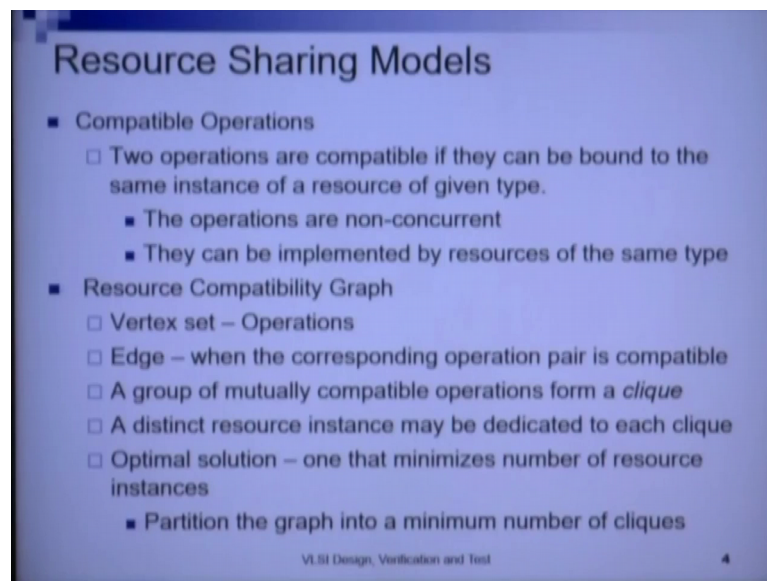


**Embedded Systems – Design Verification and Test**  
**Dr. Santosh Biswas**  
**Prof. Jatindra Kumar Deka**  
**Dr. Arnab Sarkar**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Guwahati**

**Lecture – 08**  
**Hardware Architectural Synthesis - 3**

Welcome to module 2 of lecture 6. In the last module, we had started with allocation and binding an important post scheduling step in high level synthesis. In this lecture we will proceed with allocation and binding further. Before taking a look at allocation and binding algorithms, we will first take a look at different resource sharing models.

(Refer Slide Time: 00:55)



**Resource Sharing Models**

- Compatible Operations
  - Two operations are compatible if they can be bound to the same instance of a resource of given type.
    - The operations are non-concurrent
    - They can be implemented by resources of the same type
- Resource Compatibility Graph
  - Vertex set – Operations
  - Edge – when the corresponding operation pair is compatible
  - A group of mutually compatible operations form a *clique*
  - A distinct resource instance may be dedicated to each clique
  - Optimal solution – one that minimizes number of resource instances
    - Partition the graph into a minimum number of cliques

VLSI Design, Verification and Test 4

We will look at a few definitions. First, we will understand: what are compatible operations. 2 operations are compatible if they can be bound to the same instance of a resource of given type; that means, when does 2 operations become compatible, when the same functional unit can be used to execute both these operations. When can the same functional unit instance be used? When the 2 operations are non-concurrent, if they are being execute if they are executing at two different time steps, only then can I use the same functional unit to execute both these operations on the same functional unit.

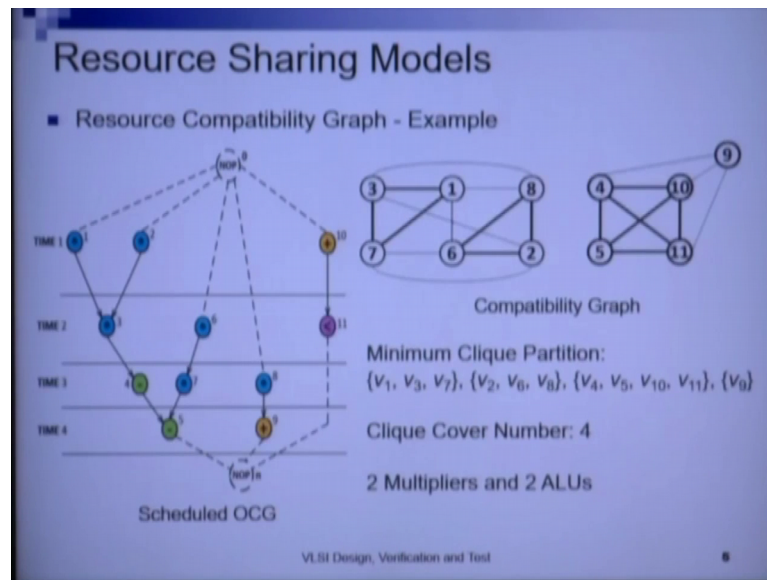
And, obviously, they have to be implementable by resources of the same type. Now with the definition of compatibility we will take a look at 2 types of resource sharing graphs. So, first is the resource compatibility graph. The other will be resource conflict graph, these 2 resource sharing graphs we will understand. So, what is a resource compatibility graph?

