Part
Indian Institute of technology Madras
Presents

NPTEL
NATIONAL PROGRAMME ON TECHNOLOGY ENHANCED LEARNING

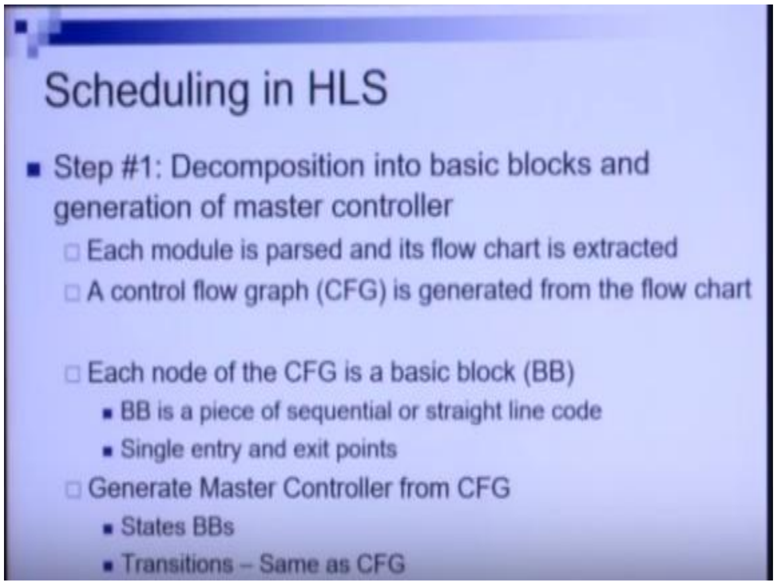
Lecture-6

VLSI Design Verification and Test

Dr. S. Sankaran
Associate Professor
Department of Metallurgical and Materials Engineering
IIT Madras
Email: ssankaran@iitm.ac.in

Hello. Welcome to module one, lecture three of the course VLSI Design Verification and Test. In this module, we will take a deeper look into the scheduling of operations within a basic block in high-level synthesis.

(Refer Slide Time: 00:21)



Scheduling in HLS

- Step #1: Decomposition into basic blocks and generation of master controller
  - Each module is parsed and its flow chart is extracted
  - A control flow graph (CFG) is generated from the flow chart
  - Each node of the CFG is a basic block (BB)
    - BB is a piece of sequential or straight line code
    - Single entry and exit points
  - Generate Master Controller from CFG
    - States BBs
    - Transitions – Same as CFG

So, before going into the scheduling, a bit of a background, how have we come to this place? So firstly, we said that the application will be defined in terms of a high-level language, high-level hardware description language like Verilog and describing its behavior. Now, after we have obtained this Verilog code, each module of this Verilog code will be parsed and the flowchart from this module will be extracted.

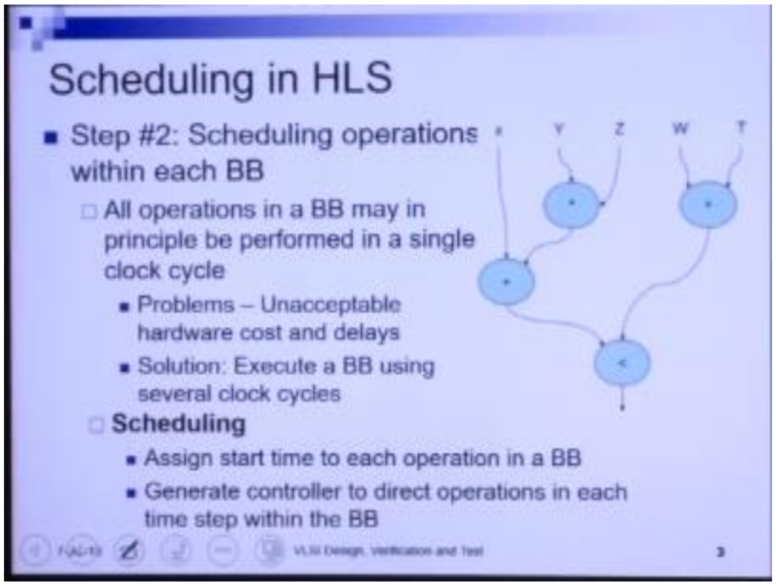
Now corresponding to this flowchart, we will be able to obtain a control flow graph of the module. So what will be -- what will the control flow graph show? The control flow graph will show how the control flows through various paths in the module. So there would be various mutually exclusive tasks or operations that will be conducted and the control will decide which mutually exclusive sets of operations will be chosen and which will be not based on data input.

So now, how does a control flow graph look like? A control flow graph -- in a control flow graph, the nodes are basic blocks as we said before. A basic block is a piece of sequential or straight-line code with a single entry point and single exit point. Control cannot come into in the middle of the basic block and control also cannot go out of the basic block in the middle.

So control comes at the beginning of the first treatment of the basic block and goes out at the end of the basic block. So, and edges, what do they denote? They denote the dependency among the basic blocks. Now we also said that given this control flow graph with basic blocks and the dependencies, I can directly generate the master controller. So what will the master controller control? The master controller will control when will the operations in a basic block be initiated, right?

However, as we also said before that the complete master controller cannot be generated at this step because the complete master controller will be a tool that will allow us to ascribe timing information to each operation in a hardware synthesized circuit. However, until we have not ascribed explicit timing information to each operation within a basic block, we cannot ascribe the complete timing information of the module. So, therefore, to obtain the complete master controller, we need to ascribe timing within a basic block, timing to operations within a basic block, and timing to operations within a basic block is done through scheduling.

(Refer Slide Time: 03:28)



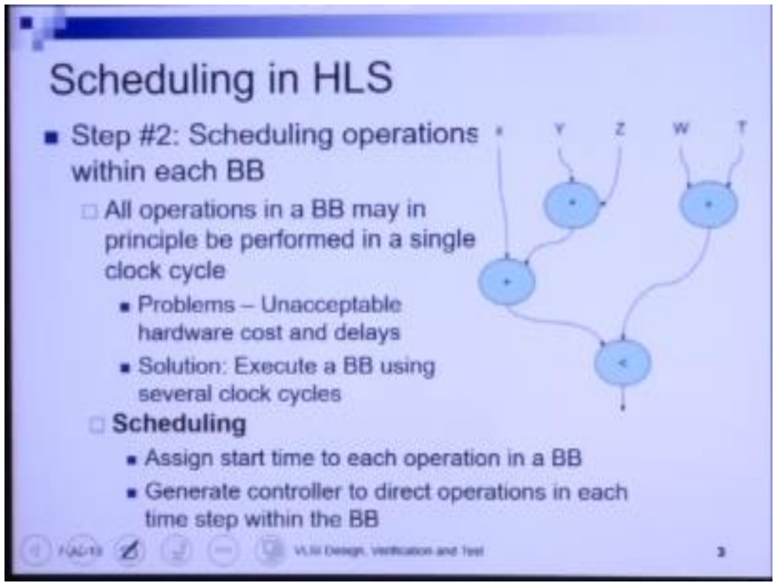
Scheduling in HLS

- Step #2: Scheduling operations within each BB
  - All operations in a BB may in principle be performed in a single clock cycle
    - Problems – Unacceptable hardware cost and delays
    - Solution: Execute a BB using several clock cycles
  - **Scheduling**
    - Assign start time to each operation in a BB
    - Generate controller to direct operations in each time step within the BB

So next we have now arrived within a basic block and we have to schedule operations within a basic block. Now before going into scheduling, we also said this before that all operations within a basic block can ideally be scheduled in a single clock cycle. However, if we do so then there will be unacceptable hardware costs and delays.

