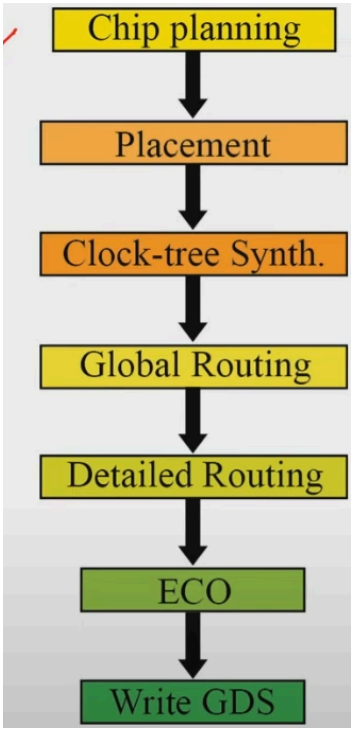
VLSI Design Flow: RTL to GDS

Dr. Sneh Saurabh

Department of Electronics and Communication Engineering
IIIT-Delhi

Lecture 47
Chip Planning - I

Hello everybody. Welcome to the course VLSI design flow RTL to GDS. This is the 37th lecture. In this lecture we will be discussing chip planning. In the earlier lectures we had seen that the physical design task is broken down into multiple smaller tasks and among them the first task is chip planning. In chip planning we basically do the planning for the physical design or the layout of our circuit.



Now chip planning is also a complicated task and we break down into further steps. The first step is related to hierarchical design implementation. Basically for industrial design we implement our physical design or layout using hierarchical design methodology and therefore we need to do certain tasks such as partitioning and so on and that becomes the first task for chip planning. And then comes the task of floor planning in which we place bigger objects on the die and carry out other certain important tasks related to where the object should be on the layout and then we do power planning.

So these are three major tasks that we need to carry out during chip planning. So in this lecture we will be discussing hierarchical design implementation and some aspects of floor planning and in the next lecture we will be discussing floor planning in more detail and we will also be discussing power planning. Now before moving to chip planning let us look into a quotation from Lao Tzu who is an ancient Taoist philosopher. Now this statement is very old but is still very important in terms of planning. So the statement is, while times are quiet it is easy to take action or before coming troubles have cast their shadows it is easy to lay plans.

A journey of a thousand miles began with a single step. Though it is a very old saying or statement it is very much relevant for VLSI. So when we are actually doing chip planning or when we are trying to do some complicated task then what is the suitable time to lay our plans. So the most suitable time is at the beginning when the times are quiet and therefore and before things have become bad we do a good planning for our chip or the layout of our chip and in subsequent steps or in subsequent design flow we carry out or implement those plans in small small steps. So this is how we should proceed.

So this is why we will want to do this chip planning at the earlier stages of physical design. The reason is that at that time we had not yet decided many things about the physical layout of our design and therefore there is a lot of flexibility. There is a lot of flexibility and therefore the most important decisions related to physical design must be taken at that point at the beginning of the design flow or during the chip planning stage. Because if we make a wrong decision at this point then that wrong decision will percolate throughout our design flow and it will impact the final cure. So at the beginning when we are starting the physical design flow we should take the most important state, most important decisions related to physical design and because it offers a great deal of flexibility.

Now let us look into how chip planning is done. Now chip planning depends a lot on what our implementation methodology is, like how we want to proceed with doing the physical design for our chip. So an implementation methodology or physical design implementation methodology are of two types. The first one is flat design implementation and the second type is hierarchical design implementation. So when our design is small in that case what we can do is that we can flatten our design by flattening what we mean is that we get rid of the hierarchy in our design.

For example when we have discussed Verilog we have seen that we write a Verilog top module inside that we have an instance and then that instance refers to a master module and within the master module there can be multiple instances. So we have a design hierarchy. So what in flat design implementation is done is that we get rid of the design hierarchy and in the top level design we have only the standard cells. We have standard cells so whatever is contained in the instances and their master modules we put them or we take them and instantiate directly at the top level. So at the top level we will be having only at the top level of our design or chip will have only say standard cells and the instances of the libraries and some hard macros.

So everything is flattened out that is what the flattening is and then we proceed with physical design implementation or creating the layout for this chip in one go. So this is

what flat design implementation is. Now this method is suitable or is okay or workable for designs which are of small size but when the design size become bigger as in industrial designs we can have millions of instances on billions of instances. In that case the flat design implementation becomes too difficult not only for the designer but also for the implementation tools or the EDA tools that we use. So in that case for a big design when we have a multi million gate design in those cases we proceed with hierarchical design implementation which is a type of divide and conquer strategy.