In the resource compatibility graph the vertex set are operations there is an edge between 2 operations if these 2 operations are compatible; which means that there will be an edge between these 2 operations, if both these operations are not scheduled at the same time step and both these operations can be executed by an operator of the same time. A group of mutually compatible operations then form a clique here what is a clique a clique is a maximal complete sub graph, alright. So, a mutual a set of mutually compatible operations form a clique or a complete sub graph, why? Because, all these operations will be connected by edges between each other and why are that? So, because all these operations are mutually compatible.

Neither of these operations are scheduled at the same time step, and the same functional unit type can be used to schedule all the operations in this clique. A distinct resource instance can be dedicated to each click. An optimal solution to the resource to the resource sharing problem is what. One that maximizes resource sharing; so, the optimal solution you will be one that minimizes the number of resource instances. So, how do we minimize the number of resource instances? If we can minimize the total number of cliques, for each click I must dedicate a single resource instance.

And if we can minimize the total number of clicks I will be able to minimize the total number of resource instances required. So, partition the graph into a minimum number of cliques. And obtaining this minimum number of cliques is to obtain the clique cover number. So, the clique cover number of a graph is this minimum number of cliques into which the graph can be partitioned into.

(Refer Slide Time: 04:12)



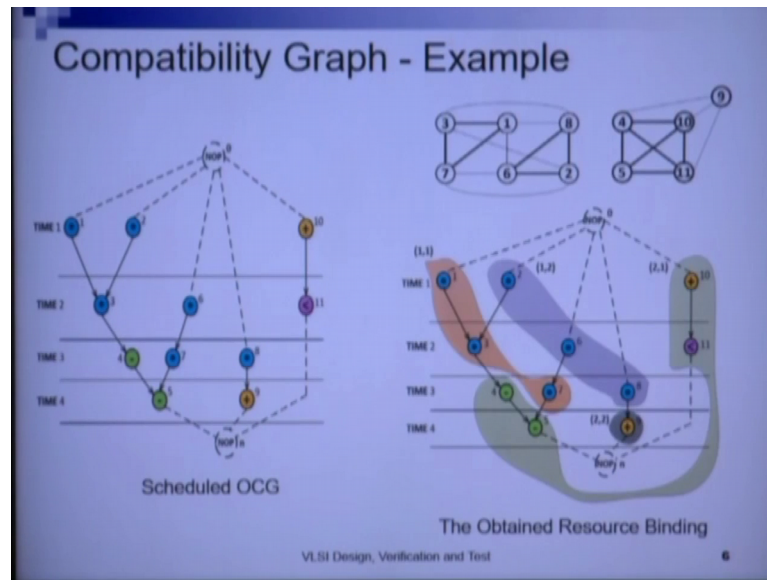
Now, in this is a scheduled operation constraints graph. On the left and on the right we have the corresponding compatibility graph. In this graph we see that operation one and operation 3 share an edge, why? Because operation one and operation 3 can be implemented by the same resource type multiplier, and they are scheduled in different time steps one and 2 in this case.

Similarly, 1 and 7 are also compatible, why? Because 1 and 7 are 2 operations that can be implemented by the same type of function unit multiplier again, and they have been scheduled in different time steps here time step one and time step 3. Likewise, all operations which share an edge are compatible. One important thing to note here is that the compatibility graph will have as many disconnected components as there are resource types.