So why can it all -- how can all operations be done in a single clock cycle? Because there are no mutually exclusive operations within a basic block, all operations are performed. It’s simply a data flow graph showing the dependency of the data. So data comes in, gets transformed through operations and goes out at the end of the basic block. There are no control operations in it. There are no mutually exclusive operations in it. So, however, the length of the clock cycle has to be at least equal to the propagation, the total propagation delay in the critical path.

(Refer Slide Time: 04:45)



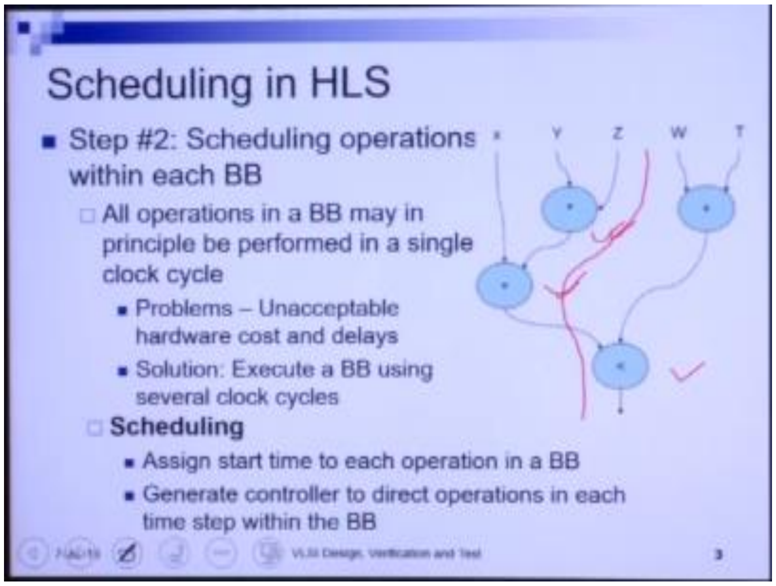
Scheduling in HLS

- Step #2: Scheduling operations within each BB
  - All operations in a BB may in principle be performed in a single clock cycle
    - Problems – Unacceptable hardware cost and delays
    - Solution: Execute a BB using several clock cycles
  - **Scheduling**
    - Assign start time to each operation in a BB
    - Generate controller to direct operations in each time step within the BB

So what do we mean by this? So each of the operations, this one, this one, this one, each of them has a propagation delay because it has to be executed through a resource. And what is a critical path? Critical path is the highest weighted path from the source to sink inside a basic block. For example, this path has three operations, this plus this plus this. So this is a critical path now the clock cycle has to be at least big enough so that all three operations can be performed in sequence through this basic block, right?

And therefore, because this clock will also feed to other operations, other throughout the circuit at different places, so this clock cycle although is sufficient for this basic block may cause unnecessary delay in other basic blocks. Well, let us say in that basic block, I have two sequential operations like this and this. I don't have the third. So, therefore, it will perform these two operations and wait because the length of the clock cycle is such that three sequential operations can be executed. So the clock cycle length has to be judiciously decided and adjusted according to the circuit that we have, according to the need that we have.

(Refer Slide Time: 06:28)



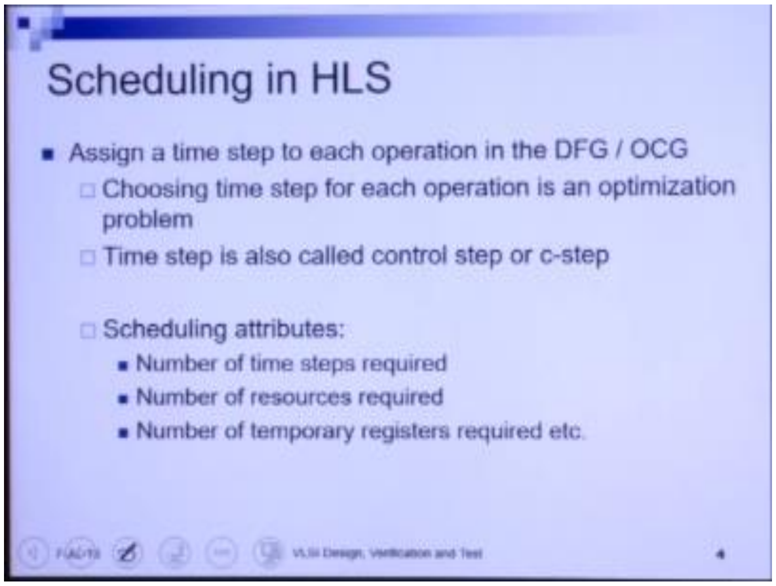
Scheduling in HLS

- Step #2: Scheduling operations within each BB
  - All operations in a BB may in principle be performed in a single clock cycle
    - Problems – Unacceptable hardware cost and delays
    - Solution: Execute a BB using several clock cycles
  - **Scheduling**
    - Assign start time to each operation in a BB
    - Generate controller to direct operations in each time step within the BB

Now why did we say all this? Because then we need to ascribe timing. If we don't do everything in one clock cycle, we need to do different operations in different clock cycles. So, therefore, ascribing such operation to different clock cycles is the very problem of scheduling. Ascribing the timing information to each operation within a basic block is the problem of scheduling.

So what does scheduling do? It assigns a start time to each operation in a basic block and the controller, it generates -- it allows the generation of the controller so that those operations can be directed as those time steps within the basic block.

(Refer Slide Time: 07:22)



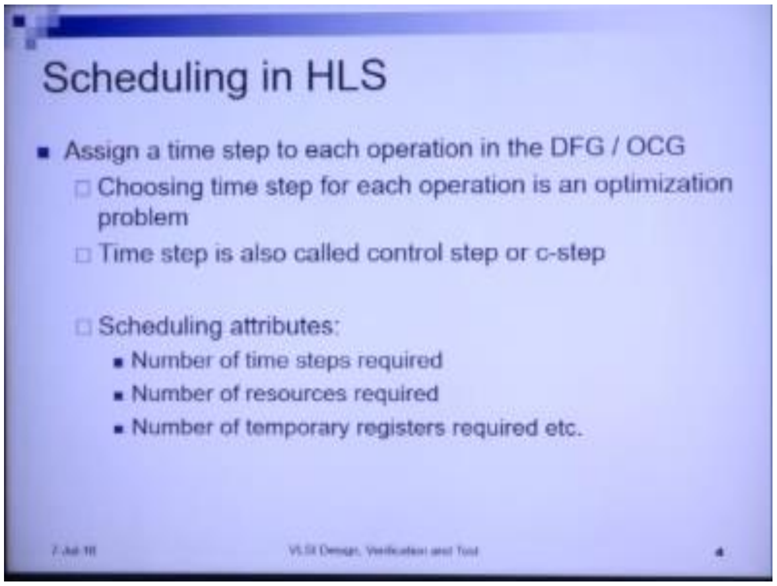
Scheduling in HLS

- Assign a time step to each operation in the DFG / OCG
  - Choosing time step for each operation is an optimization problem
  - Time step is also called control step or c-step
  - Scheduling attributes:
    - Number of time steps required
    - Number of resources required
    - Number of temporary registers required etc.

VLSI Design, Verification and Test 4

Now choosing a time step for each operation in a basic block is basically an optimization problem, as we will see in depth in the next two classes. The time step is also called a control step or a c-step.