So it basically divides the complete design into smaller designs and then implements them and then builds the complete design out of it. So in hierarchical design implementation the major tasks are partitioning, budgeting, block implementation and then top level assembly. So these are some of the most important tasks that we need to carry out while implementing our design in a hierarchical manner. Since most of the industrial designs follow this kind of strategy of divide and conquer, let us go and understand what these steps are: partitioning, budgeting and so on. So when we have got a very big design, very big in terms of the number of gates or transistors that it contains.

If it is a very large design in those cases we partition it into smaller parts. Now how to proceed with partition, how do we partition. One natural way is to partition using logical functionality meaning that if we have a chip we think about what part of the chip is doing some function and we group them together . For example if there was a chip and it was doing some it was containing say CPU and another media processor and some other kind of entities at the top level. Then we can say that at the top level it consists of this partition, this partition, ROM, PLL, CPU, controller, power controller, media processor and so on.

And then if the block size or if the partition size is still not good enough for us to handle we further do it in iteratively or recursively going down the hierarchy. For example we can again partition the main CPU to controller, ALU, register bank and similarly ALU can for example in another hierarchy ALU can be further partitioned into say decoder, computational unit, multiplier units and a unit which is doing say DCT computation . So given a complicated design or chip which is doing a complicated functionality we can break down or partition our design into multiple entities based on the functionality . So the different entities that we get though the different partitions that we get are typically known as blocks. So in this case for example at chip level for this chip we have first block which is say consisting of blue logic, second block consisting of say shared cache, third block is of ROM, PLL and so on.

So these are various blocks . Now one block can contain another block. For example in this block main CPU will contain say controller, ALU and register bank which are

further a sub block of the block main CPU . So this is one way of doing the partitioning. Now there are other approaches to do partitioning.

For example we can group modules into clusters. So we have a big design which consists of various modules. We can group together similar kinds of or based on some similarity we group together some of the modules and create hierarchies of modules . That is another way to do the partitioning. It may not be with respect to functionality but maybe we are just partitioning based on how big a block is.

Based on that we are clustering together modules and grouping them together into the same block . Another way to partition is using the say partitioning algorithm. So suppose there is a netlist, there is a netlist which contains say 10 million instances and we want to partition it . This netlist is a flat netlist meaning that it contains only standard cells . This has got only standard cells, but it contains lots of standard cells meaning that there are say 5 say 10 million standard cells.

And it is not easy to handle for the tool for example. Let us suppose that is the case and we want to break it down into say smaller blocks of say 1 million gates . So we will want to create say 10 partitions or blocks for this . Now how to proceed creating partitions . So in that case we take help of algorithms and these algorithms are based on say min-cut algorithms .

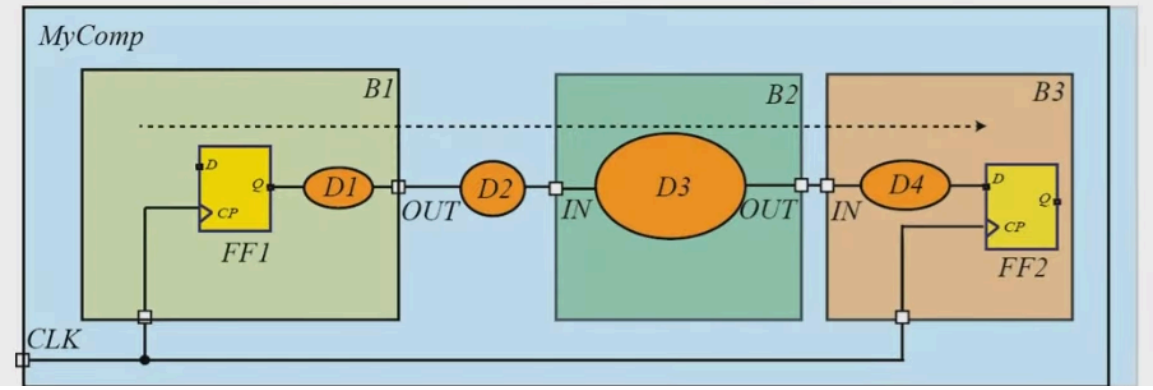
Min-cut algorithm which basically tries to find a partition or cut in our design such that the number of cuts or nets crossing the blocks are minimized . This is one way or this is one of the objectives that can be used by the partitioning tool. So why do we want to minimize the number of cuts? So what we mean by cut is that suppose this is a top level design and we had blocks B1 and this block another block B2 . Now how many nets are crossing between these two blocks are basically cuts . And we want to basically minimize these numbers of cuts.

