VLSI Physical Design with Timing Analysis

Dr. Bishnu Prasad Das

Department of Electronics and Communication Engineering

Indian Institute of Technology, Roorkee

Week 01

Lecture 04

Graphs for Physical Design

Welcome to the course on VLSI Physical Design with Timing Analysis. In this lecture, we will discuss about the graphs that is used in VLSI physical design. So basically the graph is the one of the fundamental data structures which is used in VLSI physical design algorithms. So, there are several algorithms there which is used in case of VLSI physical design which use the graph as the data structure in which what we are doing is that we are representing the our electronic circuits and we are analyzing the circuits and we are optimizing the circuits using the graph. So, the graph is significantly important data structure for the optimization analysis of the electronic circuit. So let us discuss into some of the applications of graph algorithms. So, it is used for the representation of the netlist. So, you have a electronic circuits they are reprinted in terms of nets and we can reprint that netlist in terms of a graph and it is also useful for doing the placement of the different blocks in case of floor planning phase of the VLSI physical design. So, the graphs several graphs algorithms are useful for routing purpose. Routing is one of the important phase of the VLSI physical design in which we use the graph for optimizing the wire length and similarly the graph based algorithms are also used for critical path problems.

The graph-based algorithms are useful for identifying the longest path in the circuit in the critical path of the circuit and here we will describe some of them. The graph here if you can see in the left-hand side, I have a circuit given to us how we can represent this circuit in terms of a graph. So, all the primary inputs these are the primary inputs, and these are the primary outputs. So basically, all the primary inputs and primary outputs are reprinted by a vertex. So, I have four primary inputs four vertex and there are edges between A and B.

So, A, B, C, D and this is your G1. Similarly, here you have G2 now we have another node this is G3. Now I have another node Y which is the primary output. So now if you look into this one, we have represent this circuit in terms of a graph. So, this A, B, C, D, G1, G2, G3 and Y are the nodes and the connection between them are called edges.

So now we will look into the graph actually. The graph is basically consists of two things one is set of vertices and set of edges. So, you have two things one is set of vertices V and set of edges E. So, if you look into this graph I have vertices A, B, C, D, E and F and G. It is a set of nodes or the vertices. So, this is the elements of the set V. Similarly, the edge is basically let us say this is one of the edge this is another edge. So I can denote the mark the edges by let us say E1 I will mark the edges by E1, E2, E3 like this. So, you can also write this A, B as E1 that is also possible. So now I will go to another graph which is mostly used in case of VLSI physical region that is called hypergraph. Hypergraph is basically a different type of graph where it consists of nodes and hyperedges.

hyperedges is basically the subset of two and more nodes. So, it has if you can see here this edge is connected to two and node one and two as well as to the three. So, this is called a hyper edge. Now what is the degree of a node actually the node or vertices? How many basically edges are incident on that one? So here if you can see the node A or the vertex A has two edges. So, degree is 2. And the node E has degree of node E is basically 3 degree of node E is 3. So, then we can define a path. A path is between two nodes is an ordered sequence of edges from a start node to the end node. So, we will start from a node and we will go through a path and we will reach a end node that is called a path. And what is the loop is basically we are creating a cycle there.

So, we are starting from a start node ends with the same node. So here we have a cycle, or a loop is a closed path where the start and end points are the same. So, this is called a loop. Now we will look into two more important graphs, one is called undirected graph and directed graph. So, in case of undirected graph there is no direction is used no edges has no direction but in case of a directed graph the edges have direction from one side to the other.

So, this is called ordered basically the direction is called ordered specific ordered but in case of undirected graph this is called unordered nodes actually. So, look this is an example of a undirected graph because there is no directions are given in this graph. But if you look into this here, I have a direction from node A to node B. I have a direction from node B to node C. So, this is called directed graph.

So, there is another graph called cyclic graph. What is this cyclic graph? Basically at least one directed cycle is there. So, there is a loop is there. A directed graph has at least one directed cycle that is called cyclic graph. If there is no cycle that is called acyclic graph. So, in case of EDA algorithms mostly we deal with acyclic graphs that is why it is called directed acyclic graph or DAG. Those are used in static timing analysis to do timing analysis of circuits. So, we use directed acyclic graph which we will discuss in while we are going to discuss the timing analysis. So, this is a cyclic graph because if you can see here we have a cycle here but this is a not a cyclic graph because this is going in only in the forward path. There is no either this path or this path.

So, there is no cycle here. So, this is a acyclic graph. Now we have a complete graph. What is this complete graph? Basically if you have all the nodes are connected with each other each and every node will be connected with each other that is called a complete graph. A graph in which an edge connects each node to every other node.

Let us say if you can see to the node A or vertex A it is connected to B, it is connected to C, it is connected to D. Similarly for other nodes like B, C and D. So, number of edges in case of a complete graph is nC2 that is called n into n minus 1 by 2. Similarly, this is the number of edges in case of a complete graph.

