CHAPTER-1
DIGITAL LAYOUT

IC Mask Design

Faculty of Engineering
Alexandria University

Chapter Preview

- Close look at automated layout software
- Why automated layout only works with certain cells
- Knowing the circuit really does what it should
- How to know in advance if your floorplan choice is good
- Automated programs getting stuck
- Troubleshooting tips
- Which nets to wire first
- Which nets to wire by hand

Chapter Preview (2)

- Techniques to guarantee rule-perfect layout
- Flowchart of digital layout procedures
- Lots of feedback loops
- How to keep the power moving through big cells
- Chicken or egg wiring and timing circle
- Did you really build what you designed?
- How to build quickie chips for testing

Opening Thoughts on Standard Cell Techniques

- The majority of integrated circuits built today are large making it impossible for a mask designer to manually layout the chip
- The majority of large digital chips are laid out with the assistance of computer-aided tools
- Understanding how these automated digital layout tools operate allows you to develop skillful daily habits in your work

Design Process

Verifying the Circuitry Logic
- HDL Coding
- Functional Simulation

Compiling a Netlist
- Logic Synthesis
- Drive Strength (Fanout)
- Buffer Cells
- Clock tree synthesis

Floorplanning
- Block placement
- Gate Grouping
- Block-level Connectivity
- Using Fly-lines
- Timing Checks

Placement
- I/O Drivers

Routing
- Power Nets
- Clock Net Wiring
- Other Critical Nets
- Remaining Nets
- Finishing the wiring by hands

Verifying the Circuitry Logic

- Circuit designers typically use languages called VHDL or Verilog to design their enormous digital circuits
- These HDL data files are then submitted to a computer simulator, which tests the chip circuitry while it is still in software form
- The simulator needs to have process-specific descriptions and physical representations of each logic function it wants to use, such as rise time, fall time, gate propagation delays
- All of these files are collectively known as a standard cell library or logic library
- The company that is supplying your silicon usually provides a standard cell library

Example: VHDL Code Segment

VHDL Code Segment

```
architecture STRUCTURE of TEST is
 component and2x
 port(A,B,C,D: in std_ulogic := '1';
 Y: out std_ulogic);
 end component;
 constant VCC: std_ulogic := '1';
 signal T,Q: std_ulogic_vector(4 downto 0);
begin
T(0) <= VCC;
 A1: and2x port map(A=>Q(0), B=>Q(1),
Y=>T(2));
 A2: and2x port map(A=>Q(0), B=>Q(1),
C=>Q(2), D=>Q(3), Y=>T(4));
 Count <= Q;
```

Compiling a Netlist

- The HDL code will be fed to a silicon compiler or logic synthesizer that translates the high HDL code into a file that contains all the required logic functions, as well as how they are to be connected to each other
- The file basically says, for example, "In order to add two 16-bit numbers together, I need 25 gates and here's how they should be connected."
- This file, called a netlist, will drive your automated layout tools
- At this point, we know what gates we need, and we know how they must be eventually wired to each other
- The automated synthesis tool controls the circuit parameters such as speed, area, and power according to the design needs

Example: A Netlist Segment

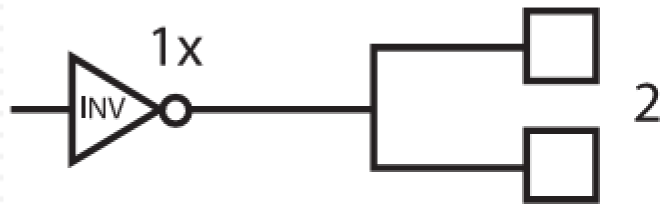
Netlist Segment

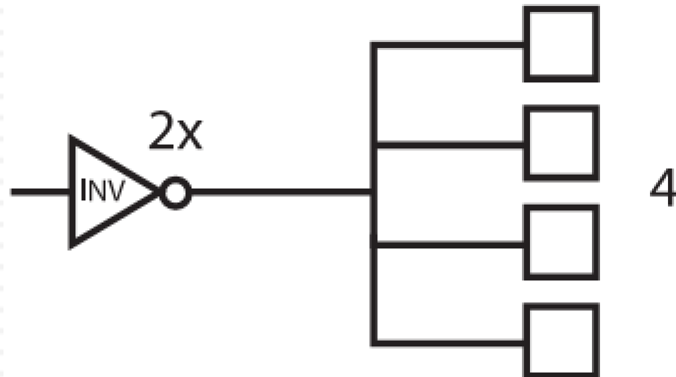
```
module test ( in1, in2, out1);
 input in1,in2;
 output out1;
 wire \net1 , \net2 , \net3 ;
 AND2_2X U1 ( .Z(net1), .A(net2), .B(net3) );
 AND4_2X U2 ( .Z(net1), .A(net2), .B(net3),
.C(net2), .D(net1) );
endmodule
```