So why will we want to do this minimization? So the reason is that if we minimize the number of cuts then making interconnections between blocks or creating or doing routing between blocks becomes easy . That is the first reason. The second reason is that if the blocks cross partitions or if nets cross partition or block or crosses two blocks or are lying in two blocks then the timing of that net can become affected or impacted and the delay can increase. The delay can increase because it is going through say from one block and then going through the top level of this entity and then going again into the other block and so on. And therefore the timing of that net can be bad.

And the third reason why we want to minimize the number of cuts is that if we have less

number of cuts then the number of pins that will be in the block, say B1. In this case there are four cuts and therefore there will be four pins corresponding to each cut in the block B1 . Now if the number of pins in a block is less then the physical design for the block B1 will become easier or carrying out the physical design or physical implementation for the block B1 gets easier if the number of cuts are less . And that is why we will try to minimize the cut in our in our in while partitioning and then we can proceed with implementing each of the blocks . So this is how we can do the partitioning of our design and into multiple blocks.

Now once we have done the partitioning of a design then we need to do what is known as budgeting. Now what is budgeting? So budgeting is the process of allocating some fraction of the clock cycle to different blocks and the top level design for signals crossing block boundaries . So if there are signals which are crossing block boundaries then we have to allocate a priori that how much time do we allow a signal to propagate in a given block ok. So let me explain this with the help of an example. Suppose this was my top level entity, this one was the top level entity.



Now this top level entity contains three blocks B1, B2 and B3 and suppose there is a data path which is starting from FF1 from this point this path it goes through a combinational circuit element D1 then combinational circuit element D2 then combinational circuit element D3 in the block B2 and then into the combination goes through the the pins of the blocks and then go reaches block B3 and and it will go through D4 and finally it get captured by FF2 . Now FF1 and FF2 are working on a synchro on the same clock CLK . So this is a synchronous design. Now if this is a synchronous design we want that the signal that starts from FF1 should reach FF2 by the next clock edge . Then only it will be able to satisfy the setup requirement .

So it means that the signal that starts from FF1 clock pin it should the delay between this point and this point between FF1 and FF2 that is delay shown by D1, D2, D3 and

D4 should be such that the it should the the signal should reach the capture flip-flop by the next clock edge by that means that the maximum delay it can have is the clock period . Now how to enforce this constraint that this constraint must be enforced on each of the blocks because later on we will be implementing each of the blocks separately . So to specify the constraint that B1 must honor, B2 must honor and B3 must honor, we create an SDC file or the constraint file separately for each of the blocks. For example we will have B1.

SDC for this one, B2.SDC for the second and B3 for B3.SDC for the third and one for the top level will have top top level top.SDC . Now in this SDC file that we should have for this block we should have a constraint which should say how much time that D1 can take maximum . If D1 takes lot of time delay is very large then the signal will not reach in one clock cycle to the to the block B3 and therefore depending on how much delay we expect in expect to be in each of the block B1, B2, B3 we will allocate some fraction of the clock cycle.

For example if we assume that most of the computation for this path is being done in block B2 and say 70 percent of the delay we expected to be in this path . Then we will write our SDC file such that we allocate 70 percent of the clock cycle to B2 and maybe say 10 percent to B1 and 10 percent to B3 and 10 percent for the top level. Note that even if the top level does not contain any instance for example here it contains the instance or the combinational logic D2 even if it is not containing any instance then also we must allocate some portion of the clock cycle to the top level also. Why? Because the top level when we do routing might go through the top level and it might encounter some delay at the top level and therefore we must not forget to allocate some fraction of the clock cycle to the top level top level entity also . So we create the block level constraints or SDC file .

So this is how we convey the timing constraints that are valid for the top level to the block level . So we are conveying the constraint to each of the blocks related to timing. But there are other constraints for example there can be physical design constraints also. For example suppose at the top level we are thinking or we are planning that a net should be going from say B1 and it goes through B2 and reaches B3 and we are doing some routing at the top level which crosses through or over the B2 and therefore we must allocate some portion of the interconnect at the block level B2 which does not allow the block B2 to utilize that routing resource . So whatever the routing resource constraints are there at the top level we must also convey that routing resource constraint or physical design constraint to the block B2 also.

