferred at the inputs of that adder. The second adder does not need a mux because there is only one addition being mapped to it anyway.

Now, I have the adders, I have these muxes again they are possible muxes, what else is there what kind of multiplexing I need to do. This adder like, this adder needs to perform a plus b in one clock cycle in the other clock cycle it needs to perform e plus f right so that is what. So, I need to be able to send a and b in one clock cycle to the adder and I need to be able to send e and f in a different clock cycle to the adder which means that I could have a structure like this two muxes at the inputs and a and b are one input each to the two muxes so that in one clock cycle I can enable that path and I can also simultaneously enable this path, leading a and b to proceed to the adder.

In the other clock cycle I need to send e and f the select inputs to the two muxes need to be coordinated so that the correct set of operands are arriving at the adder in the same clock cycle. I need to send e plus f, e and f in the second clock cycle. Now, where are E and f, I performed these two additions right in the first clock cycle at the end of it I could store them in a register with the outputs are stored in a register those are my e and f. So, I need paths to be provided this path and that other path from the register into that second input of the respective muxes so that e and f could be simultaneously forwarded to the adder in the second clock cycle.

So, hopefully this design is clear I am performing only c plus d in the second adder. So, there is no multiplexing that is needed there actually most of our design is completed here so that if your binding is an important logical step that needs to be performed which helps us get closer to the data path architecture.

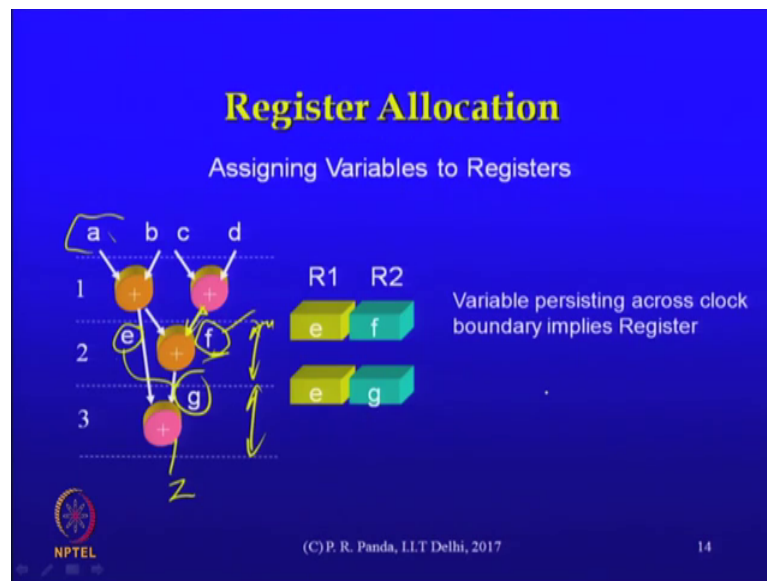
Student: That will be (Refer Time: 46:29) constraints on muxes also like muxes and.

There could be constrained where do we get these muxes from we said that we pick up the adders from the component library muxes we would also pick up from the component library. It would usually be parameterized in some way because it is hard to

decide in advance what kind of muxes I should use as it turned out here I need 2 to 1 mux, but if the you see that if the level of sharing was 3 way then I would need a 3 to 1 mux right. So, there would be different kind either the I have 2 to 1, 4 to 1, 8 to 1 and so on explicit discrete components of the muxes, which are there in the library. It could also be that I have a generator of a mux in the library to which I give a parameter that says I need a 5 to 1 mux and it gives me the associated component, but the properties of that component would also have to be generated say this is the a projected area of such a component and this is the projected delay of that component.

We need to be aware of the delays of course, these are also combinational logic and they add delays there, but we will get back to the implication of the muxes particularly as far as timing is concerned it is a very interesting question. But for now, let us just say that the muxes are just inferred from the selection and the sharing decisions that are made in the FU binding step.