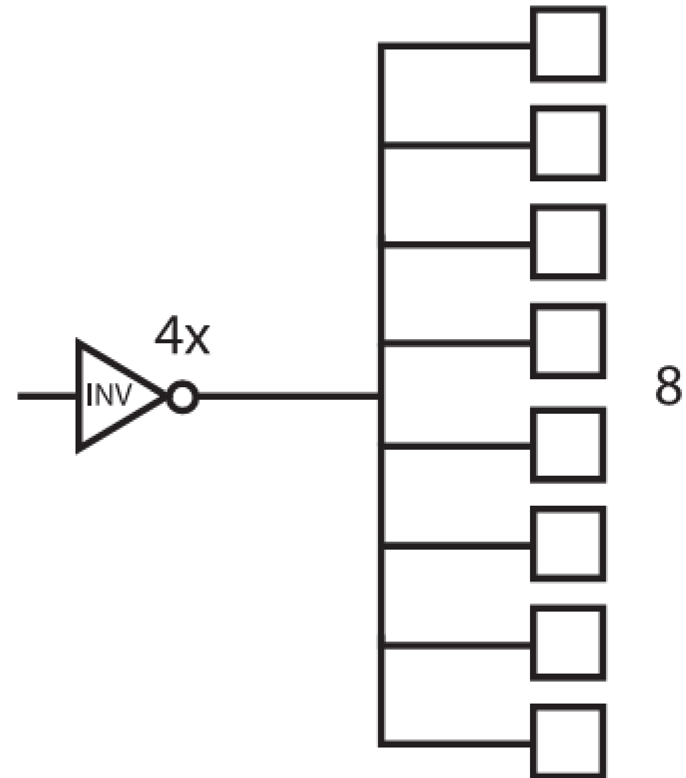
Drive Strength

- If we try to drive too many gates from a single source, we might overload our driving transistors and the circuit will not work
- Therefore, before we can start layout, we need to modify the netlist to make sure that these large nets are adequately driven
- To do this, we replace the cells that are driving the net with cells of identical logic function that have larger driving capability (also called drive strength or fanout)
- The fan out number indicates how many devices a gate can drive
- For example, we might see that our cell library has 10 or 15 different sizes of inverters.
- These inverter selections might be referred to as 1x, 2x,or 4x inverters where x indicates the drive strength

Example: Inverters with Various Drive Strength







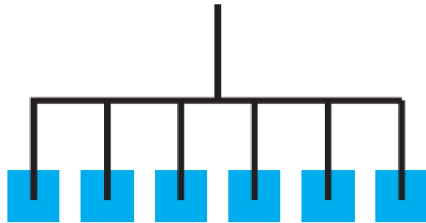
Buffer Cells

- If the compiler breaks a large net into smaller, more easily driven sections, it will insert additional gates to drive each smaller newly created net.
- These extra gates are not part of the original logic and called buffer cells
- Buffer cells help drive gate and wiring capacitance but has no logic function associated with it
- In the next slide, we will see an example of how the compiler uses these concepts to drive a large clock net