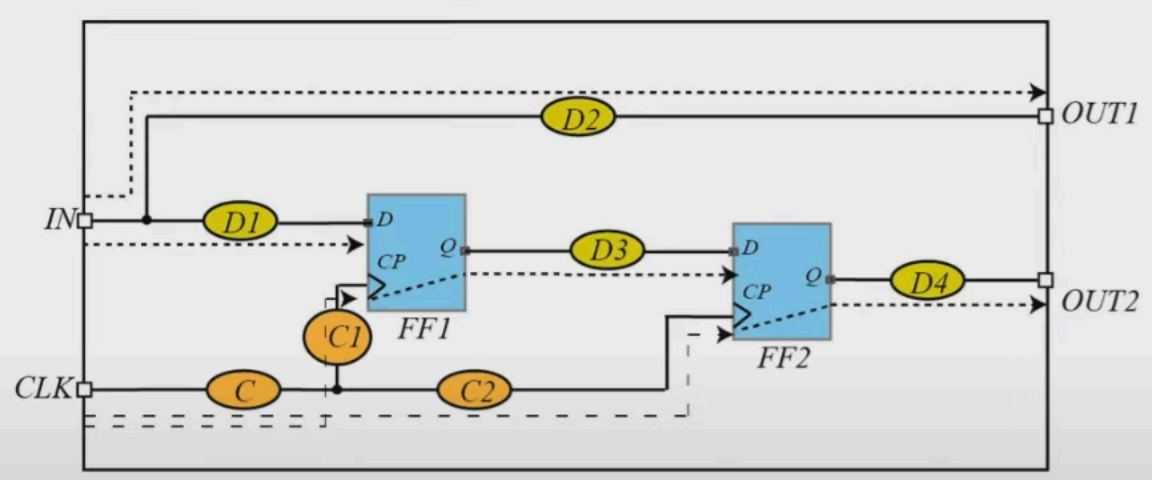
So it is not only the timing constraint that we need to worry about that we need to push

down from the top level to the block level but we also need to push down the physical design constraint that may be imposed by the top level to the block level entities . And it may be also related to the test or DFT path. For example if there is a scan chain which takes input in this block B2 . So we have to say where the scan chain will get the signal meaning that where the scan input is, where is the scan output for the block and then this scan in and scan out can be connected further to other blocks and so on . So the test will also enforce some constraint at the block level and that also must also be conveyed during when we are creating blocks.

Now once we have created our blocks and we have pushed down the constraints that must be satisfied by the blocks we implement each of the blocks separately . Different teams can implement different blocks. For example B1 can be implemented by one team and B2 may be implemented by another team and so on. So while implementing this what must block implementer look into is that it must honor the timing budgets that was given that was imposed from the top level and it also must also satisfy other constraints related to say physical design and test and so on. So the block level implementation should verify that at the block level all the constraints that have trickled down from the top level or pushed down from the top level are honored.

Now once the blocks are ready the blocks are ready then we go into the top level. We integrate these blocks at the top level at the top level and then we must also verify . We must also verify that after the blocks are integrated whether we are getting the correct timing and other other violations are not there for example design rule violations are not there and so on. So when we do top level assembly we need to verify.

Now here the problem comes . Now if we put everything in the block that was there at the top level and we carry out verification then the complexity of verification at the top level remains the same because all the blocks are there at the top level . But then what we can do or how we can avoid this problem is that we can omit the details of the timing pass contained entirely within the block. Why can we omit? We can omit it because while doing the block implementation we have already verified that at the block level the constraints are met . So the flip flop to flip flop path which are say contained fully within a block that we need not verify again at the top level we can simply ignore them . For example suppose this was our block suppose this one was B1 and this we want to implement it at the block level and we want to put it at the top level .



Now in this block B1 there are different types of paths for example there are paths from this flip flop FF1 to FF2. Now the timing of this path has been verified while the timing of this path has been verified while implementing the block B1. And therefore we did not verify it again . So what we should do is that at the block level we should only be worried about the interface path and what are these interface paths. So the interface paths are the paths which come directly from the inputs for example, this is an interface path .

This is an interface path which goes from input to the output and this is an interface path which is directly going to the output. So these are interfaces and these path characteristics must be known when we are doing the top level timing analysis. So when we are doing top level timing analysis then we need to know the characteristics of the interface paths of the individual blocks in our design. Suppose there was a design top and this B1 was there. This B1 that I have shown here was instantiated here. Now this B1 can receive signals through this input and finally to this flip flop FF1 FF1 .