So, here we have 2 resource types multiplier and ALU. So, for ALU I have a single a connected component, and another connected component for the multiplier operation type. Now, what are the max minimum number of clicks? What is the minimum click partition in this graph? We have here  $v_1, v_3, v_7, v_1, v_3, v_7$ ; so, the one with emboldened edges. So, these complete subgraphs with emboldened edges form the clique. So,  $v_1, v_3$  and  $v_7$  form a clique  $v_6, v_8$ , plus  $v_2$  form another clique  $v_4, v_5, v_{11}$  and  $v_{10}$  form another clique and  $v_9$  forms another separate clique.

So, these are the mean this is the minimum clique cover that we have; so, the minimum clique cover has 4 cliques and in indeed we require 2 multipliers and to ALU's for the scheduled operation constraints graph.

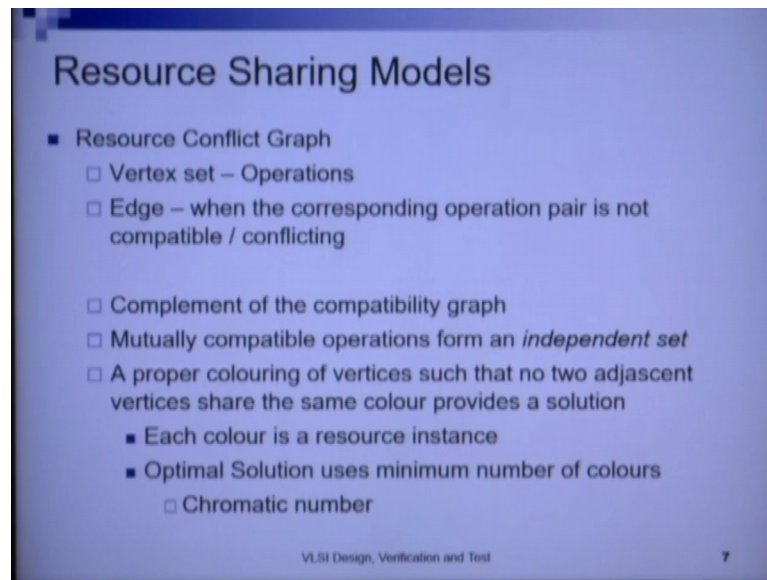
(Refer Slide Time: 06:17)



Now, as an example of the actual resource binding that we can obtain here is this. So, we have a one multiplier say let us say multiplier 1, multiplied instance one is used to schedule operation 1, 3, 7 which is in the same clique. Again another multipliers say multiplier 2 is used to execute operations 2 6 and 8 in time steps 1 2 and 3.

Similarly, eh ALU one is used to execute the operations 10, 11, 4 and 5 in 4 different time steps. And the operation v 9 is given another dedicated ALU for it is execution. So, this is an example allocation, and this is the best possible allocation as it turns out.

(Refer Slide Time: 07:17)



**Resource Sharing Models**

- Resource Conflict Graph
  - Vertex set – Operations
  - Edge – when the corresponding operation pair is not compatible / conflicting
  
- Complement of the compatibility graph
- Mutually compatible operations form an *independent set*
- A proper colouring of vertices such that no two adjacent vertices share the same colour provides a solution
  - Each colour is a resource instance
  - Optimal Solution uses minimum number of colours
  - Chromatic number