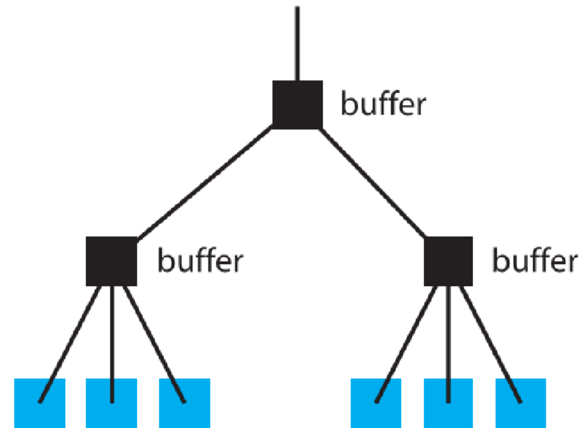
Clock Tree Synthesis

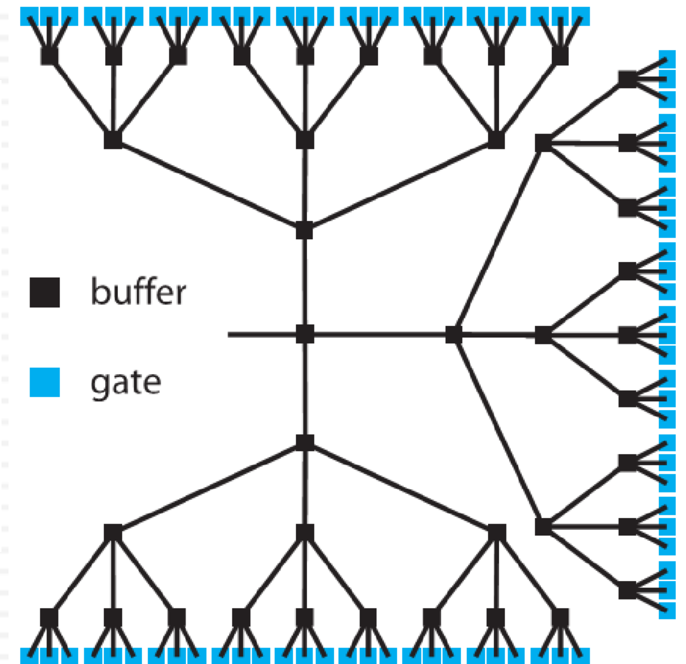
- The wiring nets for the clock signal are called clock nets.
- A clock net is usually very large and connected to thousands of gates
- It is impossible to create a cell with enough drive strength to drive all the gates on a clock net
- We split the clock net into smaller sections and add buffer cells
- The net is split into a branching-out pattern, called a clock tree

Example: Clock Tree



Original clock net
(no buffers)





Post-Synthesis Simulation

- With a large number of added buffer cells, the extra cells will introduce extra delays that were not accounted for in the original simulations
- Not only that, but other large nets may require this same sort of tree synthesis as well, adding even more buffer cells, also creating delays
- Therefore, once the clock net is synthesized, and any other large fan out nets are buffered, we need to re-simulate our design using the compiled net list. Compiling creates a need to re-simulate
- This sort of iteration is common in chip development

Layout Process

- We are now ready to use a suite, or package, of automated software tools called the place and route tools
- Place and route tools cover the gamut of higher level and lower level software assistance leading to your final layout
- As the name implies, these programs generally place the gates and route the wires, in addition to other helpful functions