So at the top level we must verify that the delay of this top level entity is not large such that it introduces some timing violations within the block . And therefore we must know the characteristics of the interface paths while we are doing verification at the top level . So we retain the information of the interface timing paths of the block level and we create an abstract model. What is an abstract model? So in the abstract model we get rid of all the internal paths, meaning flip flop to flip flop paths . So now in a typical design there are lots of flip flop to flip flop paths and therefore if we get rid of those paths most of our design becomes very very small or the block becomes very very small.

And therefore we are saying that we are creating an abstract view of the block because it is a reduced form of a block. We have got rid of the information which is not required for verification at the top level. And there are many ways in which these interface

modeling of interface paths can be done. One of them is using say extracted timing model which is a model using dot lib file and the other can be using interface library model which is based on or which models the interface paths using logic. So now commercial tools can support this abstract model in many other ways .

So I will suggest that please look into the manual that you will use for the tool that you will be using in your physical design implementation. In those manuals it will be indicated how an abstract timing model can be created for a block and you can use that model when you are integrating that particular block at the top level. Now what are the merits of hierarchical design methodology? So the first merit is that the physical implementation and other EDA tools need to handle smaller problems. So if there was a 10 million design we partitioned into 10 designs of 1 million. Now EDA tools need to handle only 1 million designs rather than 10 million.

And therefore the physical implementation tool will be the problem for the physical implementation tools that become smaller or get reduced. And therefore in physical design implementation we can use even those tools which cannot support a large number of gates at the top level. For example suppose there is a tool which cannot handle say 10 million gate designs. That tool is supporting say 1 million gate designs. So we can partition our design into 10 such small small blocks and then use the same implementation tool.

So we can get a design which is or finally we can get the top level design which consists of 10 million gates which is beyond the capability of the physical implementation tool. And it also actually speeds up the design implementation. Why? Because once we have done the partitioning then each of the blocks can be worked on concurrently by different teams. And therefore the overall design time of a chip can be reduced. Now what are the disadvantages of block hierarchical design methodology? So one of the disadvantages is that it is challenging to partition a design optimally.

Of course we can design , we can partition our design in many ways but is it optimal? Are we getting the best best best figures of merit? The answer is that it may not be because the problem of partitioning itself is very difficult or challenging. And the second is that we may lose some opportunities of interblock optimization. So a physical design tool is only seeing the view of a block. And therefore if there is some scope of optimization between two blocks there are paths which are going from one block to another and some optimization in timing and other characteristics can be done. Those opportunities can get missed in this hierarchical design methodologies.

Despite these disadvantages for typical industry scale designs we use hierarchical design

methodologies because it speeds up our chip design process and it becomes easier for the tool and also designers. Now let us go into the second task of chip planning that is floor planning. Now what is floor planning? So the floor planning is the planning phase for the layout. We are creating a layout and for the creation of this layout that the planning phase is known as floor planning. So this is our floor. Creating a floor plan is basically a way of conveying our intention or our intent about the physical design or the layout.

Now our floor planning basically prepares a design for other physical design tasks and therefore it has a huge impact on the figure of merit or F1. So the rest of the design flow from floor planning onwards for example the placement of the clock tree synthesis and routing those will be highly impacted by what decisions we make during the floor planning stage and therefore floor planning is very very important. Now during floor planning we must consider the routability of our design, performance, power and of course during floorplanning we must pack the entities as compactly as possible. Now what are the major tasks that we need to carry out during floor planning? The first is to define the die die size or chip size and the aspect ratio of the die or chip. The second is placing a IO cell or input outputs what are these IO cells we will see in subsequent slides and then there are we need to place hard macros or blocks during floor planning stage and then there is a task called pin assignment which needs to be carried out during floorplanning and also we need to create rows of for standard cells.

Note that we are not placing standard cells, we are just saying where the rows of the standard cells will be on the layout during the floor planning phase. So we will look into all these tasks in some detail in the subsequent slides. So the first thing that we need to do in floor planning is to decide the die size or we need to choose the smallest size of the die that can fit the design. Now why do we want to choose the smallest size of the die? So the reason is that if our die size is smaller then on the same wafer we can have more dies and therefore the cost per die of a chip will come down and therefore we should have the die size as small as possible . And the other thing is also related to the yield in from the earlier discussion you must might remember that yield is kind of inversely proportional to the area meaning that if area of a die goes down then the yield goes up and therefore we want to reduce the area of the die so that yield can be moved and therefore we will be saving on the due to yield gains also if our die size is smaller.