(Refer Slide Time: 07:55)



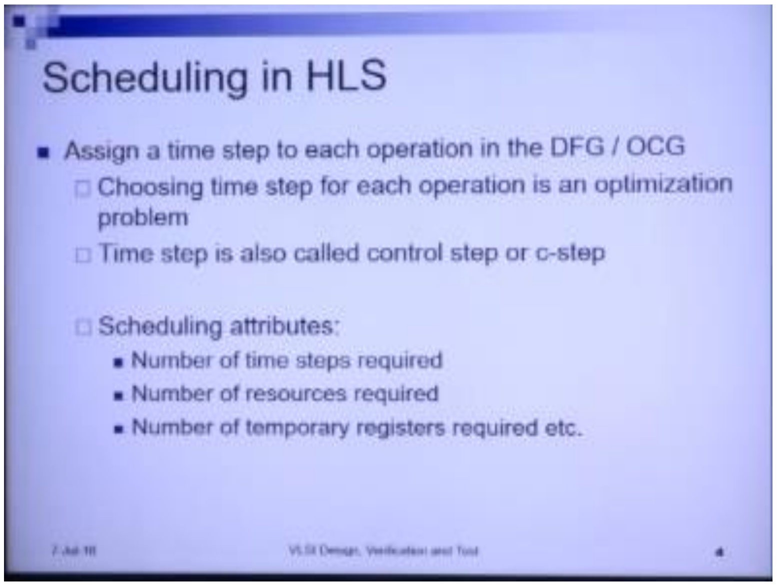
Scheduling in HLS

- Assign a time step to each operation in the DFG / OCG
  - Choosing time step for each operation is an optimization problem
  - Time step is also called control step or c-step
  - Scheduling attributes:
    - Number of time steps required
    - Number of resources required
    - Number of temporary registers required etc.

7 Jul 10 VLSI Design, Verification and Test 4

So scheduling is the task of assigning a time step to each operation in the data flow graph or the operation constraints graph. However, choosing a time step for each operation in general is not a trivial problem. It is generally an NP-complete optimization problem as we will look inside in depth, and depending on the performance of a schedule, the performance or how good the schedule is can be characterized through various attributes.

(Refer Slide Time: 08:27)



Scheduling in HLS

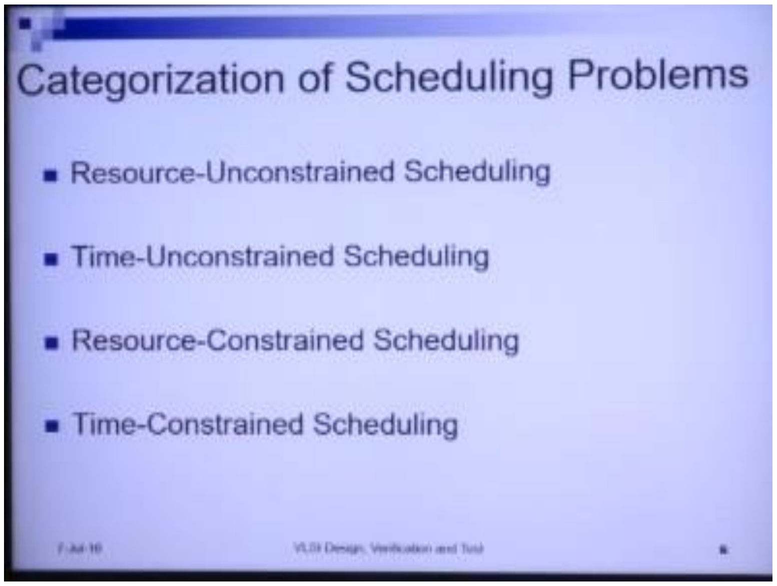
- Assign a time step to each operation in the DFG / OCG
  - Choosing time step for each operation is an optimization problem
  - Time step is also called control step or c-step
  - Scheduling attributes:
    - Number of time steps required
    - Number of resources required
    - Number of temporary registers required etc.

7 Jul 10 VLSI Design, Verification and Test 4

For example, the number of time steps that the schedule took to assign time steps to all operations. So the lower the number of time steps within which all operations in the basic block can be scheduled, better is it in terms of performance of the circuit, faster will be that circuit. However, in general, to perform in a lower number of time steps requires higher resources, more resources as we will also see.

So the next attribute is how much resources were consumed by the schedule and other attributes could be how many temporary registers were required MUXes, DEMUXes etc., as we will see.

(Refer Slide Time: 09:31)



Categorization of Scheduling Problems

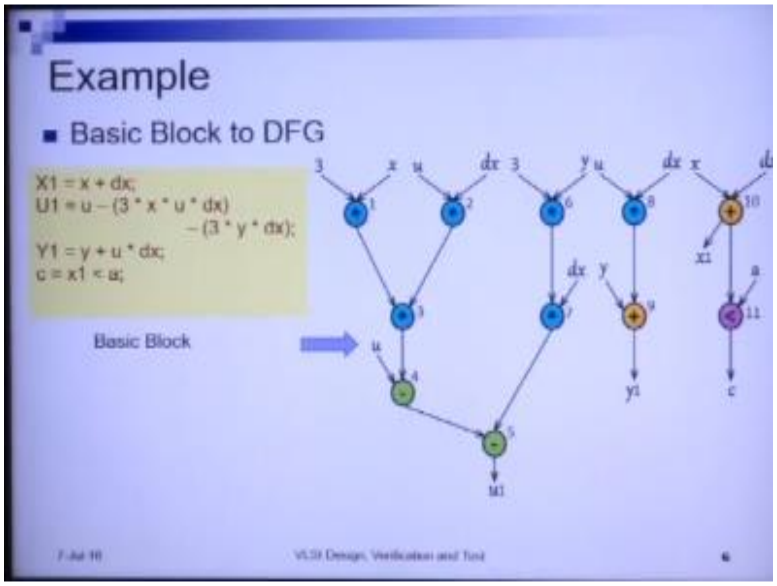
- Resource-Unconstrained Scheduling
- Time-Unconstrained Scheduling
- Resource-Constrained Scheduling
- Time-Constrained Scheduling

So depending on the type of scheduling that we are doing, depending on the number of resources that we have, the amount of time that we are taking to schedule, there are principally four different kinds of scheduling problems that are there. Literature says, there are four different kinds of scheduling problems within a basic block.

The first one being resource-unconstrained scheduling where we have no constraints on the amount of resources that we can use. The second one is time-unconstrained scheduling where performance is not an important factor. We want to take as low resources as possible and delay is not a matter to us. So, typically, such circuits or such chips would be -- we would like them to be very small in size. That is why we want to have a critical view on the resource consumed and not so on time.

So the next two cases are more generalized scheduling problems: resource-constrained scheduling and time-constrained scheduling. We will look at each of them one by one.

(Refer Slide Time: 10:44)



Example

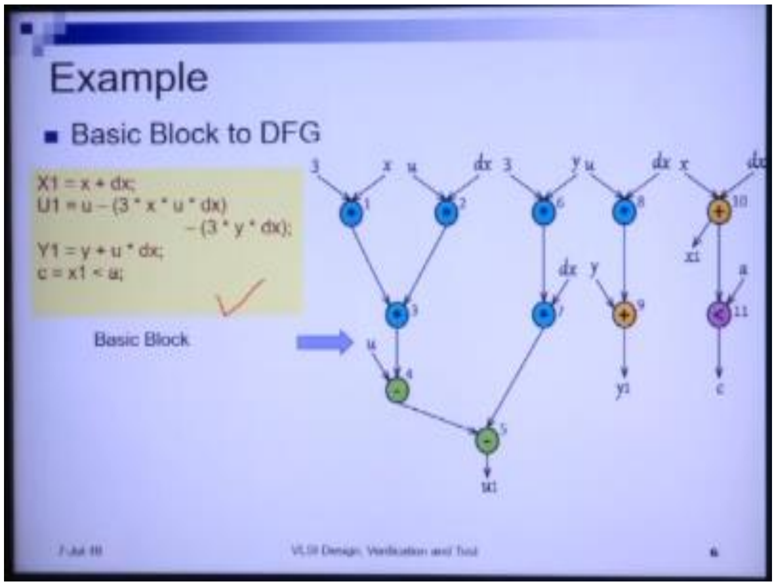
- Basic Block to DFG

```
X1 = x + dx;
U1 = u - (3 * x * u * dx)
               - (3 * y * dx);
Y1 = y + u * dx;
c = x1 < a;
```

Basic Block

So before going into the scheduling problem, we will take a general example, this running example, we will be using throughout the topic of scheduling that we will be dealing with. So on the left of the screen, we have a basic block. So this can be assumed to be the hardware description in Verilog, a behavioral description of a basic block. So X1 = x + dx; U1 = u - (3 * x * u * dx) - (3 * y * dx); Y1 = y + u * dx; and the last one c = x < a, now the corresponding data flow graph of this basic block is shown on the right of the slide. So see the operation one takes as input two independent variables 3 and x.