VLSI Design, Verification and Test 7

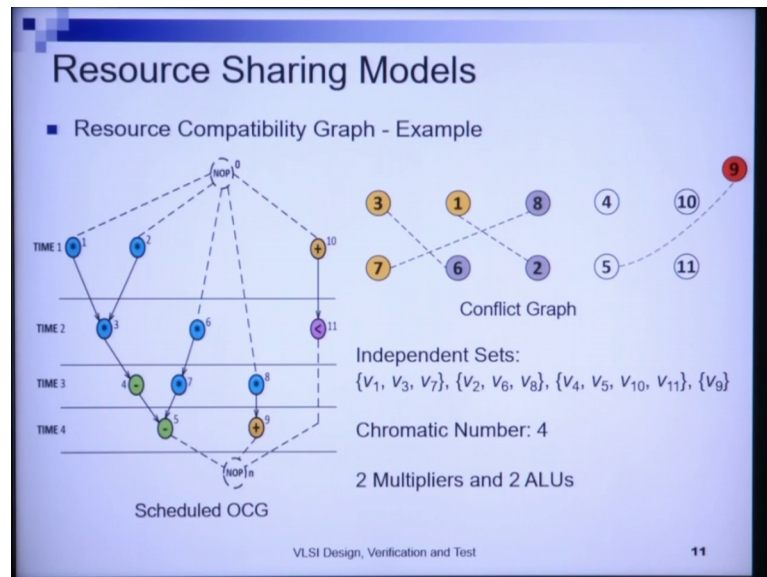
The second resource sharing model is the resource conflict graph. Resource conflict graph has the same vertex that as the compatibility graph. So, it is this it is vertex set are also the operations of the operation constraints graph. What are the edges? The there is an edge between 2 operation between 2 operations if they are not compatible; that means, in the compatibility graph 2 operations shared an edge if they were compatible, in the conflict graph 2 operations share an edge if they are not compatible.

Hence, the operator the conflict graph is a complement of the compatibility graph. So, when will 2 operations be conflicting? When these 2 operations are scheduled either in the same time step or these 2 operations are unrelated. In meaning that, they cannot be executed using the same functional unit type; however, it is true that if you have a set of compatible operations they will form an independent set. What is an independent set? The vertices in an independent set will be those which do not share an edge between them.

So, mutually compatible operations will form an independent set. A proper coloring of vertices such that no 2 adjacent vertices share the same color provides a solution. So, what do the vertex coloring do? Vertex coloring tries to find the minimum number of colors that are required to color the vertices of a graph, such that you cannot assign the same color to 2 vertices that share an edge.

These colors actually they denote resource instances. Each color is a resource instance, and optimal value uses the minimum number of colors. Such minimum number of colors is called the chromatic number of the graph.

(Refer Slide Time: 09:28)



Example of a conflict graph; so, in this we see that 3 and 6 share an edge. 3 and 6 share an edge, why? Because 3 and 6 are operations of the same type and being scheduled in the same time step; because they are scheduled in the same time step they are conflicting, and cannot be executed by the same resource instance and therefore, they share an edge; obviously, 3 and 4 also share an implicit edge which is not shown. Because 3 and 4 are 2 operations unrelated operations which must be executed by operators of different type, there is an implicit edge between them which we have not shown here.

Now, what are the independent sets in this conflict graph 3 7 and 1 is an independent set they do not share an edge between them. 2 6 and 8 is another independent set. They do not share an edge between them. 4, 10, 5 and 11 is another independent set and v 9 forms another independent set. It is important to note here that the cliques mean the compatibility graph are now independent sets in the conflict graph.

The chromatic number here is same as the vertex cover number in the compatibility graph is 4, and therefore, we again require 2 multipliers and to ALU's allocation and binding will again be same. Now, one of the important outcomes of this analysis steel study of these models is that, given general compatibility and conflict graphs, both the

minimum clique partitioning problem and the minimum graph coloring problem are np complete.

And hence exhaustive or and enumerative type solutions only do exist. For the general versions of the compatibility and conflict graphs; now with this understanding of resource models, we will first we will now see how from the scheduled operation constraints graph can be obtained a compatibility graph.

(Refer Slide Time: 11:57)

**Resource Sharing Models**

- Given a scheduled OCG
- Number of operations:  $n$
- Resource type of an operation is denoted by:  $\tau(v_i)$
- Start time of an operation:  $t_i$
- Execution delay of an operation:  $d_i$
- So, a compatibility graph will have the edges:
  - $E = \{v_i, v_j\}$ , such that
    - $\tau(v_i) = \tau(v_j)$  and  $t_i + d_i \leq t_j$  **or**  $t_j + d_j \leq t_i$
    - Can be constructed in  $O(n^2)$  time by traversing OCG

VLSI Design, Verification and Test 12

Similarly, we using a similar procedure, we can easily obtain the conflict graph as well because, as we have seen that the conflict graph is just a complement of the compatibility graph. Now given a scheduled OCG let  $n$  be the number of operations. Resource type of an operation is denoted by  $\tau(v_i)$  this comes from the scheduling problem.