Now what are the restrictions on the die size we can make it a smaller and smaller and smaller but how smaller the lower limit on the die size will come from that what entities we want to place on our die we must while deciding the die size we must include the area of the major components that will be there on our layout . So in this figure I am showing a very simple kind of layout. Do not take it as a we have a this is a typical

layout but a simple layout in which I am showing various components that can be there on our layout . So the components that can be on our layout and for which we must allocate space on the die are first is the IO cells or input output cells or IO pads they are also called so we must have we must allocate space for this . So this is the space we are saying we must allocate for the IO pads, the blue one the blue area that we have shown that we are allocating for the IO cells or input output cells through which the signals will come to our chip or die . Now the other entity is the standard shells so the standard shells that are in our design must be placed on the layout .

So we must allocate space for the standard shells and standard shells we typically put in the rows . So these are the rows of standard shells that we must think that we must allocate space for . Now what should be the area of this space so when we are doing floor planning by that time we have already got the net list . So from the net list and using the libraries in the technology library we can get what is the area of the standard cells total area of the standard cells and that will give us at least how much space we must allocate for the standards . Similarly we will have macros so macros can be a hard macro meaning that there can be memory elements or so on or these macros can be say the blocks this is a top level entity and on this there will be blocks that we have got after partition .

So we must allocate space for the blocks and macros and sometimes we need to also allocate space for hello. So around the macro we say that there is a sub region where we do not allow anything to be placed . So those become some area around the macro where nothing is allowed to be placed and as such that area must also be accounted for when we are designing when we are deciding the die size. Now why we are allocating this hello or empty space around the block we will see that in the subsequent lecture. And the final thing is the interconnect and this is the most interesting or most difficult or challenging thing for floor planning.

So when we are doing floor planning we have not yet done the routing but still we need to allocate space for the routing and or the interconnects that will be connecting various entities on our chip or on the layout. Now how to do that because we have not yet made the layout and we do not know what the interconnect area will be required . But before starting the floor plan, sorry the physical design we must know what is the die size and therefore this is a challenging task we have to do some estimate . So typically we use a measure which is known as utilization to allocate some portion of our layout to the interconnects.

Utilization = (Cell area + Macro area + Halo area) / Core are*a*

So what is utilization? So utilization is the ratio of the cell area . So if we consider this core, the area of the die that is left after we have removed the portion of IO cells is the core. So this is the core area which is shown in pink . So this is the core area . Now in this core area what entities are sitting the standard cells, macros and the hellos . So if we take all these ratios of all these entities and take the total area and divide by the core area that is the whole area that is shown in pink we get how much extra space is left for interconnects .

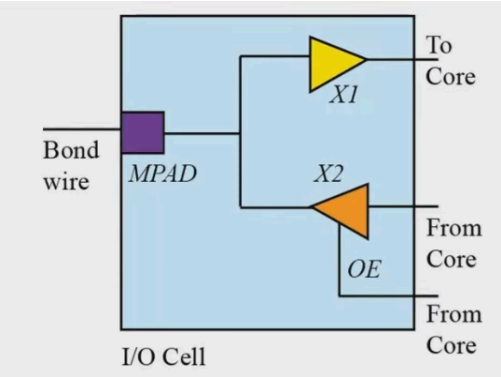
So if we divide we get the number which shows how much utilization we are doing, meaning that if we are saying that there is a 100 percent utilization then no space is left for interconnects . Ideally we will want that situation we will want that all the layers and all the interconnections of the entities can happen over the in these entities. Note that the interconnects connection is done in layers and in industrial designs we have 10 to 12 layers of interconnects available. So ideally we will want that all the interconnections are done at the higher levels of metal and through standard cells over the macros and so on and all the routing is done over these entities and we need not allocate any space for the interconnects .

So in that case utilization will be 1 or 100 percent . We want that I that is the ideal situation but in that case finding a feasible solution will be very very difficult and the routing can fail at the when we when we do say the the detailed routing at that time the tool may say that I do not have sufficient routing resource to do the routing and then we need to again go back to flow planning and other other other chip planning stage and allocate extra space and so on. So therefore during the initial stage itself we allocate some portion to the interconnect and we say that the utilization is rather than 1 we say that it is something between 0.6 to 0.8 . Typically we use these numbers and this number will basically come from say the previous designs .

Previous designs will give us some kind of number based on which we have a good number or good estimate of the utilization and which we have used in closing or achieving design closure in the previous designs . So from the previous designs we can get some good estimates for the utilization and we can move forward . And also we should understand that the die size is also determined by the package size or it is very strongly dependent on it. It also strongly depends on the package size which is available to us . So the package size and its aspect ratio many times decide what should be the size of our die. So once we have decided the size of the die and its aspect ratio the next part comes in deciding the locations of the IO cells.

