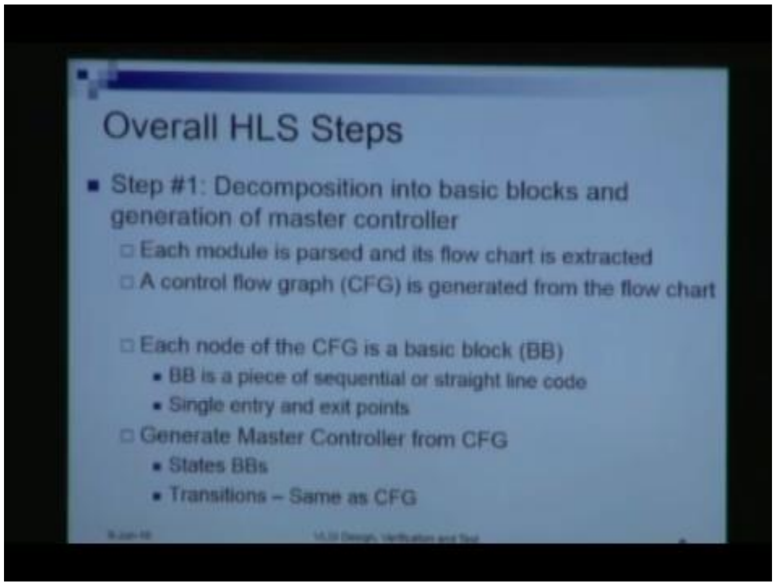
We will begin module 2 of lecture 2. So the overall steps are as follows instead what we become force the system into basic blocks and generate the master controller. So what we have here and what do we have we said that we have a set of concurrent communicating processes and each of these modules will be a code.

Now this code will have a control and wake up yoga so what we are saying here each module is fast and its flow graph is extracted. So if I have a program if I have a high level language or hardware description language program I can extract the flow graph or the flow chart out of it right.

(Refer Slide Time: 00:54)



Overall HLS Steps

- 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

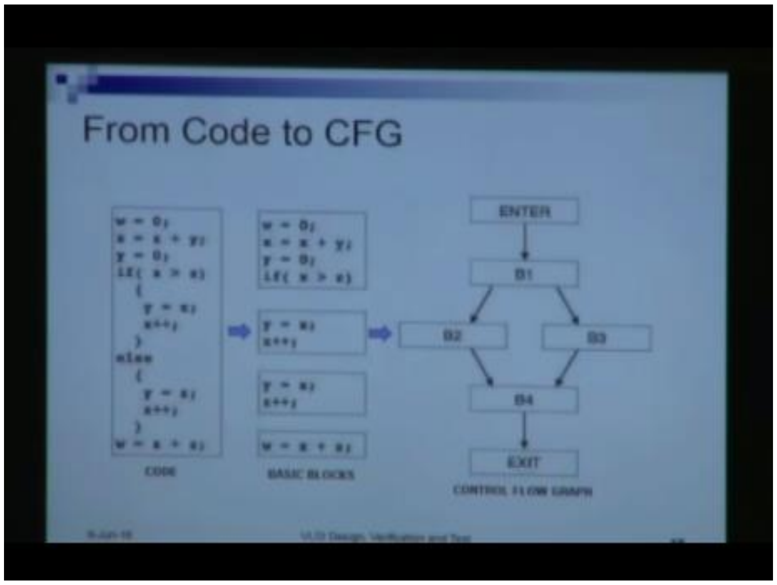
Now given this flowchart from this flow chart it is easy to obtain a control flow graph of the application. Now what is a control flow graph of this program in this each mode of this graph will be a basic block and the edges will define the dependencies between these basic blocks right. So what is the basic block, a basic block is a piece of sequential or straight line code which has a single entry point and a single exit point.

So therefore a basic block is a piece of code which does not contain conditions or loops inside it which means that the control, the program control will never come to some point which is inside this basic block. So the control will come only at the beginning of the basic lock and control there will be a sequential straight line code set of statements control will flow from one statement to the another new sequence through the basic law and come out of this sequence okay.

So neither can control go out in between this basic block somewhere neither can control come inside the basic block within a set of statements right. So this is the definition of a basic block, given this control flow graph we generate the master controller. So basically the master controller has a one-to-one correspondence with the control flow graph.

So the states of this master controller definition will be the loads of the control flow graph and the transition will be the edges of the control flow graph right.