(Refer Slide Time: 11:45)



Example

- Basic Block to DFG

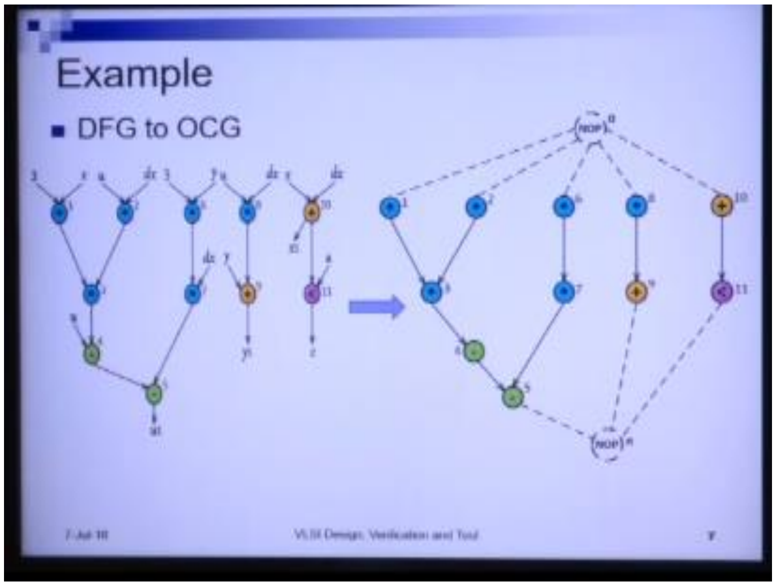
```
X1 = x + dx;
U1 = u – (3 * x * u * dx)
                  – (3 * y * dx);
Y1 = y + u * dx;
c = x1 < a;
```

Basic Block

Operation two takes two independent variables u and dx. So operation three performs the operation 3 into x into u into dx. And operation six, for example, performs the operation 3 into y. Operation seven performs the operation 3 into y into dx. And then here operation four performs u minus 3 into x into u into dx, and in five we obtain the final U1.

Similarly, we obtain the Y1 and c variables as outputs of 9 and 11 operation, the operation number 9 and operation number 11, respectively. Now given this data flow graph, we have to obtain the operation constraints graph.

(Refer Slide Time: 12:42)

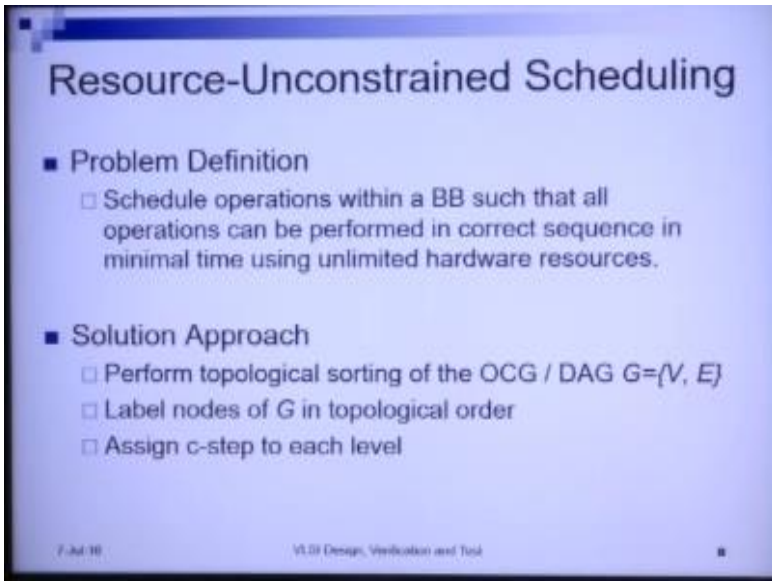


Example

- DFG to OCG

So this -- the data flow graph that we obtained in the last slide is shown on the left-hand side of the slide and the corresponding operation constraints graph is shown on the right side. Here the independent variables have been removed. We have made the graph polar having a single source node and a single sink node and we have only shown the dependencies among the operations, right. So in this way we have obtained the operation constraint graph. So this operation constraint graph basically we will be using throughout the topic of scheduling.

(Refer Slide Time: 13:25)



Resource-Unconstrained Scheduling

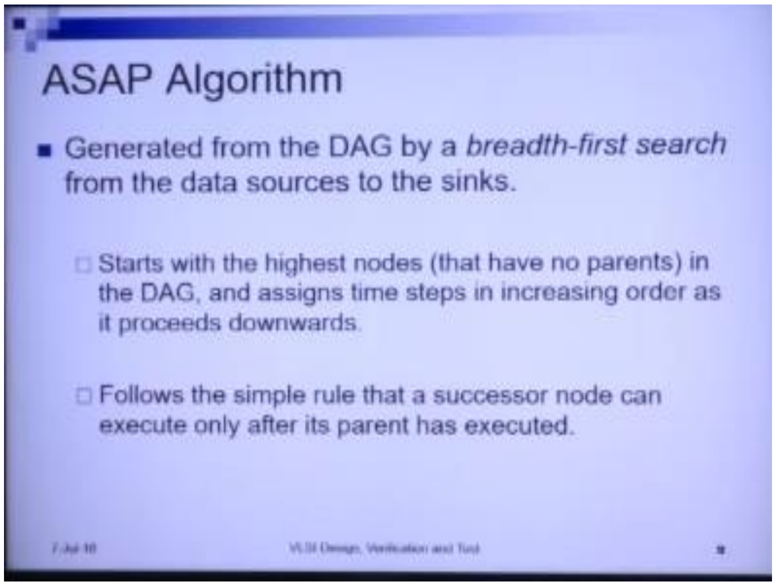
- Problem Definition
  - Schedule operations within a BB such that all operations can be performed in correct sequence in minimal time using unlimited hardware resources.
- Solution Approach
  - Perform topological sorting of the OCG / DAG *G={V, E}*
  - Label nodes of *G* in topological order
  - Assign c-step to each level

7-Jul-16 VLSI Design, Verification and Test

So the first type of scheduling that we had said was Resource-Unconstrained Scheduling. So the definition is -- the problem definition is this: “Schedule operations within a basic block such that all operations can be performed in correct sequence in minimal time using unlimited hardware resources.” So there is no limitation on the hardware resources that we have.

Hence the solution approach is to just topologically sort the operations constraint graph that we have and label the nodes in topological order. So this topological order will give me the schedule because finally, we just have to assign a distinct c-step to each level that we have obtained in the after topological ordering of the graph.