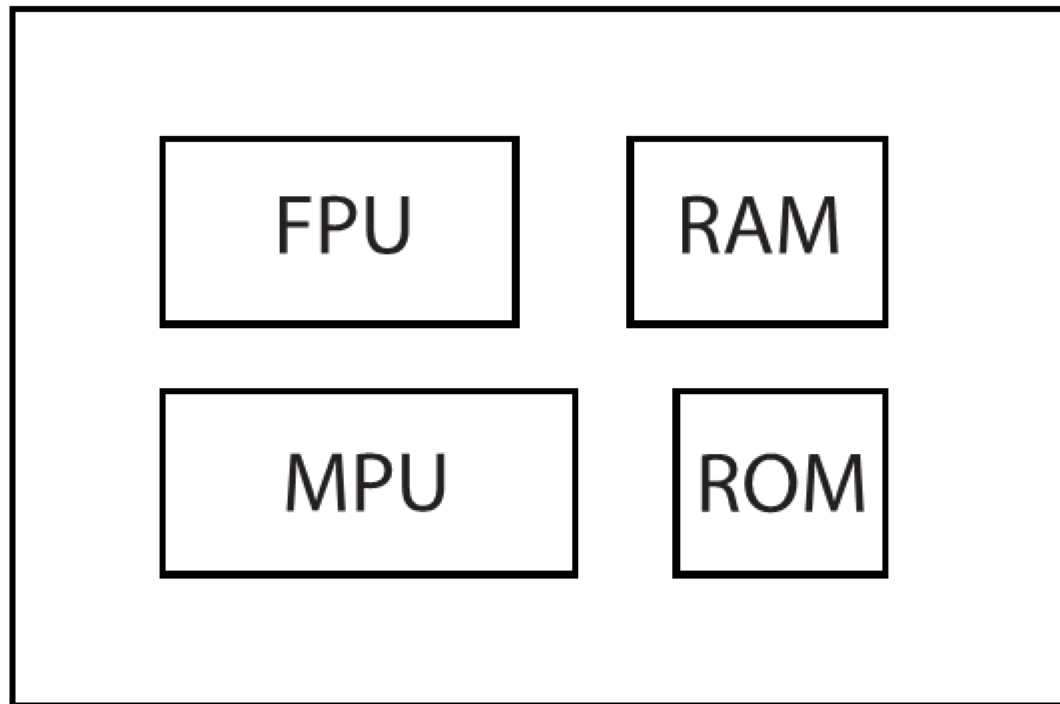
Floorplanning

- A floorplanning tool will help you create areas of functionality on your chip, determine the connectivity between these areas, determine your I/O pad placements, and give you feedback on how easy your floorplan might be to wire
- The floorplanning tool gets its connectivity and gate information based on the netlist file, created by the compiler software
- Let's follow the floorplanning tool in more detail, beginning with your initial decisions

Block Placement

- Typically, your chip will be divided into various functional areas
- For example, if you are working on a large digital chip, there might be a microprocessor unit in your chip (MPU), perhaps a floating point unit (FPU), maybe a RAM block and a ROM block
- Using the floorplanning tool you decide location of each area of functionality and you will have a chance to change these decisions later to get a better floorplan

Example: Floorplanning



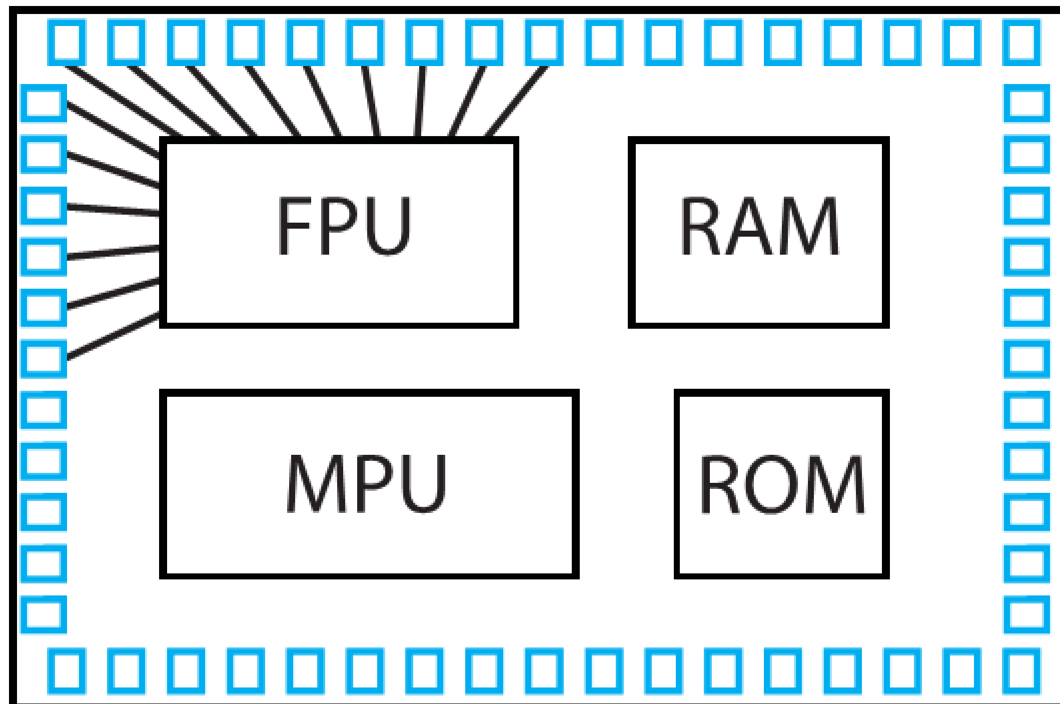
Gate Grouping

- Once your areas of functionality are specified, the first task you would want to do is gather together, to some degree, the gates used in each block
- For example, you do not want FPU gates scattered throughout the ROM or RAM blocks
- Associated gates should all be located near each other where the floorplanning tool helps to gather your gates together
- The exact placement of each gate is not determined at this point.