(Refer Slide Time: 02:46)



From Code to CFG

```
w = 0;
x = x + y;
y = 0;
if( x > z)
  {
    y = x;
    x++;
  }
else
  {
    y = z;
    x++;
  }
w = x + z;
```

CODE

```
w = 0;
x = x + y;
y = 0;
if( x > z)
```

```
y = x;
x++;
```

```
y = z;
x++;
```

```
w = x + z;
```

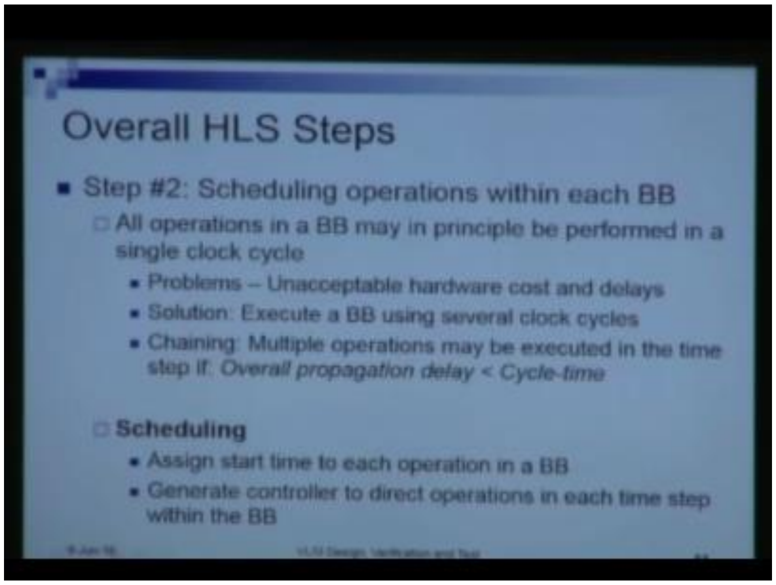
BASIC BLOCKS

So now we take an example to show it let us say that we have a code like this w=0, x=x+y, y=0, if x>z then y=x x++ else y=z x++ and at the end w=x+1. So we see that what are the pieces of straight line code we have here first we start with w=0 there are no conditions or loops x=x+y, y=0 and the if statement, the statement if will always so once we enter this block there is no way in which the if statement will not be executed.

So the if statement will be part of the first basic block right, because there is no chance that I will not execute the each state but at the control will flow outside the first basic block. So therefore the each statement to be part of the first basic block. Now if the condition is true we go to the second basic block y = x++ if this condition is false we go to the third basic block y=z, z+y=z and z++ and then we have a fifth basic block –doubles from X +1.

So we see that these are the maximal pieces of trade line code that are possible from this program and then each of these basic block so this is basic block 1 this is basic block 2 this is block 3 and this basic block= 4 so we have an entry point so another important point here is that as a CFG is polar in the sense that it has a single entry point and the single exit by the source and this thing so it enters through the first it enters and then it comes into the first basic block if it is if it is true it executesV2 if it is false it executes v3 and the condition here if it is true or false and then it executes v4 finally exits okay. So this is how a control flow graph use.

(Refer Slide Time: 05:05)



Overall HLS Steps

- 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
    - Chaining: Multiple operations may be executed in the time step if: *Overall propagation delay < Cycle-time*
  - **Scheduling**
    - Assign start time to each operation in a BB
    - Generate controller to direct operations in each time step within the BB

Now give up this control flow graph we said that we can obtain the master controller so what will this master controller come from how and when should each basic block be activated right however we said that the master controller is very similar to the control flow graph that is what exactly the control flow graph why did we say that because we cannot ascribe timing information to the master controller until we have expect timing information to the basic blocks themselves.