Now we have connected graph. Connected graph means there is a at least one path between each pair of node. It is a one at least one path should be there. So let us take an example here. This is there is no path between B and E. So, this is not a this is a unconnected graph.

So, this is a this graph is a unconnected graph. However, the left-hand side graph this one is a connected graph. So, this one is a connected graph because this every node is have at least one path to other nodes in the graph. So now we will go into the tree. Tree is basically is a graph with n nodes and they have connected by n minus 1 edges.

So, it is a graph with n nodes, but they are connected by n minus 1 edges. So tree has no cycle. So, it has no cycle. We do not have any cycle in a tree. And tree is basically also known as maximally acyclic graph. Why it is called maximally acyclic graph? If I add one edge to that one, then it will create a cycle. So, each pair of node is connected by exactly one edge. So here in this case two nodes will be connected exactly by one edge. So, there are two types of trees are there. One is the undirected where there is no direction is there and directed when there is a direct direction is there from each node to the other node.

So, there are two types of tree. One is called undirected tree, and the second one is called a directed tree. So, this is a graph. The left-hand side is basically a graph and then the right-hand side this one is a tree. Why it is a tree? Basically, if you can look into it there is no cycle here. If I connect these two it will create a cycle.

So that is why it is called maximally acyclic graph. So, it is also called if you can look into maximally acyclic graph. That means that if I add one edge to that tree then it will create a cycle. But in case of a tree, I do not need a cycle. So it is a maximally acyclic graph. So now we will look into spanning tree. It is a connected acyclic subgraph G bar or G prime contained within G V, E that includes or spans every node V belongs to V. So it is basically a acyclic subgraph. I have a graph given G V; E is given to me. This is given to me. I need to find a subgraph G bar or G prime, and it should connect each node with the every each node there is should be edge between every node. That is called a spanning tree. Spanning tree is a connected acyclic graph. So, it is a subgraph of a graph G V, E such that it spans every node V belongs to V. And what is the minimum spanning tree? So minimum spanning tree is that if you take the sum of all the edges that is minimum.

If I take the sum of all edges total edge cost is minimum. So now we will look into how we can represent the graph in terms of some form of data structures such that we can process it, do different operation on that one. So, there are two standard method which is used to represent the graph. One is called adjacency list, then one is called the adjacency matrix. So, adjacency list basically for sparse graph basically if you for which E node is much less than V square the adjacency list representation is preferred.

So, whenever you have a sparse graph is there then we have less number of edges. In that case the adjacency list is a efficient method of representing the graph. But for a dense graph where E mode is close to V square, E mode is close to V mode square in this case the adjacency matrix representation is preferred. So, in case of E mode is much much less than V square adjacency list representation is better in terms of memory. So, we will go one on example of that one. So, what is adjacency list representation? I have a graph G V, E which is consist of a adjacent of V list one for each vertex in V. So for basically for U belongs to V we need to create a adjacency list for each node which is connected to that vertex. So, we have a list of nodes or the vertices then we have list corresponds to each vertices. So, for each vertices those corresponding edges will be added in the list.

Let us take an example here. So here if you can see we have how many nodes are there? I have 6 nodes are there or 6 vertices, vertices and nodes are same thing. I am using it alternatively. So here if you can see I have 6 nodes are there. So, all the 6 is creating a list here then for each vertex let us say 1. So, 1 is connected to 2 and 6. So I have to write 2 then 6 then this is a null pointer. Similarly, I have to do it for all. So, this is a example. So, 2, 6 similarly for node 2, node 2 is connected to 3 and 6. So similarly you can 2 is connected to 3, 1, 6. So it 3, 1, 6 should be connected to the node 2 here. Similarly, 3 is connected to 2, 6, 5 and 4. So this is a list for the node 3. So, by that method and you remember that last one is a null pointer to denote that there is no further edges which can be there in the list. So, these are the adjacency list representation of a undirected graph. This is for undirected graph. So, this is an example of a adjacency list representation of a directed graph. So, we have a directed graph. So, in this case we need to consider the direction while creating the list. So let us say I have a node 1. Node 1 is connected to 2, 5 and 4. So node 1 will connected to 2, then 2, then 4, then 5. So, 2, 4, 5 are the node connected to node 1. So, this is the way we can represent this one. But there is no edge from 2 to 1. So, if you can go to 2, 2 to 1 edge should not be there. So 2 to 3 should be there and 4 should be there. But 2 to 1 is not there because it is a directed graph. Since it is a directed graph, we need to consider the direction at which it is connected. So, this is the thing, this is the final representation of the complete directed graph. So, this is one of the efficient method when there is a your graph is sparse, less number of edges are there. So, this is a best method of representation when then your graph is not sparse basically. Basically, less number of edges are there. So, the adjacency list-based representation is useful.