Block Level Connectivity

- Next, your floorplanning tool will help you place the input and output (I/O) cells of your chip
- For instance, you would want all the inputs that go to the FPU close to the FPU block in the corner
- To help you with this, some tools will actually place the I/O cells in the appropriate areas automatically; other tools will provide graphic feedback for you based on your placement decisions
- The floorplanning tool also shows basic wiring connections that must travel between blocks
- It will show connections between the FPU and the RAM blocks, for example.

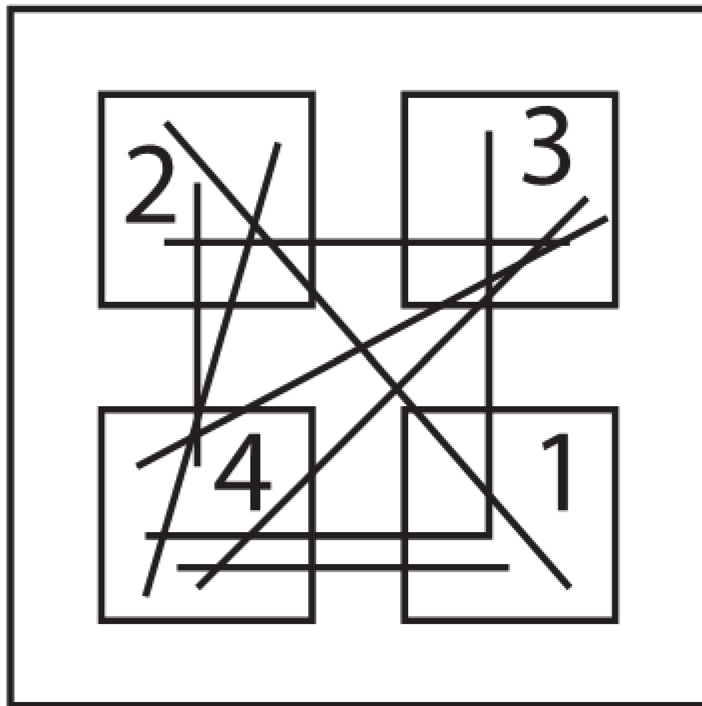
Example: Block-Level and I/Os Connectivity

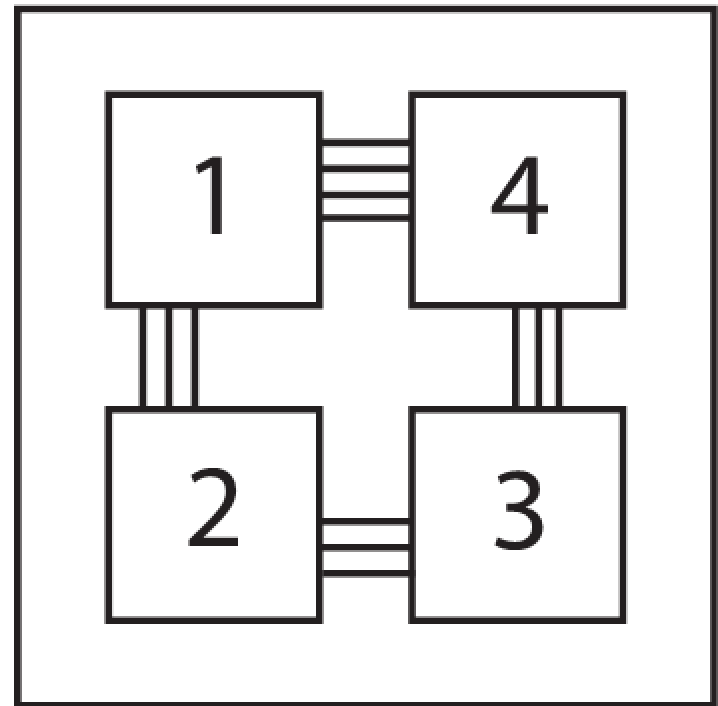


Using Flylines

- The floorplanning tool will show you all the wiring lines coming from each block connecting to the I/O pads and to other blocks.
- All these myriad of wiring lines are what most tools call rat's nestsorflylines
- You will make changes to your block floorplan so that your rat's nest eventually looks as clean, nice, and wireable as possible
- You might decide to relocate entire areas of functionality.