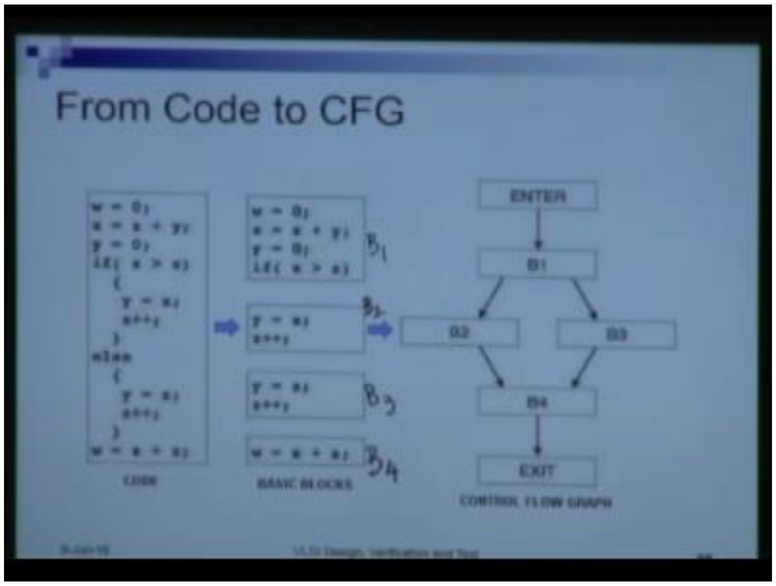
So after we have expect timing information to the basic blocks themselves we can expect timing information and then it becomes a full-fledged controller so therefore before we can generate the final must master controller we have to first look inside each basic block so let us therefore take a look what happens within a basic clock.

Now we what we have to do is we have to schedule the operations within a basic block so then what is the basic block if you see we will see that a basic block as low control or groups inside it so therefore it only contains a set of parts for data to flow so control data will go into at the beginning of the of this basic block data will enter input data with enter these input data will go through a series of transformations because of the combinational operations it is not sequential why because there are no controls no loops so ideally all the operations or computational operations so the data flows through the set of combinational operations and comes up at the end of the basic block.

So what we understand is that the basic block therefore can be represent through a data flow graph the entire application module as such is therefore a control and data flow graph the control flow graph represents what the nodes represent the basic blocks and the connections between the basic blocks and each basic block is it is in itself a data flow graph so the whole combined a graph that we get is actually up control and data flow graph it is an identical structure that you are looking at right.

Now because there is there are only combinational components inside a basic block we can very well understand that all operations within a basic block in principle can be performed within a single block cycle why because let us if you go into the previous basic block.

(Refer Slide Time: 07:58)



From Code to CFG

```
w = 0;
x = x + y;
y = 0;
if( x > z)
  {
   y = x;
   x++;
  }
else
  {
   y = z;
   z++;
  }
w = x + z;
```

CODE

```
w = 0;
x = x + y;
y = 0;
if( x > z)
```

B1

```
y = x;
x++;
```

B2

```
y = z;
z++;
```

B3

```
w = x + z;
```

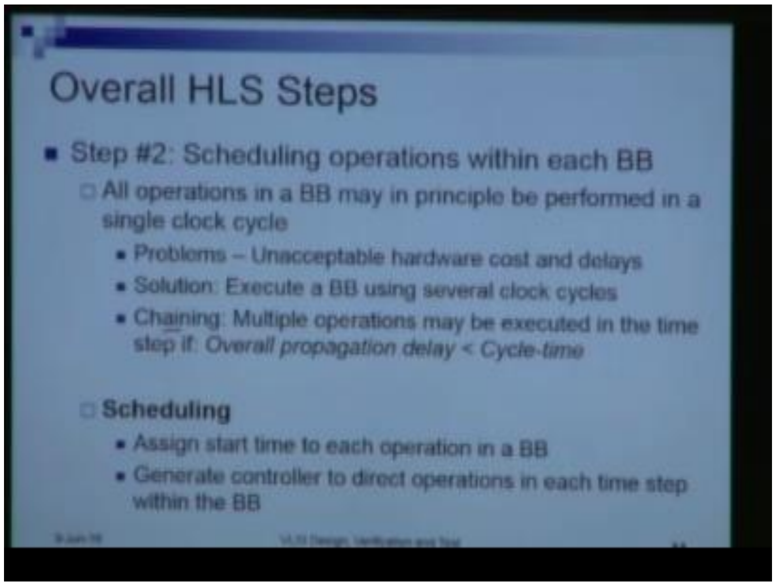
B4

BASIC BLOCKS

VLSI Design, Verification and Test

Here w = 0 x =x + y, x = 3, 1 this is X greater than x greater than Z this X greater than the computation we see that the there is no controller loop inside it so data enters everything cannot be done in parallel with can it has to be done in sequence but there is no time we do not need time to explicitly fast in between them. In a single clock step I can do all these operations together.