Now what are IO cells? So IO cells are special circuit elements to which a chip

communicates with the external world. So the external world will communicate with our chip or through some pins or through some cells or special cells. So those cells are known as IO cells and they are also known as IO pads . So these IO cells can be of many types. It can be of say input type, it can be of output type or it can be of bi-directional type.



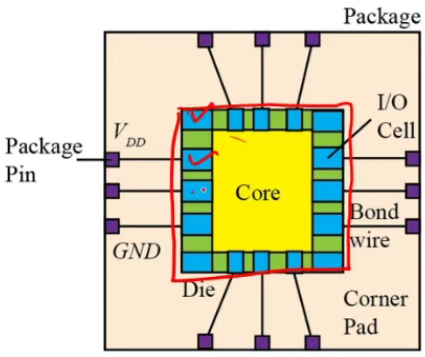
So in this figure I am showing you a bi-directional IO cell . So in an IO cell there is a metal MPAD which I am showing this is a metal through which a connection is made . So from this metal the wire will go to the package that will show in the next slide but understand that this is the IO cell and inside this IO cell there will be some metallic pad through metallic contact from this the wire will go to say one buffer and another buffer . So there are two buffers. So whenever there is a signal there is a signal which is OE which is coming from the core OE is output enabled.

So if OE is equal to 0 this tri-state buffer is in high impedance . This is a high impedance. We can consider that it is an open circuit and in that case this IO cell will work as an input signal whatever will be coming at the input that will be going to the through this buffer X1 it will go to our core logic . And if OE is equal to 1 OE is equal to 1 meaning out 1 is equal to enable then this IO cell is will work as a output output cell meaning that whatever will come from the core whatever the value 0 or 1 will come from the core that will be available at this MPAD and that signal can be read by the external word . So that is why this is the IO cell that I have shown here is a bi-directional IO cell. Now what are the functions of IO cells? So one of the functions we have already seen , that it communicates with the external one . Other than that there are other functions of the IO cells the most important being the drive capability to enhance the drive capability.

So typically the cells that are driving say from the core logic that is coming from core logic those cells may not be may not have sufficient drive capability to drive the load that will be driven at the output of this line . And therefore these buffers that are there in the IO cell must enhance the drive capability . So the IO cells typically enhance the drive capability using buffers or a chain of buffers then there can be voltage transformation also . So these IO cells can also work as voltage level shift meaning that the signal that is coming from the external world may be at some other voltage level and our core logic may be working at some other different level .

So the IO cell can provide this voltage transformation between these two entities. Then the third important and very important function for the IO cell is that it gives protection to our circuit . What it means is that while we are handling our chip when we are handling our chip we may touch the pins or even the machines can touch the pins and because of that there can be an electrostatic discharge. What is an electrostatic discharge? So electrostatic discharge is a short duration very high voltage pulse; it may be several kilowatts . So you might have seen this electrostatic discharge may be while you are touching say the handle of a door and you got a certain shock and that happens because there was a charge transfer between our body and the and say the door handle and that through the ground and so on . And therefore when we handle the chip either through machine or manual handling there accidentally this electrostatic discharge can even happen meaning that as a very high several kilovolt pulse can be generated but for very small duration .

And if we allow that very high voltage to go to our core circuit our core circuit element gates that those transistors that may get damaged. So the IO cell will have some circuit element not shown in this figure to prevent the electrostatic discharge to penetrate or penetrate and reach our core core circuit. Now how are these IO cells connected to the package? So let us understand. So this is the core this is our die this is our die and these blue elements these are IO cells these are IO cells.S



Now we saw that in IO cells there was a metallic element MPAD. So that MPAD from there the wire will go and will connect to the package pin . So there is a package pin. These are package pins that are shown here . So when we think of a chip we look into the to the to the end and there we see that there are some pins for a chip. So those are the pins of the package and that is what we are referring to here as package pins and those package pins are connected to our die through this wire, this metallic wire. So these are known as bond wires and these bond wires are finally connected to MPAD in the IO cell that we saw in the previous slide .

And through that the signal will go through to the core logic all or the signal will go out from the core logic . Now there are other kinds of IO cells which serve the purpose of bringing in power and ground lines to our cores or to our circuit and those are known as power pads. So power pads are special cells that supply power to a chip . For example

this is a this is this is the pin of a chip which is VDD . So from here the power is coming and this will go to this IO cell and this IO cell will be a power pad .