Example: Using Flylines





Timing Checks

- Since the final floorplanning tool output files specify where the gates will be generally located, the placement tool roughly knows how long all the wires will be
- These wiring length estimations are based on the physical dimensions of the digital library
- Using this information, your floorplanning tool can output an estimated wire length file that goes back into the digital circuit simulator
- You now can runsome simulations to determine how your estimated wiring lengths will affect your digital circuit
- If the wire lengths are indeed overly affecting the circuit timing, the designer will need to modify either the floorplan or netlist

The floorplan/Timing Loop

Placement

- The exact positions of all the logic gates within each block is specified using a placement tool
- The placement software starts by selecting one block to work with and looks for components associated with the selected block
- The placement software continues placing logic gates based on their connectivity and the output files from the floorplanning tool
- The initial placement scheme can be considered just a first pass
- Device placement might require multiple passes based on the tool
- There are placement tools available that can make gate placement decisions based upon the signal timing of a design
- This placement approach is known astiming-driven layoutand has quickly become standard practice

I/O Drivers

- The I/O drivers are placed at the placement phase
- These I/O drivers are the special cells that will drive the input signals, provide outputs, contain protection and test circuitry
- These drivers are placed using separate placement tools that know about the I/O rules
- They place the I/O pads separately from the placement of the standard logic cells
- Finally, after all these automated tools and all the timing feedback looping, you have the best placement you think you can reasonably make

Routing

- With your gates and I/O cells nailed in place, you will now start to wire everything together
- The routing tool has two priority nets—power and clock signals
- It will route these two types of nets first since they are the most critical
- After the power rails and the clock signals have been placed, your wiring software will continue to wire the remainder of the circuitry, beginning with any other circuitry you declare as critical

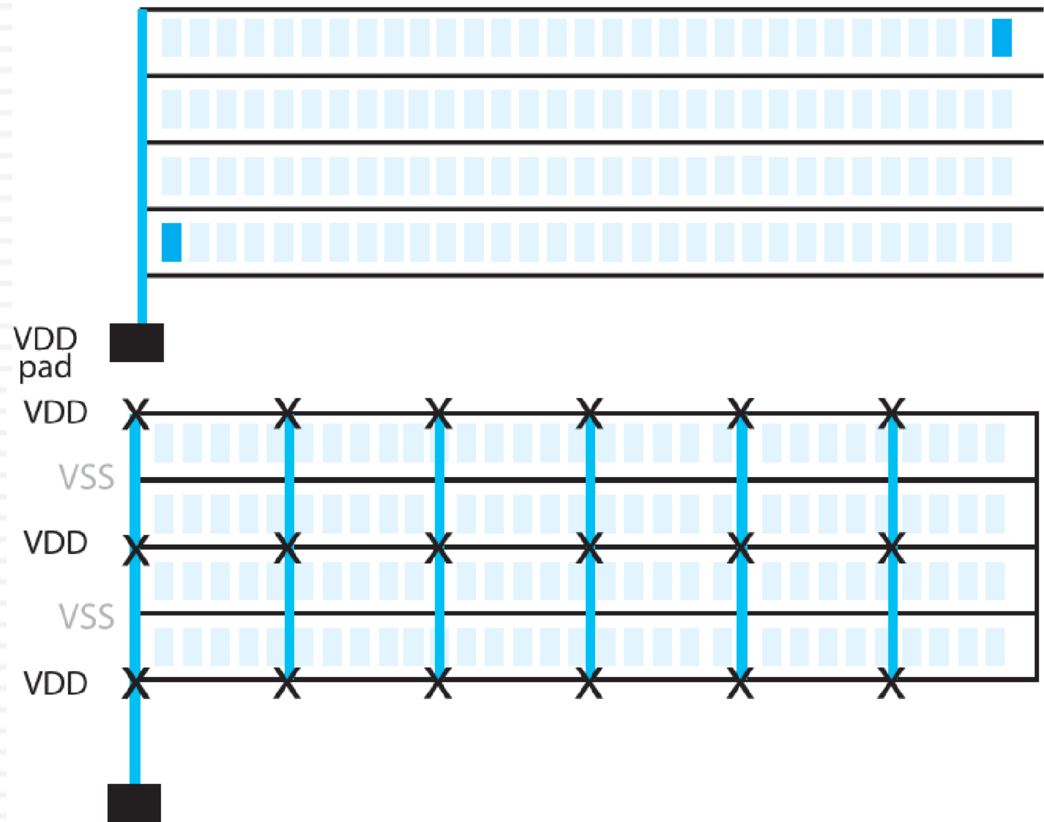
Power Nets

- There are certain rules for connecting power to logic gates
- Wiring must be centered in certain places and run in certain directions
- You end up with power rails running through the middle of your gates
- The wiring software is driven by the net list, which is aware of every component
- Therefore, it can tell you when it has completed wiring the power nets