Then we will go to weighted graph actually. The graph V, G as a weighted graph with weight function W. So, we have one weight is assigned to each of the edges. So, we will replace that thing with that edge weight. So, this is a adjacency list representation of a weighted directed graph. So, it is a weighted directed graph. So, what is happening here? I have a node 1 which is connected to 2, which is connected to 4 and which is connected to 5. So, if you can see is that node 1 is connected to 2 and connected to 4 and connected to 5. Then you have a null pointer to say that there is no more connections are there. Then you have a edge associated with each one of the node. So that is what is associated with that one that will be stored inside that node itself. So let us say I have a two nodes are connected with each other and let us say I have some delay parameter there in my circuits. So that delay information can be stored as a parameter in case of a node. So, this weighted directed graph is useful for your static timing analysis. So now you have go into the adjacency matrix representation of the graph. So, it is a matrix like format. You have basically the size of the matrix of that graph. Let us say the graph is V. Graph is represented by V, E. So, the size of the matrix is basically V mod cross V mod. We have the size of the matrix is basically V mod by V mod. And what is the element of the matrix? The element of the matrix aij is basically 1 if there is a edge between i to j.

If i to j has a edge then this will be 1 otherwise it is 0. So otherwise, it is 0. So let us take an example here. So basically, I have how many vertex here? Vertex is number of vertex is 6 here. So, the size of my adjacency matrix is 6 by 6. Size of my adjacency matrix is 6 by 6. And if I have a edge from 1 to 2, let us say I have a edge from 1 to 2. So, this is 1 and this is 2. I have a 1 here. So, I have a 1 here. So, I have a edge from 1 to 6. So, 1 to 6 there will be 1 here. So, 2, so if I go to the vertex 2 what are the connections are there? It is connected to 1. It is connected to 1. It is connected to 6. It is connected to 3. So, I have 1. Then I have 3. I have 6. Similarly, you can write it for other vertices.

So, this will create your adjacency matrix of a undirected graph. Similarly adjacency matrix for a directed graph, so we need to look into the direction of the edges while creating the matrix. And here your number of basically number of nodes is 5. So, the size of the matrix is 5 by 5 and let us say 1 is connected to 2, 4 and 5. One is connected to 2, 4 and 5. So one node it is connected to 2, 4 and 5. So like this. So, we can fill the rest of the elements of the matrix. So, this is for a directed graph. So, in case of adjacency matrix representation for a weighted graph, so in case of a weighted graph the weight of the u, v can be stored as the entry of the corresponding row and column of the adjacency matrix. So, this is an example of a weighted directed graph actually. We are representing this weighted directed graph using a adjacency matrix. So, we have some edge width involved with each of the edges actually. So that each let us say I have 1 to 2, edge we have 2, so then this is 2. This 2 corresponds to this 2. Basically, from edge 5 to 2 you have a weight is 8 here. This is basically reflected here. So, in that way you can fill up the rest of the things. Now we will discuss various graphs related to physical design. So basically, we

do lots of layout in physical design. So, if you can look into the layouts, they are rectangular in nature. If you see you have a, this is your active let us say. I will draw the poly; poly is let us say green color. So, this is the poly. So, it creates a transistor. So, this is a poly, this is basically poly and this is active. This is basically create a transistor. So, you see whenever we are doing layout, they have number of rectangles there. So, this is a rectangle, this is active is a rectangle, poly is also a rectangle. Even also we have a metal lines. Let us say I have another metal lines going from there. So let us say I have a metal line connected with the active. So, this is another rectangle. So, this is all, this is metal 1. So, in case of routing algorithms or routing problems we have rectangles which is represented as your routing wires and often this is called, this is very thin and long. So, the thin dimension we can reduce, and we can represent that metal. Let us say this is a metal because this dimension of width is same, we can represent this one as a line. So, we will abstract out the width of the metal. Similarly in case of a placement problem or a compaction problem let us say I have a block is there we have to consider both the dimension. So, this is basically we have two types of graphs. One is called line graphs; one is called rectangular graphs. So, we have basically two different types of graphs in case of VLSI physical design. One is graph related to set of lines, the graphs related to set of lines and the graph related to set of rectangles. So how different graphs can be there in the VLSI physical design we will discuss one at a time. So, the lines, there are two types of lines are there. One is called aligned to the axis. Let us say if you can see here this line is parallel to your x axis. So, this is called aligned to the axis. But if you see these lines these are not aligned to the axis. So now let us take a interval. Interval means I have two points LA and RA. This is a interval. So, LA is the left hand side, RA is the right hand side of the line A. So, this is a line to x axis. So, this is represented as LA, RA. So now related to the intervals I have set of intervals. We have three different graphs are defined. One is called overlap graph, one is called containment graph, one is called interval graphs. So, the graph overlap, containment and interval graphs those are used extensively in place of routing algorithms. So, we have a overlap graph which is defined by G0 and which is V, E0 where V is basically number of intervals. Here how many intervals are there? A, B, C, D, E, F. So, these are the intervals. All are basically going to the V set. And we need to define the edge set, which is coming, how the vertices will connect to each other. So, in that case we have a constraint here. This is the constraint.