(Refer Slide Time: 14:22)



ASAP Algorithm

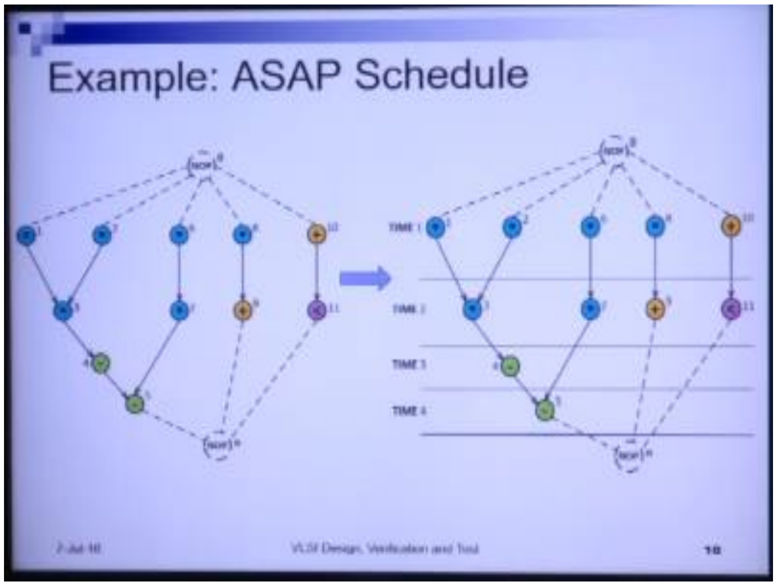
- Generated from the DAG by a *breadth-first search* from the data sources to the sinks.
  - Starts with the highest nodes (that have no parents) in the DAG, and assigns time steps in increasing order as it proceeds downwards.
  - Follows the simple rule that a successor node can execute only after its parent has executed.

7-Jul-10 VLSI Design, Verification and Test

So such an approach has been adopted in the ASAP algorithm. So it's generated from the DAG or operation constraint graph by a breadth-first search from data sources to sinks. So how does it operate? It starts with the highest nodes (that have no parents), the independent nodes in the DAG, so all independent nodes are scheduled in level one and then time steps are assigned in increasing order as we proceed downwards.

So after we have scheduled the independent operations, a few more nodes become independent and ready for scheduling, and they are scheduled and we perform as many nodes, as many ready nodes are there, we schedule all the ready nodes that are available at a particular time step to that step. So it follows the simple rule that a successor node can execute only after its parent has executed.

(Refer Slide Time: 15:21)

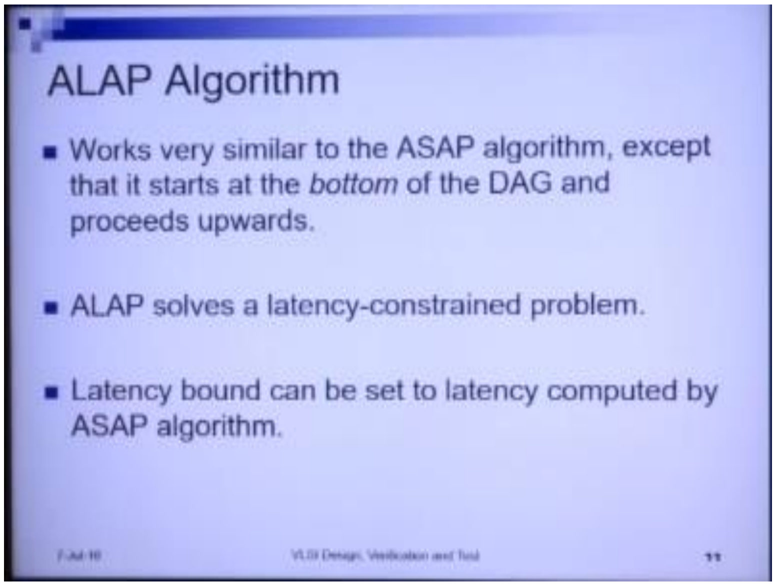


Example: ASAP Schedule

So an example, the operation constraint graph that we obtained is shown in the left and on the right we have the corresponding ASAP schedule. See that all the independent operations, this, this, this, this and this have been scheduled in step one and then after these have been scheduled, then 3 becomes ready, 7 becomes ready, 9 becomes ready, 11 becomes ready and they are all scheduled in step two. After they have been -- after 3 has been scheduled, 4 can be scheduled, so four is executed in time step 3; 5 can be scheduled after 4 has been scheduled. So 5 has been scheduled in time step 4, right.

So we will therefore get a very fast performing basic block if we do an ASAP schedule, but obviously, we need unlimited hardware. For example, in step 1, we required four multiplication operations and if we had a multiplier that will perform these operations -- performing these operations, we need four multipliers, because four multiplication operations are being performed concurrently in step 1 and hence the area of the obtained circuit will be high.

(Refer Slide Time: 16:56)



ALAP Algorithm

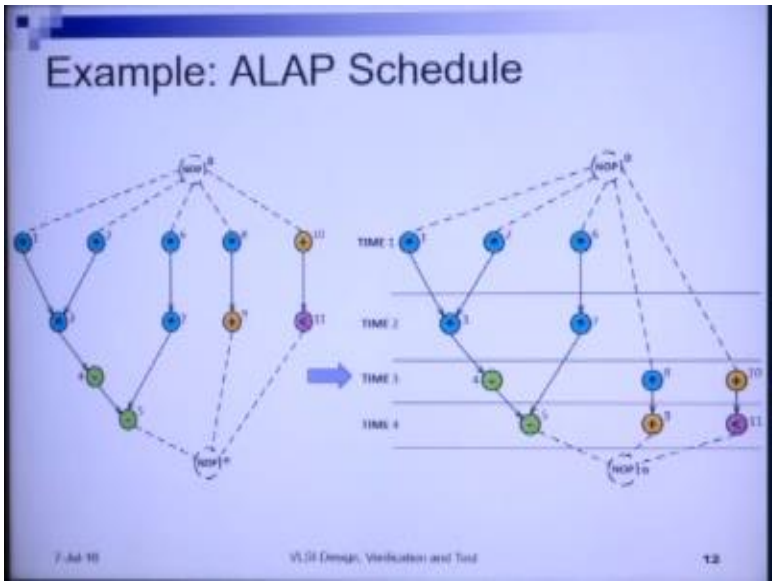
- Works very similar to the ASAP algorithm, except that it starts at the *bottom* of the DAG and proceeds upwards.
- ALAP solves a latency-constrained problem.
- Latency bound can be set to latency computed by ASAP algorithm.

7-Jul-10 VLSI Design, Verification and Test 11

Now a similar algorithm is the ALAP Algorithm. How is it different from ASAP? ASAP has been obtained from as soon as possible schedule. ALAP on the -- ALAP correspondingly has the name as late as possible, right? It works very similar to the ASAP algorithm except that it starts at the bottom of the DAG and proceeds upwards. ALAP solves the latency-constrained problem.

And the latency bound, we can give any latency bound that is suitable to us. Obviously, we can give the latency bound that has been computed by the ASAP algorithm. For example, in the previous ASAP schedule, we had obtained a latency bound of four. So, if you use the same latency bound four and apply it to the ALAP algorithm, we will update something like this.

(Refer Slide Time: 17:47)

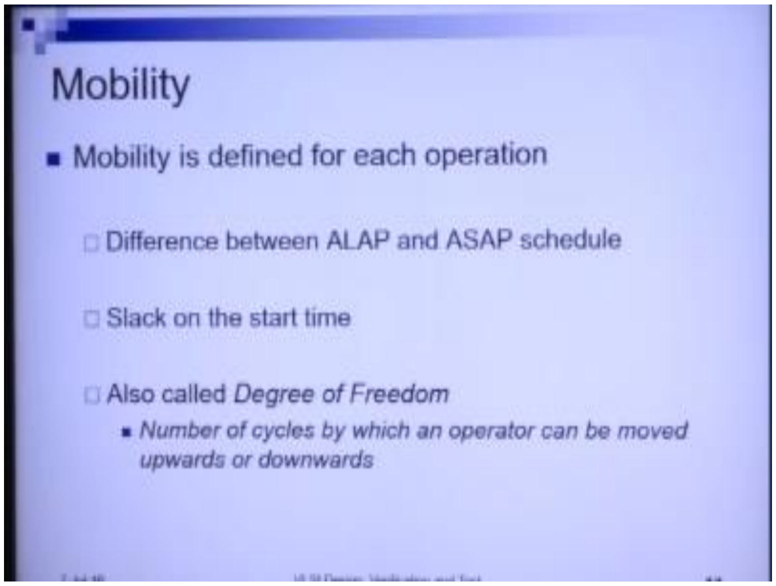


Example: ALAP Schedule

On the left hand side, again, the operation constraint, the unscheduled operation constraints graph has been shown. On the right side, we obtain the ALAP schedule. So ALAP schedule as we said, the ASAP schedule starts from the source towards the sink. ALAP schedule progresses from the sink towards the source. So the NOP has been scheduled -- the start of the NOP operation is at time step 5. So all operations on which NOP depends can be scheduled in the previous time step at time step 4. So NOP depends on operations 5, 9 and 11. So these three operations have been scheduled in time step 4.