So,  $\tau(v_i)$  tells me the type of resource that I required to schedule operation  $v_i$ . The start time of the operation is  $t_i$  because this is a shed you will do operation constraints graph I know the distinct start time for each operation  $i$ . The execution delay of operation  $v_i$  is also known this is  $d_i$ . So, the compatibility graph will have edges  $e$  equals to  $v_i v_j$ , such that  $t_{v_i}$  equals to  $t_{v_j}$  the first here first term tells me that both  $t_{v_i}$  and  $\tau(v_i)$  and  $\tau(v_j)$  must be same; that means, they must  $v_i$  and  $v_j$  must be executable by the same type of resource. The second constraint tells me that if  $t_j$  is scheduled later than  $t_i$  than  $t_i$  then  $t_j$  must be scheduled at least  $d_i$  time steps later than  $t_i$  starts. So,  $t_j$ 's start time must be at least  $d_i$  time steps after  $t_i$ . Why this is so?

So, because  $d_i$  is the delay of operation  $i$  operation  $v_j$  follows  $v_i$ . And hence otherwise there is an overlap in the execution intervals of these 2 operations and therefore, the same operation instance cannot be used to execute both these operations  $v_i$  and  $v_j$ . Similarly, if  $v_i$  starts after  $v_j$  then  $v_i$  must start at least  $d_j$  time steps after  $t_j$  after  $v_j$  starts. So,  $t_j$  plus  $d_j$  must be less than or equal to  $t_i$ , only then can I ensure that both operations  $v_i$  and  $v_j$  will not overlap in that execution intervals.

And hence both these operations  $v_i$  and  $v_j$  can be implemented by the same type of resource. Now we understand that how can we obtain these edges I need to traverse the operation constraints graph. And then I need to consider each operation in the scheduled operation constraint graph. Then I find out which opera for a given operation in the operation constraints graph, which are the operations that are adjacent to it in the operation constraints graph. For all those operations if this constraint is true, I can put an edge between these 2 operations in the compatibility graph.

Now through such a traversal of the operation constraint graph we understand that all edges of the compatibility graph can be obtained in big O of  $n^2$  time, where  $n$  is the number of operations in the operation constraints graph.

(Refer Slide Time: 15:40)

**Resource Sharing Models**

- Compatibility graph becomes *chordal* or *triangulated* when,
  - Every cycle with more than three edges has a *chord* or edge joining two non-consecutive edges in the cycle
- Interval graph – A subclass of chordal graphs
  - Vertices can be put in one-to-one correspondence with a set of intervals
  - Vertices are adjacent if corresponding intervals overlap
- A graph becomes a comparability graph when
  - It satisfies transitive orientation property
    - Assign orientation w.r.t order of start times
    - In the resulting directed graph  $G(V, F)$ 
      - $\{(v_i, v_j) \in F \text{ and } (v_j, v_k) \in F\} \Rightarrow (v_i, v_k) \in F$
- *Polynomial time clique partitioning and colouring for these graphs*

VLSI Design, Verification and Test

13

Now, as we said for general compatibility and operation concern for general compatibility and conflict graphs, both the minimum clique partitioning problem and the graph coloring problem is np complete. And hence must be solved by enumerative