Example: Power Rails Routing

Strapping

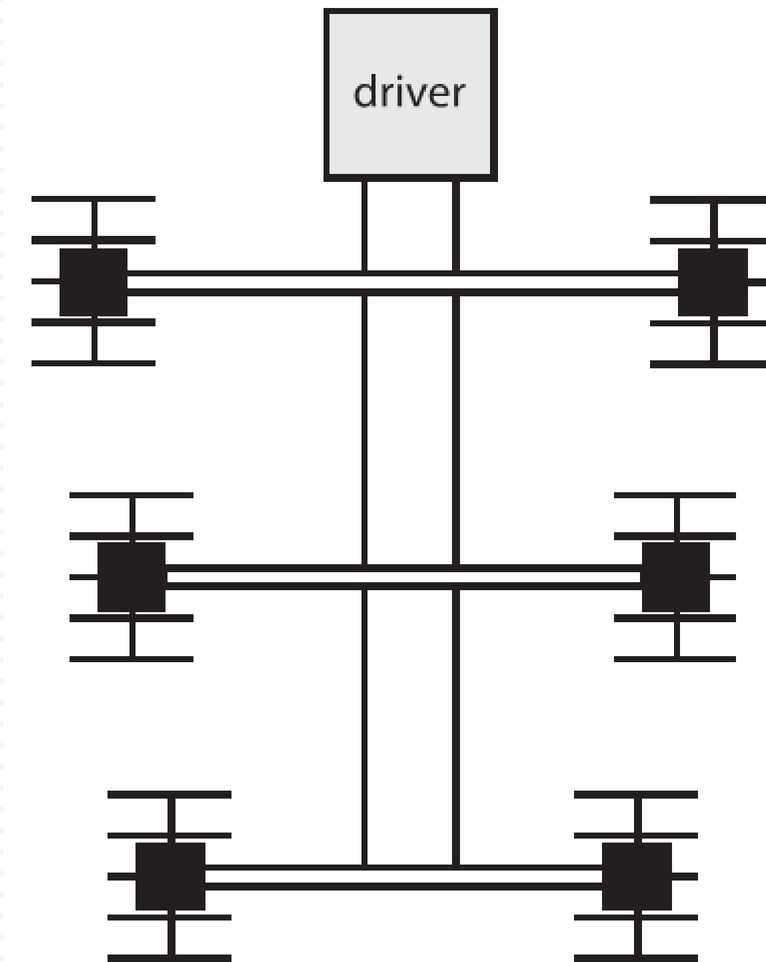
- In Figure, notice the highlighted cell at the far upper right corner, farthest from the VDD input pad.
- Distant cells see more resistance through the rails due to the length of wire
- By laying straps of metal across your power rails, you create a big waffle iron
- grid of multiple paths. Now it's like having resistors in parallel. The overall resistance reduces



Clock Net Wiring

- Once you have finished running the power rails, your software usually offers a specialized tool just to wire all the clock nets
- You can use the Central Clock Trunk Approach
- There is usually a clock driver cell that has enough drive strength to drive the top level clock buffers
- Place that cell centrally within your design and create a large central trunk that branches out to join to all the clock buffers.
- As the net reaches further out from the main driver, it continually splits into more and finer branches
- The wire widths at the outer edges become smaller and smaller

Central Clock Trunk Approach to wiring clock nets



Finishing the Wiring

- At this point, we turn to any other nets that need special attention
- You wire the critical nets that you are most worried about first, maybe by hand
- This process can be semi-automated under your guidance
- The last thing you do is wire the rest of the circuitry
- The tool will know how to automatically wire everything else on its own
- When the auto-router finishes as well as it can, you might end up with all the wiring hooked up on your chip or you might not
- Usually you will have to use the human eye, break some nets, and move stuff around, in order to complete the wiring