At time step 3, all operations on which 5, 9, 11 depends can be scheduled. So 4, 8 and 10 has been scheduled in time step 3. In time step 2, 3 and 7 has been scheduled, similarly, and in time step 1, 1, 2 and 6 has been scheduled. So this is what is the ALAP algorithm.

(Refer Slide Time: 19:05)

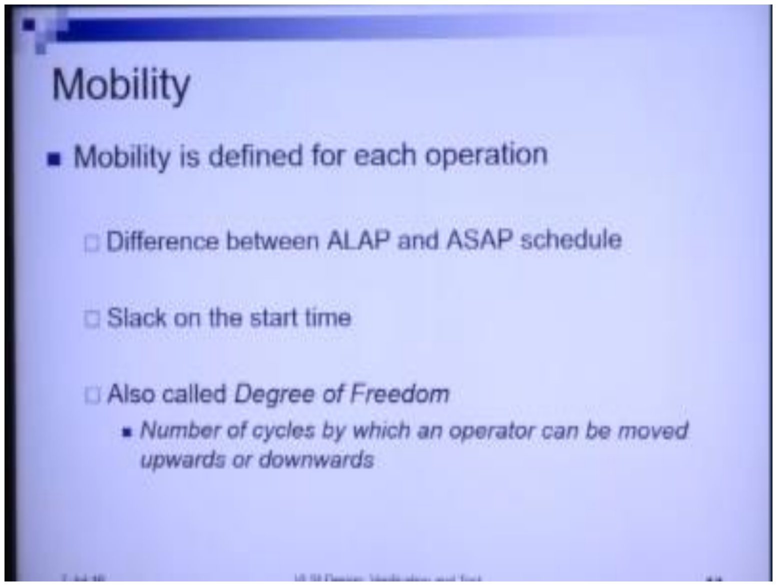


Mobility

- Mobility is defined for each operation
  - Difference between ALAP and ASAP schedule
  - Slack on the start time
  - Also called *Degree of Freedom*
    - *Number of cycles by which an operator can be moved upwards or downwards*

Now from the ASAP and ALAP algorithm, a few very important concepts can be obtained, and these concepts are applied in a more generalized scheduling problems that we will take up in due course in the next few slides. So mobility is defined for each operation. What is mobility?

(Refer Slide Time: 19:35)

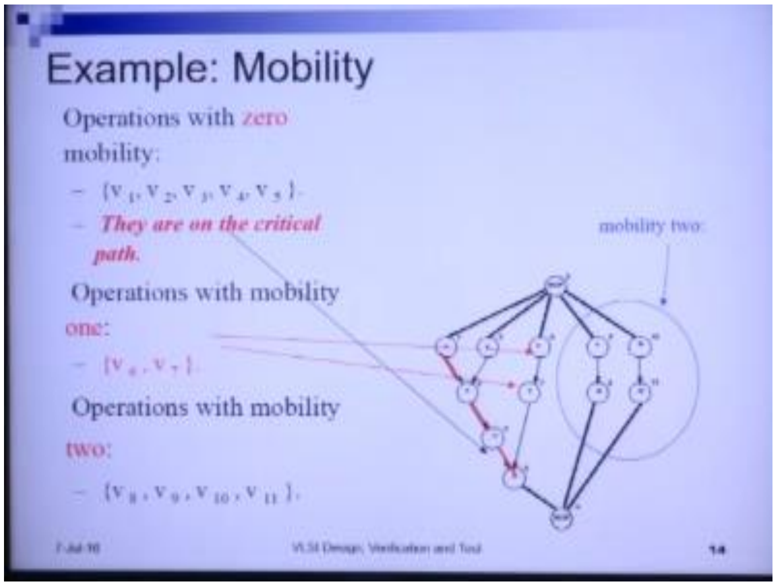


Mobility

- Mobility is defined for each operation
  - Difference between ALAP and ASAP schedule
  - Slack on the start time
  - Also called *Degree of Freedom*
    - *Number of cycles by which an operator can be moved upwards or downwards*

Mobility is the difference between the ALAP time and the ASAP time of a schedule. And what does it provide this difference between ALAP and ASAP provides a measure of the slack that is available on the start time. So this mobility, this term mobility is also called the degree of freedom of an operation node because it defines the number of cycles by which an operation can be moved upwards or downwards in the schedule.

(Refer Slide Time: 20:08)



Example: Mobility

Operations with zero mobility:

- $\{v_1, v_2, v_3, v_4, v_5\}$.
- *They are on the critical path.*

Operations with mobility one:

- $\{v_6, v_7\}$.

Operations with mobility two:

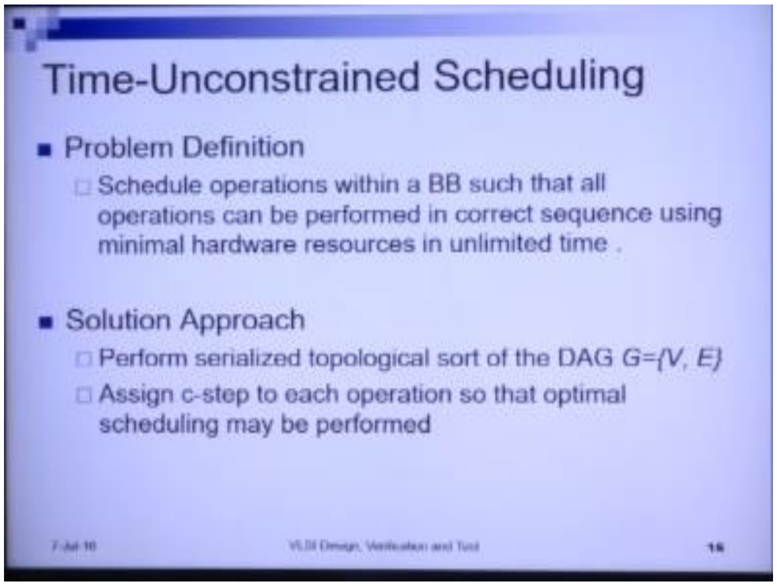
- $\{v_8, v_9, v_{10}, v_{11}\}$.

mobility two:

So taking an example, in the same operation constraints graph that we have, we see that those in the red lines, the red line in the right side here, this one represents the critical path because all nodes in this path has zero mobility. Hence v_1, v_2, v_3, v_4, v_5 all have zero mobility. See now that operation six and seven has mobility one. Why? Operation six can be scheduled in either time step 1 or time step 2. Similarly, operation 7 can either be scheduled in time step 2 or time step 3 with the latency bound of 4 that we have. So the mobility of the operation six and seven becomes one.

Similarly, the four operations 8, 9, 10, 11 has a mobility of two. Why? Because it is easy to see eight can be scheduled in operation -- in time step 1, 2 or 3. Nine can correspondingly be scheduled in time steps 2, 3 or 4. Ten can be scheduled in time steps either in 1, 2 or 3 and 11, similarly, can be scheduled in time steps 2, 3 or 4. Okay.