(Refer Slide Time: 08:28)



Overall HLS Steps

- 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
    - Chaining: Multiple operations may be executed in the time step if: *Overall propagation delay < Cycle-time*
  - **Scheduling**
    - Assign start time to each operation in a BB
    - Generate controller to direct operations in each time step within the BB

However if we have multiple many operations within a single clock step the problem is that it will lead to unacceptable hardware costs and delays why hardware past let us say we have four say compared with comparison operations between a basic block now all this because it is a single block step I cannot reuse this comparison at the comparison operator and different proc steps all these comparison operations have to be executed on different comparator units.

If I assume that I have comparator hardware resources so it has to be executed using different corporate and hardware units in because all is being done at a single clock step, so likewise for all different types of operations all has all has to be given a separate operators or functional units to execute and therefore it will lead to a little unacceptable hardware cost because of this reduced resource sharing that we have.

When we divide things into different times that when we divide it in two different times that is what happens is that I can share the same resource at different time steps which is not possible if I do it within a single cluster and also it will also increase deal if I let us say this clock is not only feeding this basic block here this clock is possibly feeding many other basic blocks indifferent parts of the in this module maybe between this model and in other modules of the whole application.

So therefore all these applications will have to incur this delay and there could be certain or there could be very small basic block search when all basic blocks will not be this big so all small basic blocks because we have said that all operations within a basic block will have only one clock step we will execute whatever be the size of the basic block small or big we will do it in one block step.

So there will be many small basic blocks which will very quickly complete its operations and tests it because of this big basic block takes just one clock cycle and hence the length of the clock cycle has to be big so that all these operations can be executed within this one clock cycle and many basic blocks will sit idle after completing or its operations and hence it is not advisable that you do many operations within one block step.

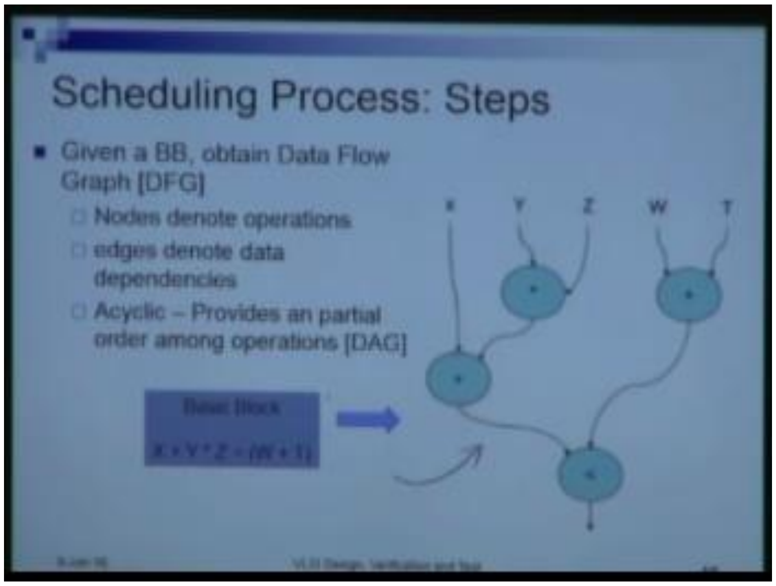
There is a trade-off which we need to have so because we are not because the scheduling all operations using a single clock step we are dividing the operations within a basic clock in two separate clock steps then we have this problem of scheduling so we have to appropriate issue in two different operations in two different concepts, so execute a basic lock using separate clock cycles and that will require scheduling.

Chaining is another concept where they can set the propagation delay of a particular operation is various one and I a particular operator I have a functional unit whose the propagation delay is very small that means it can execute the operations within it very fast such that I can all acts a multiple number of operations within a single clock cycle, so I think that is possible and chaining is the computer changing is the way by which we do it so what is changing multiple operations may be executed in the same clock step if overall propagation delay of all these operations together is less than a proctor cycle time, right is less than a cycle so therefore this only complicates the scheduling problem itself, okay.

So what is sheer dueling as such the scheduling problem is to assign a start time to each operation in the basic block and generate controller to direct operations generate controller to direct operations in each time step within the basic lock, so what we what I it mean you are assigning a tiny step to each operation in the basic block this operation can be executed over multiple cycles so the delay of this operation can be over multiple cycles.

So an operation can take four clock cycles to complete that is possible, however we will be assigning the start time of that operation so the scheduling will it will a distinct start time to each operation, okay.