(Refer Slide Time: 48:03)

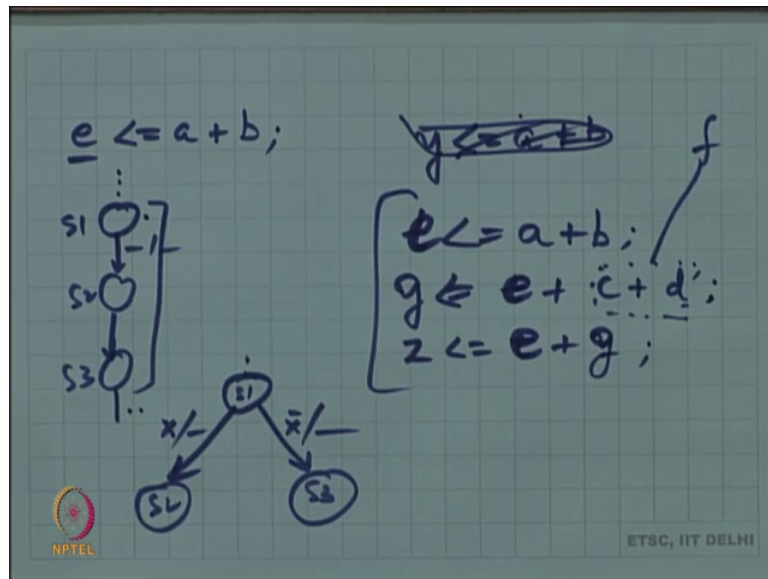


Finally, there is a register allocation step that needs to be done. So, let us assume that a b c d are inputs to the design. So, in the entity they are input ports. So, the values come from somewhere we do not need to worry about it, but I have these outputs from intermediate stages intermediate clock cycles that are used as inputs to operators in the subsequent clock cycles. That means, that I need to store them in registers how do I infer the presence of registers from a schedule essentially if you have an edge the dependence

edge like that crossing a clock cycle boundary then that implies there is a storage that is required because I may be using that component for something else in the future clock cycles I need to store the values that implies registers what kind of storage we are talking about these are register storage.

So, I need in general to assign these variables remember these variables are inferred irrespective of whether or not you explicitly had them declared in the code right.

(Refer Slide Time: 49:19)



So, you could have variables of this nature I have them as signals now, but in fact, even if they were process variables the treatment would still be the same as far as the synthesis is concerned. So, it could be that you have this `e` and `f` being explicitly declared in the code it could also be that you have a statement like this. Let us say I have `t` equals `a` plus `b` `y` equals `t` plus `c` plus `d` and you have the output which are not labeled here, but there would be `z` equals `t` plus `y` changes to `e` instead of `y`, let us call it `g` to be consistent with the picture fine.

So, I have the `e` and the `g` being explicitly named here as intermediate signals, but that `f` which corresponds to just the `c` plus `d` right that is not explicitly named it is something that is inferred it is just some temporary and variable that the synthesis tool has internally created. That needs to be stored, so I have variables that are either explicitly declared or they are implicitly inferred if you have a complex computation all as part of a large

expression you need to break them down into smaller units some of which may need to be stored.

So, if you have these dependencies crossing clock cycle boundaries then there is the need for inferring register they need to be stored. The question of course, is that I have e f and g being such variables that need to be stored here 3 variables, but there is an interesting register allocation problem that arises where I make the decision of storing these variables in registers because I need to ask this question of how many registers are needed, in this example how many registers do we need to store those 3 variables I can have 3 registers of course, but I do not need all of 3 registers how do I decide how many I need, why would I be able to do with less than 3 registers.

Student: (Refer Time: 52:30).

Registers can be reused just like those data path elements those ALUs were reused because they were freed up once they did their job in one clock cycle in a new clock cycle you could use it for some other computation, the same logic applies to registers also. The same register that was used to store f at that clock boundary for this duration its freed up in the next clock cycle and can be used to store something else. So that leads to an interesting problem to be solved. I have a bunch of variables in the design that has already been scheduled and now, I need to infer a bunch of registers what is the mapping between variables and registers that possibly minimizes the number of registers I use. Could I have reused the same register for e and g, the same register could not have been reused to store both e and g why?