Similarly there will be a ground pad for example this is a ground pad . So these power pads must also be placed on our die. So functionally these power pads are just providing contacts with VDD nothing else . Now another thing to note is that the IO cells also need power and ground connections . For example if we look into the power pad here now these power pads contain buffers and other circuit elements and those will also need a power supply .

These buffers will also need a power supply which is not shown here . You will have a ground line and VDD line and so on . So the IO cell also needs power supply. So typically what we do is that the power supply of IO cells are kept separate. Because IO cells draw a large amount of current it needs to drive larger capacitors and therefore sharp large current transients can occur in these IO cells and therefore there can be voltage fluctuations . And if those voltage fluctuations reach our core cell core cells then core core logic gates then probably those logic gates can malfunction .

And therefore we try to keep the power supply of the IO cells separate and the power supply of the core cells separate. Now how many power pads should we have . Now power pads are supplying the current to our chip . And therefore the number of power pads that we need will depend on how much current is being drawn by our circuit . So depending on the power drawn by our chip or the current drawn by our chip and what is the maximum current limit for an IO for a power pad we can decide on the number of the power pads that will be needed.

For example if the number of if the current drawn by our chip is large then we will be probably needing more power power power power pads in our circuit. And also it depends on the target voltage levels how much we allow the voltage level to be fluctuating in the core for the core logic gates. If the margin of fluctuation is smaller then probably we will be needing more power pads that must be supplying power to our core cells . Now how do we place these IO cells?

So in the floor planning stage we also need to decide the locations of these IO cells . Note that these IO cells are basically designed and those are characterized and they are put in the library cell library sorry technology libraries . So IO cells are available in the technology libraries and the tools can instantiate appropriate our technology lab sorry appropriate IO cells in our net list based on the current requirement drive capability and so on. So the IO cells will be available in our net list or we can instantiate them in our net list . But then the second task is that where to place these IO cells on the layout we

need to decide that also .

So typically it is guided by some heuristics. Now a few examples of heuristics can be like assigning nearby positions to two primary inputs that jointly drive a multi input gate. For example if two inputs are driven by the primary input then probably these two primary inputs should be placed close together or the IO cells corresponding to these inputs should be placed close together using a very simple heuristic . Then there can be another heuristic to spread power hungry IO cells all over the die . So to avoid creating voltage drop hotspots. So if we put power hungry IO cells close together then that area can become a voltage drop IR drop or hotspots .

Now what are these voltage drop hotspots or IR drop hotspots? We will see when we will discuss the power plan . But at this point we can just say that if we expect that the IO cells two IO cells will be drawing a large amount of current or then we can put those IO cells further, that is one of the heuristics we can use while placing IO cells. The third heuristic can be that we can avoid placing sensitive signals such as the clock lines close to the IO cells. Why? So the IO cells as I have said draw a large amount of current and if there are large amounts of current then these are in the vicinity of IO cells there can be large spikes and those spikes can create signal integrity issues. And if the adjacent line is sensitive for example, if it is a clock line or reset line which are more sensitive or we want to make those signals cleaner we should keep those signals away from these IO cells which are expected to draw large transient currents . Now one point that you should notice is that the way that we have shown here the way the IO cells that are connected here to the package pin through wire through the bond wire this is one type of package ok.

So this is a bonded wire package . So this is a bonded wire package . Now for CPU's and advanced chips or high performance chips we use another kind of packaging and that is known as flip chip technology. Now in flip chip technology what we do is that we have bumps. Bumps are basically metallic connections and those are at the top level of the die . So at the top level we will have multiple metallic bumps at multiple locations on the layer . And while packaging what we do is that we flip it, we flip the die upside down and allow that bump to be soldered into our package and allow that bump to make connections .

Now if our inputs and outputs can come even at the middle of our chip for example, suppose this one was this one this was our die . Now in the flip chip technology the bumps can be anywhere in the middle also and the IO cells can be in the vicinity of these bumps. So therefore what it means is that IO cells need not be only on the periphery of the chip as I have shown here. It can be inside our layout also . And if that

is the case we we we we we have more flexibility on putting IO cells close to the place where it is used and therefore the timing and other attributes of our design can improve by putting these IO cells closer to the place where the connections is or input or output connection is actually required for the die .

So this is another thing which you should note . So now if you want to know deeper into the topics that we discussed today you can refer to this book and to summarize in this lecture what we have done is that we have looked into some aspects of chip plan. So we looked into what is a hierarchical design flow and in that what are the basic design steps that we need to carry out and we looked a little bit on floor planning also. So in the next lecture we will be continuing more with or more deeper concepts of floor planning and we will be also covering power planning. Thank you very much. Thank you.