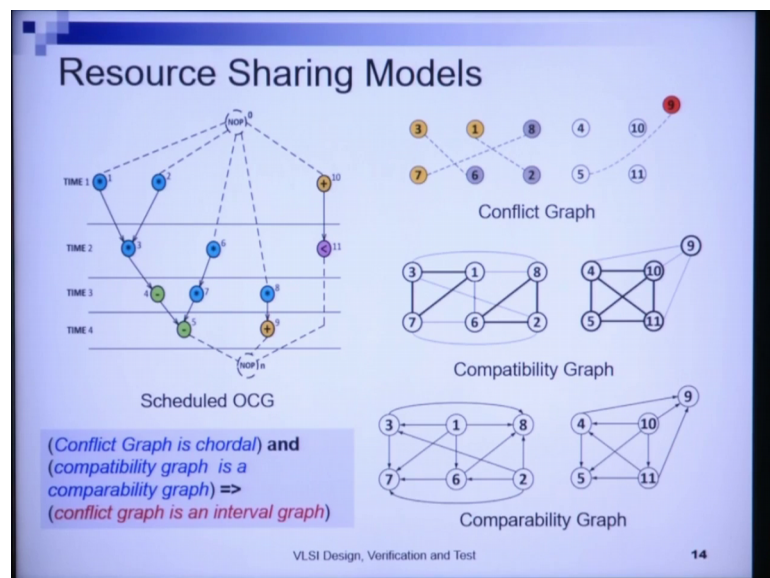


methodologies; however, there are special classes of graphs for which both the minimum clique partitioning problem and the graph coloring problem can be solved optimally in polynomial time.

So, if my compatibility graph or my conflict graph falls into one of these categories of these special types of graphs, then my graph coloring problem or my clique partitioning problem can be solved optimally in polynomial time. Hence, we need to understand these special types of graphs first; to understand whether my graph coloring problem and the clique partitioning problem can be solved in polynomial time or not.

So, first we understand what our chordal or triangulated graphs eh. If we say it in terms of a compatibility graph, a compatibility graph becomes a chordal or triangulated graph when every cycle with more than 3 edges in the graph has a chord or edge joining 2 nonconsecutive edges in the cycle. So, I am given a compatibility graph, and I want to find out is this compatibility graph a chordal graph, how do I find out? If I see that each cycle of more than 3 edges in my compatibility graph has a chord joining non-consecutive edges in the cycle, then I can say that my compatibility graph is a chordal graph.

(Refer Slide Time: 17:45)



For example, let us take this compatibility graph that I have here. In this compatibility graph let us take one cycle. 1, 3, 7, 6 in this cycle we say that there is a chord 1 7 here. This chord this is a cycle of length more than 2 this is a cycle of length 4. And we have

told that every cycle of length more than 3 will for every cycle of length more than 3. I will have a chord joining non-consecutive vertices. And hence these vertex 1 and 7 are nonconsecutive and hence there is a chord joining it in this cycle 1, 3, 7, 6

Similarly, we will see that for all other cycles of more than 3 or of lengths more than 3. Similarly, we will see that for all other cycles of length more than 3, I will have a chord joining nonconsecutive vertices in the cycle. And hence this compatibility graph that we have here is a chordal graph. Now, therefore, the minimum clique partitioning problem for this chordal graph for this compatibility graph can be solved now optimally in polynomial time.

(Refer Slide Time: 18:06)

**Resource Sharing Models**

- Compatibility graph becomes *chordal* or *triangulated* when,
  - Every cycle with more than three edges has a *chord* or edge joining two non-consecutive edges in the cycle
- Interval graph – A subclass of chordal graphs
  - Vertices can be put in one-to-one correspondence with a set of intervals
  - Vertices are adjacent if corresponding intervals overlap
- A graph becomes a comparability graph when
  - It satisfies transitive orientation property
    - Assign orientation w.r.t order of start times
    - In the resulting directed graph  $G(V, F)$ 
      - $\{(v_i, v_j) \in F \text{ and } (v_j, v_k) \in F\} \Rightarrow (v_i, v_k) \in F$
- *Polynomial time clique partitioning and colouring for these graphs*

VLSI Design, Verification and Test

13

After an understanding of chordal graphs, we will now understand what interval graph is. Simply speaking and interval graph is a graph where the vertices have one to one correspondence with the set of intervals. For example, let us consider the opera scheduled operation constraints graph here. In this schedule operation constraints graph, each vertex can be represented by a continuous single interval defining it is start of execution and end of execution.