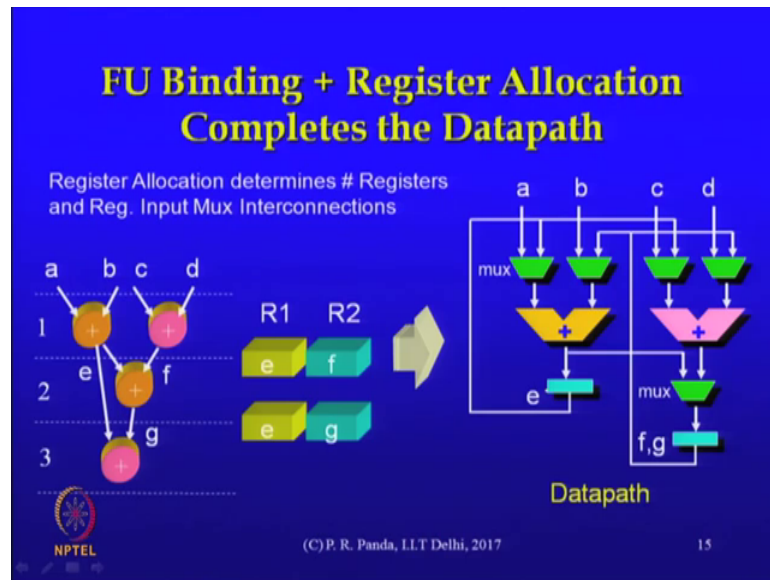
Student: (Refer Time: 53:34).

Both of them need to be stored at the same time. So, somehow I need to develop a formulation where this idea is formalized a little bit when do you need two variables at the same time. If you need two variables at the same time then they have to be in distinct registers if you do not need them at the same time then you could share registers.

So, we started off with a scheduling as an important step that step helped us in getting the FSM structure at least part of the efficient structure we could not complete the FSM, but at least we got most of the structure of the FSM because of the scheduling. Then we did function unit binding that step helped us arrive at most of the data path components at

least this part of the data path the resources computation resources part of the data path and these muxes at the input of those data paths. But finally, there is a register allocation step that helps us arrive at these decisions how many registers do I need and what are the variables that should be stored in each of those registers.

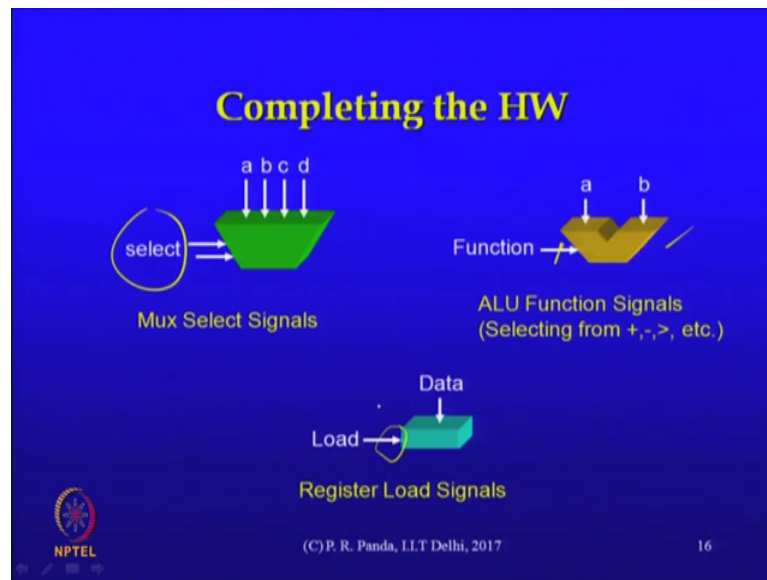
(Refer Slide Time: 54:56)



These are sort of the 3 major steps in high level synthesis, but with the FU binding and the register allocation your data path is complete.

What else remains after this? Our FSM was not quite complete we had the states, we had their transitions and we did have an idea about the conditions as of now, those are simple do not care conditions, but the outputs actually I need to provide the right select signals into the muxes.

(Refer Slide Time: 55:30)

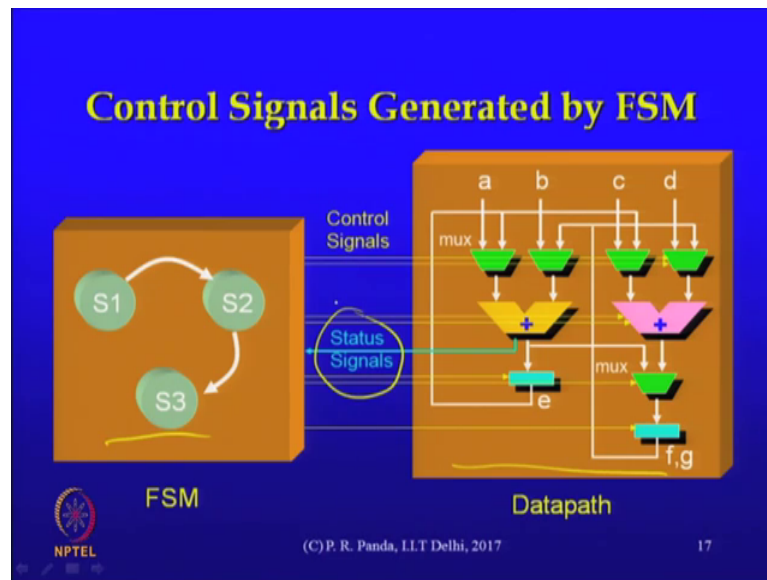


So, there are a few other things to complete the hardware the mux select signals need to be decided which resource should be connected to which input of the mux the order is important right. So, those I need to decide and correspondingly I need to fill in the details on the FSM action table.