Prefabricated Gate Array Chips

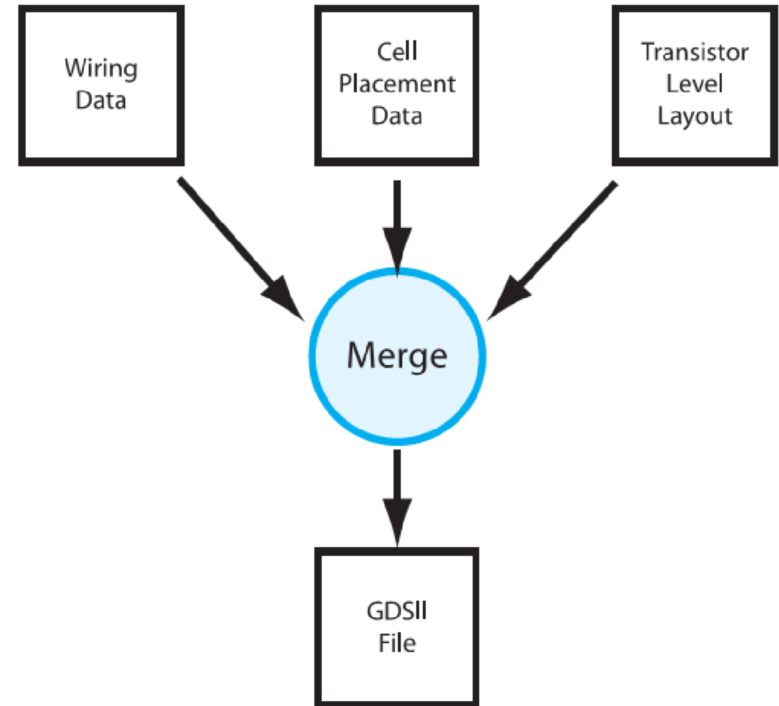
- All of the above techniques can be used on gate arrays such as FPGAs
- You will still use floorplanning tools, placement tools, and wiring tools
- However, you are not placing any diffusion or poly, only metalization and contact layers

Design Verification

- You feed your new wiring file back to the simulation people again which simulate with the actual wire data.
- No more estimation as the wiring is physically in place this time
- If the post place and route verification fails you need to redesign
- You might need to go back just one or two steps or you might need to start all over again
- Eventually, when all the timing is done, all the wiring is placed, and the chip has been re-simulated you will have a finished top-level layout of your chip
- You have converted a database from a conceptual format into a real mask design

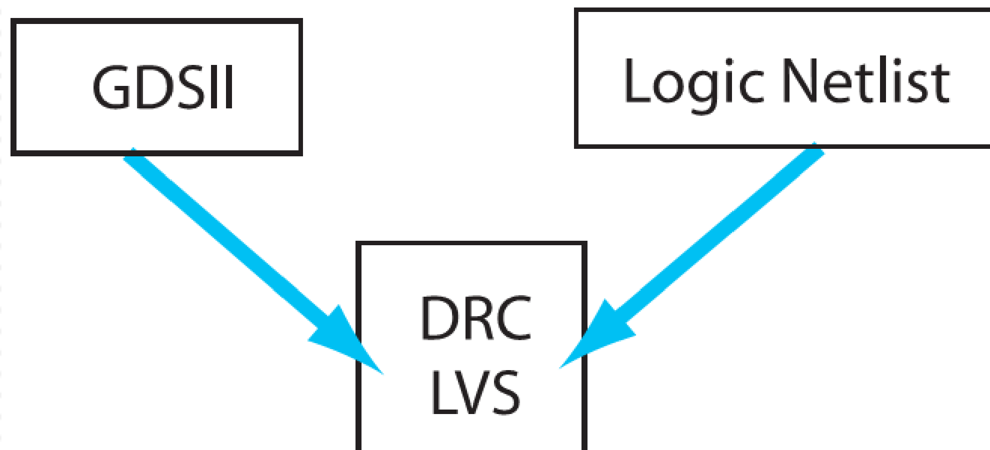
Physical Verification

- Up to this point, you have not been working with real transistors but only just been working with models
- To complete your mask design, you take this abstract, high-level database, and replace the boxes with the real logic gates
- You merge the real components from real libraries with the wiring and placement data from the place and route tools
- As you replace the abstract components with real library components, you produce a GDSII stream file of your chip
- This is a file that has all the components, all the wiring of your cells, all your vias, everything



DRC and LVS Checks

- Once you get your GDSII stream file, you then will want to run checks to be sure that the wiring is correct and complete
- At this point, we check all the process design rules using Design Rule Check (DRC) software
- We also check that the wiring and transistor connectivity correctly match the connectivity defined in our netlist
- We use Layout Versus Schematic (LVS) software to perform this connectivity check



Flowchart of Digital Layout Process