(Refer Slide Time: 21:34)



Time-Unconstrained Scheduling

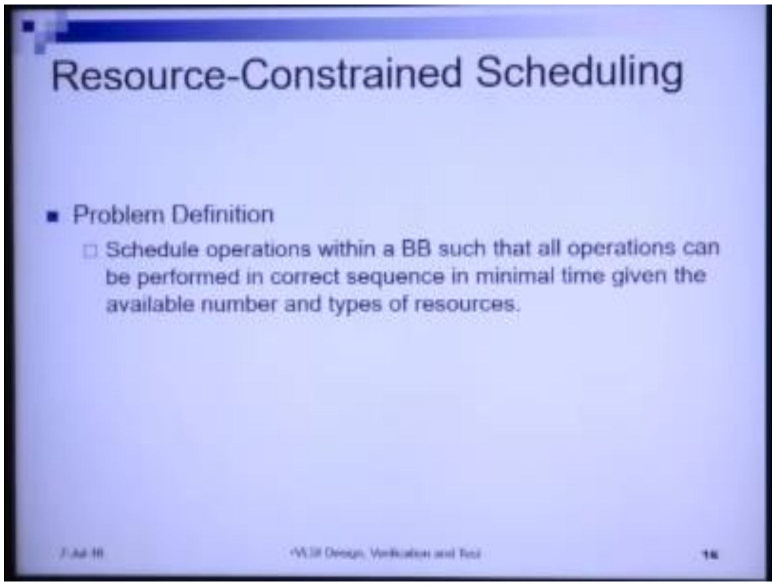
- Problem Definition
  - Schedule operations within a BB such that all operations can be performed in correct sequence using minimal hardware resources in unlimited time .
- Solution Approach
  - Perform serialized topological sort of the DAG *G={V, E}*
  - Assign c-step to each operation so that optimal scheduling may be performed

7-Jul-10 VLSI Design, Verification and Test 15

So with this concept, we now move into Time-Unconstrained Scheduling. From resource-unconstrained scheduling, now you move to time-unconstrained scheduling. So what is the problem definition of this scheduling? Schedule operations within a basic block such that all operations can be performed in correct sequence using minimal hardware resources in unlimited time.

So as we said, we now have unlimited time that is no bound on performance. Performance can be as bad as possible. However, we need to consume as low amount of resource as is possible. So how do you do that? Perform a serialized topological sorting of the DAG and assign a c-step to each operation so that optimal scheduling may be performed, right? So we will not look into this scheduling as this we -- it is easy to understand what we do here. We just perform a serialized topological sort and schedule the DAG and therefore we will not look at this anymore.

(Refer Slide Time: 22:44)



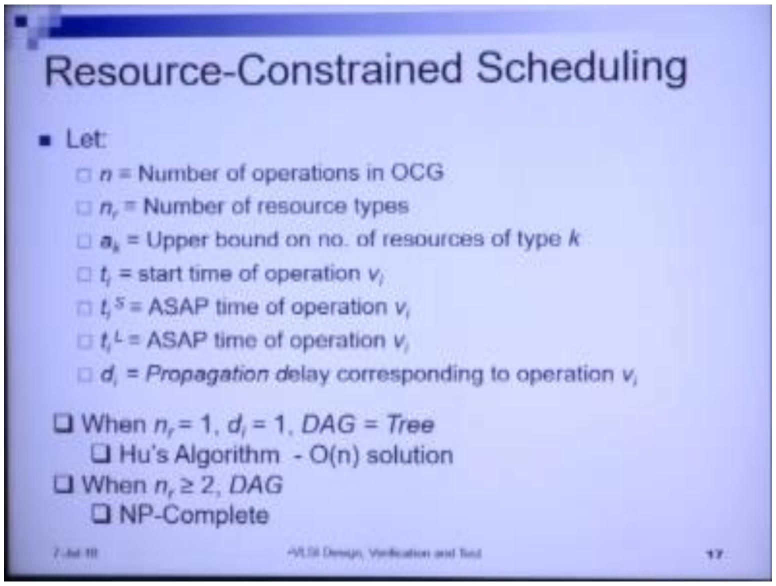
Resource-Constrained Scheduling

- Problem Definition
  - Schedule operations within a BB such that all operations can be performed in correct sequence in minimal time given the available number and types of resources.

Now we move into the more generalized scheduling strategies. So the first one here is Resource-Constrained Scheduling. So what is the definition of the problem? Schedule all operations within a basic block such that they can be performed in correct sequence in minimal time given the available numbers and types of resources. So now we have a constraint on the total amount of resources that we can use.

Now given this constraint on the availability of resources, we have to perform all the operations in the basic block in minimum time, right? So we want to maximize performance given an area constraint or a resource constraint on the circuit.

(Refer Slide Time: 23:39)



Resource-Constrained Scheduling

- Let:
  - n = Number of operations in OCG
  - n_r = Number of resource types
  - a_k = Upper bound on no. of resources of type k
  - t_i = start time of operation v_i
  - t_i^S = ASAP time of operation v_i
  - t_i^L = ASAP time of operation v_i
  - d_i = *Propagation* delay corresponding to operation v_i
- When $n_r = 1$, $d_i = 1$, *DAG = Tree*
  - Hu's Algorithm - O(n) solution
- When $n_r \geq 2$, *DAG*
  - NP-Complete

7-Jul-10 VLSI Design, Verification and Test 17

So moving on, in this type of scheduling, we will be given this operation constraints graph and we will have different symbols to represent a few things. Firstly, n represents the number of operations in the operation constraints graph, the total number of operations that are there. n_r represents the number of resource types available. a_k equals to the upper bound on the number of resources of type k.

So suppose you have both multiplication and addition operations within an arbitrary basic block, right. Now additions must be performed using a separate adder functional unit or resource and multiplications will be performed by a multiplier resource. Now let us say we have three multiplier resources and four adders. So a_k where a multiplier will be 4 because we have four multipliers and let us say three adders, so a adder will be 3 because we have three adders and n_r will be 2 because we have two types of resources. One is adder and one is multiplier.

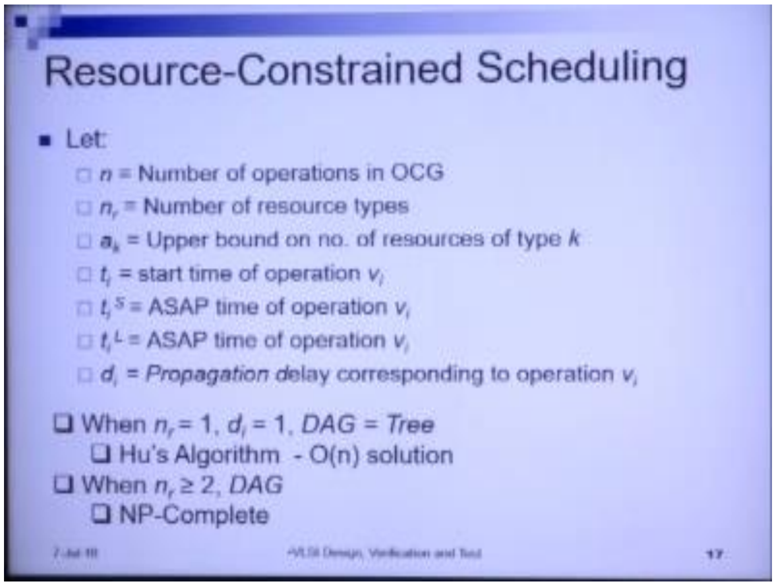
t_i will represent the start time of each operation v_i. So the operation constraints graph is represented by G = (V, E). The nodes being defined as v_i and the start time of a node will be defined as t_i. t_i^S is the ASAP time of v_i. So when we have done the ASAP schedule, we have labeled each operation with its ASAP time, its ASAP schedule time. So that ASAP schedule time will be designated as t_i^S.

Similarly, t_i^L will designate the ALAP time of operation. So there is a mistake here. It is represented as ASAP, but it will be ALAP, ALAP time of operation v_i. And d_i denotes the propagation delay corresponding to operation v_i. So the propagation delay will mean how much time it takes to perform let us say an addition on the adder circuit, on the adder resource that we have. That is the propagation delay d_i.