In general I need a bunch of ALU function signals what are these a given ALU may be capable of doing more than one function the adder that we saw was just doing added nothing else. So, there is no other function a function input in there, but an ALU may be configured to do an addition in one clock cycle subtraction in some other clock cycle a comparison in a different clock cycle. So, you need a few other bits here to say in that clock cycle what should this ALU be doing so that is what we call a function, where does this information come from? The FSM it is the FSM that knows what is the operation that should be performed and now, we also know which ALU should be performing this so that these bits should be coming from the FSM.

Then there is this load input of a register. At every clock cycle you do not necessarily load a register with new data only when you have data that is useful then you need to store it so that information also needs to be provided by the finite state machine. So, that is the other part that is necessary for us to complete the hardware.

(Refer Slide Time: 57:06)



So, these control signals would be generated by the FSM bringing us back to that picture hopefully this high level overview gave us an idea of what are the steps and how they help us in getting closer to the hardware.

Of course, the step is still there that FSM is symbolic this is not quite a netlist this part is of course, closer to netlist these are just instantiated components and the interconnections are also clear here they are not clear, but that is something that we already know how to do. Given an FSM you know how to ultimately translate it into a circuit with D flip flops and gates we will get there, but that part is already you are familiar. So, the process of the inference of the FSM is what is the part of the HLS task.

Then let us get back to this status signal when is this useful. It was not useful in this particular example we had so, but when will it be used.

Student: One operation takes more than one clock cycles.

If an operation takes more than one clock cycle what kind of status signal is useful.

Student: So, then we would need to tell the assembled (Refer Time: 58:17).

If an operation takes a variable number of clock cycles then there may be a need to perform a handshaking where that component actually sends a signal saying it is done. However, if that component takes a fixed number of clock cycles if I know that it will

take 3 clock cycles and no more than 3 clock cycles then I do not need to rely on that status right what can I do on the FSM side, if I know that a component takes exactly 3 clock cycles then.

Student: (Refer Time: 58:49) count the clock cycles,

If this was my first state there were other states therefore, I just introduced two more clock cycles right if it was taking 3 clock cycles I know it is taking 3 clock cycles then I just introduced two more dummy states in the FSM and pick up its value later on, anyway the FSM does not pick up. The value the values probably are going as the output and back into the input of some other component it only sends the control signal, but if it is small then the way to incorporate that in the FSM is to just have the right number of states wait for the right number of states do not need an explicit counter there, but if the number of states is likely to be large number of cycles is likely to be large then perhaps there is a an explicit counter that could be incorporated and that counter could be there in the data path the counter is a data path element really.

So, again the counter may have an output signal that says it is done or whatever if you are counting up to 10 end of 10 you may send a signal that could be a status signal that goes back to the FSM that is one kind. What other kind of status signal might be useful for us.

Student: Multiple status (Refer Time: 60:11) I mean (Refer Time: 60:11) status to multiple status.

Right, if you are in one state and well these were linear structures that we saw so far if the structure might not be linear right you may be the next step may not be obvious so far the next step was obvious and therefore, that condition was actually do not care here, but it may not be obvious, you may be dependent on something that is the result of the computation from the data path. It could be that you have a comparator on the data path and the output of the comparator says true or false and depending on whether particular output is true or false you may go one way or let us say that input is taken here as x then on x you go to one state on \bar{x} you go to the other state. So that is the other kind of status signals that we are talking about.

The moment you have an if statement the moment you have conditionals in the HDLs chances are that they may translate in the FSM to a forking structure like this of the FSM. And that condition under which you are going one way or the other may be dependent on some input that is that is externally an input to the entity itself or it could to the entire design itself, but it could also be an input that comes as a status signal from the data path side of the computation. It could be a result of some computation that happened in the previous clock cycle that is used to achieve the fork in the FSM.

Let us conclude here.