So, each vertex or each operation in this operation constraint graph can be represented by a single continuous interval representing it is start of execution to the end of it is execution. So,  $t_i$  to  $t_i + d_i - 1$ , this interval is the in is the interval for vertex  $i$ .

For each vertex similarly I can define a continuous interval which denotes the lifetime starting from the start of its execution to the end of its execution.

And hence simply speaking all the operations in this operation constraints graph have a continuous interval. And hence the corresponding compatibility graph or the conflict graph is basically an interval graph. So, interval graphs are a subclass of chordal graphs if you have an interval graph the property for chordal graphs will always be true. So, vertices can be put in one to one correspondence with a set of intervals. Vertices are adjacent if corresponding intervals overlap. So, vertices are adjacent if corresponding intervals overlap. So, in my interval graph vertices are intervals, and there are edges between 2 vertices if they are intervals overlap.

Now, we come to the third class. So, why have we studied interval graphs? We can still have more simpler algorithms for both clique partitioning and graph coloring for interval graphs. As we will see later we will study one algorithm for interval graphs for graph coloring of interval graphs. Now we will look at the third class of graphs. A graph becomes a comparability graph when it satisfies a transitive orientation property. So, therefore, I need to be able to impose a transitive orientation property on my graph. So, my compatibility graph or my conflict graph can only become a comparability graph if I can impose this transitive orientation property on this graph.

Now, if I impose a transitive orientation on an undirected graph it becomes a directed graph. So, my compatibility graph or my conflict graph will be a comparability graph when after imposing this direction property on the compatibility graph I will have this. In the result directed graph  $G = (V, E)$   $v_i v_j \in E$  and  $v_j v_k \in E$  implies that  $v_i v_k \in E$ . So, this direction property this transitive direction property will be imposed will be impossible on this graph. As an example let us see my let us see this compatibility graph that we have here. So, in this compatibility graph we have an edge from 1 to 6, and we have another edge from 6 to 7.

So, let us see this is  $v_i$  this is  $v_j$  and this is  $v_k$ . So, if I have an orientation if I impose an orientation from  $v_1$  to  $v_6$  from  $v_i$  to  $v_j$  I have an orientation from  $v_i$  to  $v_j$ . And then I also impose an orientation between  $v_j$  and  $v_k$ , I can impose an orientation from  $v_1$  to  $v_7$ . So, if there is an orientation from  $v_j$  to  $v_i$  to  $v_j$  and there is an edge and oriented edge from  $v_i$  to  $v_j$  there is an oriented edge from  $v_j$  to  $v_k$ , there will also be an oriented edge

from  $v_i$  to  $v_k$ . This is what this ensures, and hence this compatibility graph is a comparability graph. Because for all edges I can impose an orientation property like this.

Now, on this how can we get a sense of how to obtain this orientation. We can assign an orientation with respect to the order of start times in the operation constraints graph. Now here there is we are putting an orientation from 1 to 6, because 6 starts after one in the operation constraint curve 6 starts after 1. There is an orientation from 6 to 7, because 7 starts after 6. And hence there is an edge between one to 7 denoting that 7 starts after one. Such an orientation can be imposed for all for all edges and hence this compatibility graph is a comparability graph.

But why are we studying these graphs? By studying the chordal graphs and the comparability graphs, we can find out whether a graph is with whether the graph is an interval graph or not. If my conflict graph is a chordal graph, and the compatibility graph is a comparability graph, then the conflict graph is an interval graph. So, if this conflict graph is a chordal graph. Here this conflict graph is a chordal graph because, there are no cycles of more than 3.

In fact, there are no cycles in this graph. And hence this conflict graph is a chordal graph. We have seen that this compatibility graph is a comparability graph. We have already understood this. And therefore, this conflict graph now becomes an interval graph, right. The general theorem is that a graph is an interval graph only if it is chordal. And it is complement is compatibility graph. Here we know that the compatibility graph is a complement of the conflict graph. And hence if the conflict graph is chordal the compatibility graph is a comparability graph, then the conflict graph becomes an interval graph.

We come to the end of module 2 of lecture 6.