(Refer Slide Time: 13:24)



Scheduling Process: Steps

- Given a BB, obtain Data Flow Graph [DFG]
  - Nodes denote operations
  - edges denote data dependencies
  - Acyclic – Provides an partial order among operations [DAG]

Basic Block

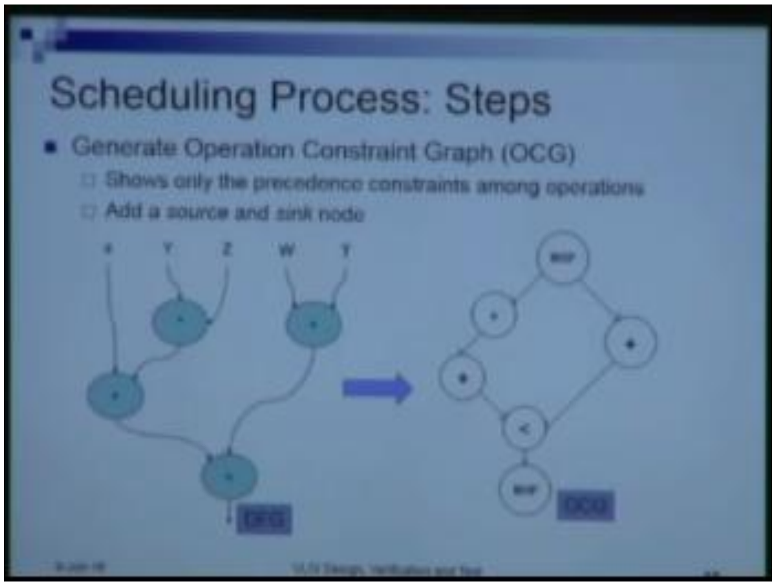
X + Y * Z – (W + T)

X
Y
Z
W
T
VLSI Design, Verification and Test

Now how do we do that, we said that within a basic block we have this code which is sequential and straight line so therefore first we obtain a data flow graph for it the angels in this data program denote data dependences and the data flow graph is a cyclic and why is it a cyclic because there are no loops inside so our only straight line control flow occurs there are no conditions in it.

So what happens is that the data flow graph is a DAD or it is a cyclic and hence we can have a partial order among its operations, so it because it is a cycling it provides a partial order among its operations and let us now consider a very small basic block let us say (x+y)z less than w+c is it obviously. Now this basic block can be converted into this corresponding data flow graph so this is within a basic block the control is outside, so the first here we have y.z, y.z (x+y)z so this is what is happening w+t on this end and the comparison operator in here. So this one is a data flow graph corresponding on this basic flow.

(Refer Slide Time: 14:55)



Scheduling Process: Steps

- Generate Operation Constraint Graph (OCG)
  - Shows only the precedence constraints among operations
  - Add a source and sink node

Now from this control flow graph we simplify as from this data flow graph we simplified it a bit to help us engineering. Now first what we see is that the variables that independent variables will anyhow be kept in separate resistors, but what particularly need in scheduling we want to schedule operations in the basic operations at different time steps within the basic block, right so operation that should be scheduled in different types so what basically need is the constraints distance constraints within the operations themselves.

If the data is independent we are not it is not really important to us at this stage, so therefore from this data flow graph we can obtain an operator constraint graph like this here the what it is almost same as the data flow graph but it only shows the operation constrains not the operator constraints but the operation constrain graph.

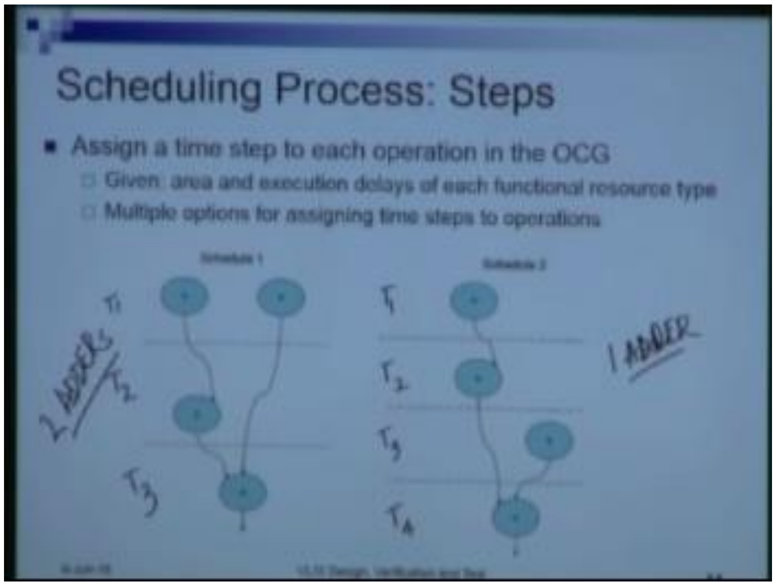
So it shows the precedence constraint constraints among the operators operations so this is it so and then we add a source and sink so data flow graph has a single start but at the single input so therefore I can add a source and sink to mantilla obviously no ops and we get this operation constraint graph from this corresponding data flow graph.