If LI is less than LJ less than RI less than RJ, then it is called a overlap graph. In short basically edge is defined between VI and VJ. If interval II overlaps IJ but does not completely contained or reside inside it. So, it will not completely cover it but it will just overlap that one. For example, if you can see here A and B there is a edge between them. But B and D if you can see here A this is B so this is your LI, LJ this is RI, RJ. So how they are related? So, LI is less than LJ less than RI less than RJ. So there is a edge between A and B. But there will not be edge between B and D because D is inside B.

So, if you can go to the next slide you have a edge between A and B. So, there is edge between A and B. Similarly, you can apply other parts also. A is having a edge between A and C has a edge between them. Then C and E has a edge between them. Similarly, you can find it for others. But D is contained inside B and contained inside A so there is no edge between A to D or B to D. So, the containment graph is basically GC where the basically it is a basically defined over the intervals. So, if you can see here your LI is less than LJ, RI is greater than RJ. So, it is completely contained inside. Basically, what is the edge is defined between VI and VJ. If interval II completely contained IJ. So, if you can look into this your there is a edge between B and D. You can see here so this is an example of a containment graph. What is happening here is that D is completely covered by B and D is completely covered by A.

So, there is a edge between A and D. There is a edge between B and D. And C is not completely covered by A so that there is no edge between A and C. Okay similarly you can do it for other edges. Then you have interval graph. Interval graph is basically union of overlap graph and the containment graph. An edge is defined between VI and VJ if interval II has a non-empty intersection with IJ. So, if you have AI equals to EO union EC then EI is our interval graph. So, this is an example of interval graph and this is the set of lines okay and this is the corresponding interval graph. It is basically EO union EC. EO union EC is basically your EI. This is the condition for creating the graph. Now I have a permutation graph okay which comes out of the matching diagram. What is this matching diagram? Matching diagram is a diagram where we have lines begin at a designated Y coordinate and ends at a designated Y coordinate. This is basically used for channel routing okay. So, we will look into this and the permutation graph is basically defined as if line I intersects line J then we have a EP. So basically, EP is basically line I intersect line J okay. So this is your matching diagram. So, what is happening is that I have a block A here, block I. In case of VLSI physical intelligence there is one block is there, there is another block is there, block II. Then they are connected in a channel. So, this is creating a channel. So how this net I is connected to net I. Net II is connected to net II. Net II whatever I was talking about the nets of a circuits actually. So, II is connected to II, III is connected to III, IV is connected to IV, V is connected to V.

So, this is called matching diagram. So, we can create a permutation graph corresponding to the matching diagram. So, the corresponding permutation graph is that one is basically intersect II, III, IV, and V. One has a edge between II, III, IV, and V. Similarly, two has basically two is connected to III, connected to IV, and also V. So, II is also connected with all of them. So, then your V is connected with IV and I and II. V is connected with IV, I, and II. It is not connected between III and V. So, this is useful for channel routing algorithms. Now it is a switch box actually. So, switch box is that we have basically blocks is there this side, this side, this side, and this side. So let us say this is block I, block II, block III, this is block IV. Like you have a crossing is there in our streets, so you have

path from all the sides. So, this is called switch box in case of channel routing algorithms. This is called switch box because all the direction you have path to route. So, if you have a intersection of the lines, then there will be a line in case of the circle graph. Let us say one is touching 6, 5, 4. One is touching 6, 5, 4. One will be 6, 5, and 4. They have a edge between them. So similarly, we can create a circle graph. Now we have a graph related to rectangles. This is called neighborhood graph GV, E. We have V is basically reference rectangles Ri and E basically your if the two neighbors are touching each other. So basically, this is your EC. So, neighborhood graph is used in global routing phase during VLSI physical design. So basically, your channel is depicted as a rectangle and the neighbors they have a common boundary they are connected by the edges in the graph. So let us say if you can have this kind of representation in case of a VLSI physical design, this is the placement of the blocks in VLSI physical design. Then the corresponding neighborhood graph will be A is touching B, so there is an edge between A and B. A is touching C, so there is an edge between A and C. Similarly, you can do it for all other edges. So, this is called neighborhood graph. So in this lecture we discussed about graphs and different terminology related to graph and how can we represent graphs using some kind of data structure like adjacency list and adjacency matrixes. Then we discuss about various graphs related to VLSI physical design and its uses in VLSI routing, placement, all these things we discussed in this lecture.

Thank you for your attention.