Now given these notations, we will look at how difficult this problem of scheduling is. For example, we actually have polynomial time optimal solutions in a very stringent few cases. For example, only when the number of types of resources are just one, we have all addition operations within the whole basic block, within the full DAG. All operations take the same delay or same propagation delay.

All operations have the same propagation delay, unit propagation delay, and the DAG is basically a tree and is not a DAG as such. For example, how does -- when does a DAG become a tree? When removing the direction constraint of the DAG, even after removing the direction constraint of the DAG, there is no cycle in the DAG, then the DAG becomes a tree. So it has been shown that if we have these constraints, then a polynomial time solution or a linear time solution can be obtained for such an operation constraints graph and that is basically obtained through the Hu's algorithm.

(Refer Slide Time: 27:50)



Resource-Constrained Scheduling

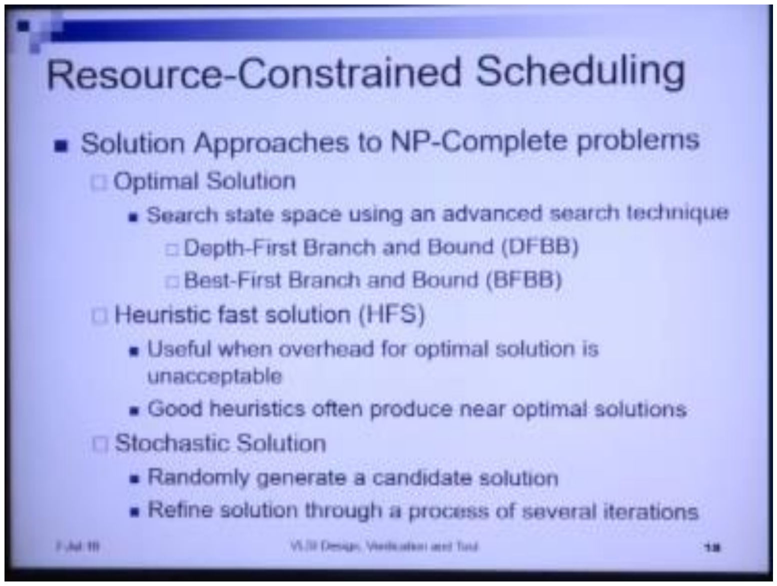
- Let:
  - n = Number of operations in OCG
  - n_r = Number of resource types
  - a_k = Upper bound on no. of resources of type k
  - t_i = start time of operation v_i
  - t_i^S = ASAP time of operation v_i
  - t_i^L = ASAP time of operation v_i
  - d_i = *Propagation* delay corresponding to operation v_i
- When $n_r = 1$, $d_i = 1$, *DAG = Tree*
  - Hu's Algorithm - O(n) solution
- When $n_r \geq 2$, *DAG*
  - NP-Complete

7-Jul-10 VLSI Design, Verification and Test 17

However, if the DAG is generalized, I have not written down here, if the DAG is generalized, then we can have at most two resources. We can have at most two resources in a generalized DAG of one type, and then only we can obtain polynomial time solutions. More than two resources, three, four, five, but a constraint number of resources, we don't know, we still don't know whether we get a polynomial time solution or not.

So, basically, what we take in general is that the solution is NP-Complete if the number of resources is greater than 2 and we have a generalized DAG. So the problem is as complicated as this.

(Refer Slide Time: 28:36)



Resource-Constrained Scheduling

- Solution Approaches to NP-Complete problems
  - Optimal Solution
    - Search state space using an advanced search technique
      - Depth-First Branch and Bound (DFBB)
      - Best-First Branch and Bound (BFBB)
  - Heuristic fast solution (HFS)
    - Useful when overhead for optimal solution is unacceptable
    - Good heuristics often produce near optimal solutions
  - Stochastic Solution
    - Randomly generate a candidate solution
    - Refine solution through a process of several iterations

7-Jul-10 VLSI Design, Verification and Test 18

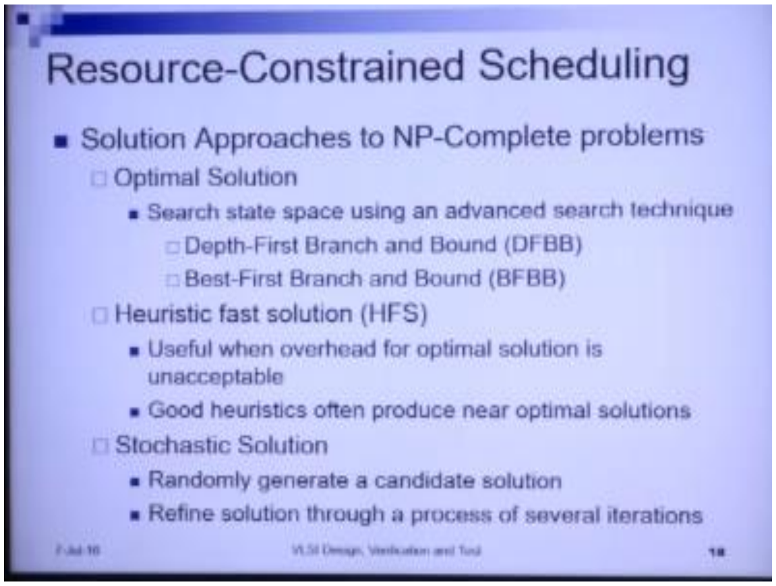
So hence, if the problem is complicated and which means that the problem is NP-complete, then we have different types of strategies to solve it. The optimal solutions can be obtained by various mechanisms. For example, we can formulate the whole problem as a linear program, as an ILP and then we have solving tools, ILP solving tools to solve such ILPs and get the solution.

We can apply advanced state-space search techniques such as depth-first branch and bound and breadth-first branch and bound to exhaustively but systematically search the whole state space and find out an optimal solution where the state space would mean all the different possibilities for assigning operations at different time steps to different resources and by exhaustively searching through all the possibilities find the best solution that we get.

However, why we said that even if it is -- even though it is exhaustive, by systematically searching the whole state space, we can prune or cut off some parts of the state space and get away by not searching them because by through search systematic technique, we can ensure that that portion of the state space will never contain the optimal solution. We will look at these optimal solutions in a bit more detail in the next lecture.

Now along with optimal solutions, optimal solutions for the huge circuits that we often have in VLSI domain is not always possible. It is not always possible to apply optimal algorithms because it may take years to give a solution. It may take even years, literally.

(Refer Slide Time: 30:53)



Resource-Constrained Scheduling

- Solution Approaches to NP-Complete problems
  - Optimal Solution
    - Search state space using an advanced search technique
      - Depth-First Branch and Bound (DFBB)
      - Best-First Branch and Bound (BFBB)
  - Heuristic fast solution (HFS)
    - Useful when overhead for optimal solution is unacceptable
    - Good heuristics often produce near optimal solutions
  - Stochastic Solution
    - Randomly generate a candidate solution
    - Refine solution through a process of several iterations

7-Jul-10 VLSI Design, Verification and Test 18

So we also have heuristic fast solutions where the entire state space is obviously not searched and hence the solutions are on most cases are sub-optimal. However, the solutions often produced are quite good with respect to the best solution that we can get and the advantage is that we often get the solution in very quick time. So heuristic fast solutions are useful when the overhead for optimal solution is unacceptable and good heuristics often produce near optimal solutions in many cases.

The other approach is the stochastic solution approach. The stochastic solution approach, in this approach we start with an initial solution. So a very crude heuristic fast solution could provide us this first randomly generated candidate solution, and from this solution we refine this solution through several iterations and moving through different areas of the state space and obtaining a solution. So it will obviously again not always give an optimal solution, but in many cases, the solutions are often acceptable and good with this, we come to the end of module one of lecture three.