Now only this operation constraints graph we will schedule now in this particular graph we have for the sake of simplicity we have removed the source and sink notes and then we have shown to ordinary schedules so what we think the process of scheduling is to assign a time step to each operation in the operation constraint graph so it will assign a start time to each operation in the operation constraint graph.

So what do we have we know the area and execution delay of each functional resource type that is available to us so this is known at the input of the schedule in problem so what are the different resource types that we have and what are the character is what are their characteristics in terms of what will be the area consumed on the floor the expected area that will be consumed on the floor by this resource site what will be their propagation delays etc these are known to us.

And the given this we can have multiple options for shooting for example for the previous operation constant graph we can have this example shows two alternative schedules in shaded one we see that this has been done in three steps the schedule what takes three times things so t1 t2 and t3.

(Refer Slide Time: 18:00)



Scheduling Process: Steps

- Assign a time step to each operation in the OCG
  - Given: area and execution delays of each functional resource type
  - Multiple options for assigning time steps to operations

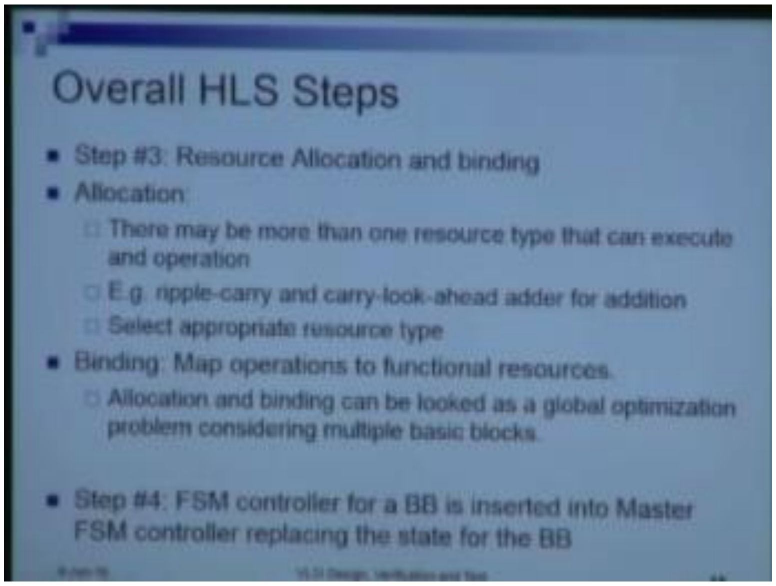
Schedule 1
Schedule 2
T_1
T_2
T_3
T_4
2 ADDERS
1 ADDER

And we see that schedule 2 takes for time steps right however so therefore schedule one we can say that schedule one has a higher performance than schedule 2 it means that when I make a hard circuit out of this if I make a hard way circuit using schedule one I will get a faster circuit as compare to schedule 2 however the area consumed by schedule one is there is also hired with respect to schedule 2.

Why because here we see that I can I have to use to adder operation to adder operator resources to added resources I need why because the addition is being performed at the same time step in parallel so these two editions cannot be done using the same resource however in this one the three adder condition operations are three distinct time steps so therefore the same adder resource can be reused in for all these addition operations.

So therefore I can do away with just one adder here but here this will require two adders hence then there are trade-offs and this is a very important trade of which we say at the area versus deliberator.

(Refer Slide Time: 19:38)

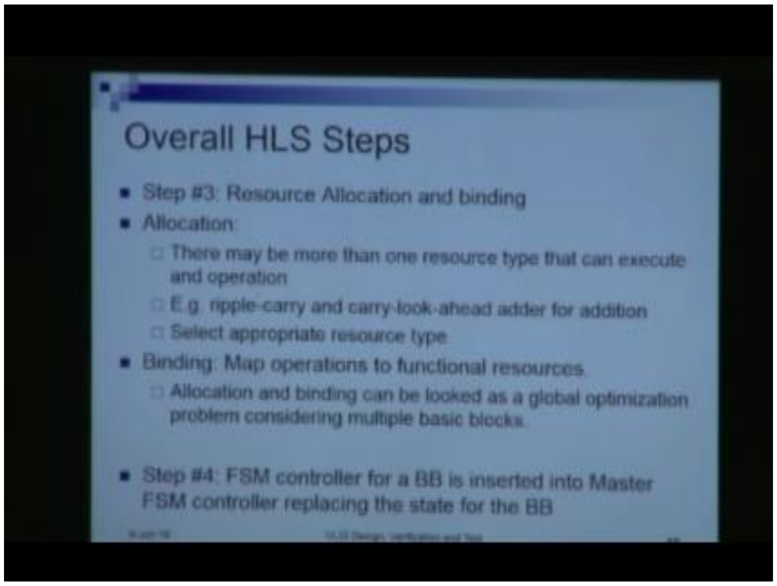


Overall HLS Steps

- Step #3: Resource Allocation and binding
- Allocation:
  - There may be more than one resource type that can execute and operation
  - E.g. ripple-carry and carry-look-ahead adder for addition
  - Select appropriate resource type
- Binding: Map operations to functional resources.
  - Allocation and binding can be looked as a global optimization problem considering multiple basic blocks
- Step #4: FSM controller for a BB is inserted into Master FSM controller replacing the state for the BB

The third important step is allocation and binding so after the shielding step we know that for each operation watch what should be the time step at which it should be executed and at which it should be started what is the time step at which the operation should be started we know that now given that we do allocation and binding what is the allocation problem the allocation problem is this there maybe more than one resource that that can execute an operation now I have to allocate actual hardware resources to the operations I have found out which operation to do in which our steps now functional units have to be mapped to them now let us see that two functional unit types can execute the same operation.

(Refer Slide Time: 20:39)



Overall HLS Steps

- Step #3: Resource Allocation and binding
- Allocation:
  - There may be more than one resource type that can execute and operation
  - E.g. ripple-carry and carry-look-ahead adder for addition
  - Select appropriate resource type
- Binding: Map operations to functional resources.
  - Allocation and binding can be looked as a global optimization problem considering multiple basic blocks.
- Step #4: FSM controller for a BB is inserted into Master FSM controller replacing the state for the BB

There I have to choose which particular functional you need to use so there may be more than one resource diagram can execute an operation for example a ripple carry adder and a caddy look ahead adder can both do a mission so the CM addition operation can be done by the ripple carry adder and they carry look ahead adder so we have to select the resource type through the problem through the through this aspect of allocation then buying incomes where we have to Mac operations to the functional units right.

We have to map operations to the functional rivets so where we do what we do is that we actually buried the functional units to the operations in the schedule allocation and binding and we looked as global optimization as a global optimization problem considering multiple basic blocks will talk of this later if needed but what we want to say here is that this is not the only basic block for which we are advocating and binding resources on now there are multiple modules and multiple, multiple modules and each of these modules there are multiple basic blocks which are executing which may possibly execute concurrently now.

Now all these in all these ones we will try to maximize, maximize the usage of resources so we want to reuse the resource for as many operations reuse registers for as many variables use the bus as much as possible to use right so this is what you want to do so that the resource utilization is maximized and that reduces the total area of the circuit so therefore this allocation and binding step is may be considered a global problem which considers multiple basic block at the same time finally at the last step of the high-level synthesis is to build the master controller for a FSM controller for a basic block.

Now because I have attached timing to each other the shade evening steps I can generate a controller that appropriately activates the functional units and the reach for the appropriate select you through the appropriate select input the correct registers and functional, functional units at a given time now for that we can generate the controller but that does not end so now we have the controller for one basic block that has to be inserted into the master controller of the entire module and finally at cross module so you will have a controller controlling the whole application.

Now this will be done instep 4 then if it is f controller for the basic block is inserted into the master it is a controller replacing the state of the basic block so in the master if it said controller I have a single state in the for the basic block now this single state will be will be replaced by the controller itself for the basic block and the timing of the master controller appropriately existed so with this we come to the end of module 2 of lecture 2.