Digital VLSI Design

Lecture 3: Logic Synthesis Part 1

Semester A, 2018-19

Lecturer: Dr. Adam Teman

November 7, 2018

EnICS
Emerging Nanoscaled
Integrated Circuits and Systems Labs

Tradition of Excellence
Bar-Ilan University
אוניברסיטת בר-אילן

Disclaimer: This course was prepared, in its entirety, by Adam Teman. Many materials were copied from sources freely available on the internet. When possible, these sources have been cited; however, some references may have been cited incorrectly or overlooked. If you feel that a picture, graph, or code example has been copied from you and either needs to be cited or removed, please feel free to email adam.teman@biu.ac.il and I will address this as soon as possible.

Lecture Outline

1 Introduction | 2 Compilation | 3 Library Def. | 4 LEF | 5 Liberty | 6 Other

Introduction

...what is logic synthesis?

Syntax Analysis · Library Definition · Elaboration and Binding · Pre-mapping Optimization · Constraint Definition · Technology Mapping · Post-mapping Optimization · Report and export

EnICS Emerging Nanoscaled Integrated Circuits and Systems Labs · Bar-Ilan University אוניברסיטת בר-אילן

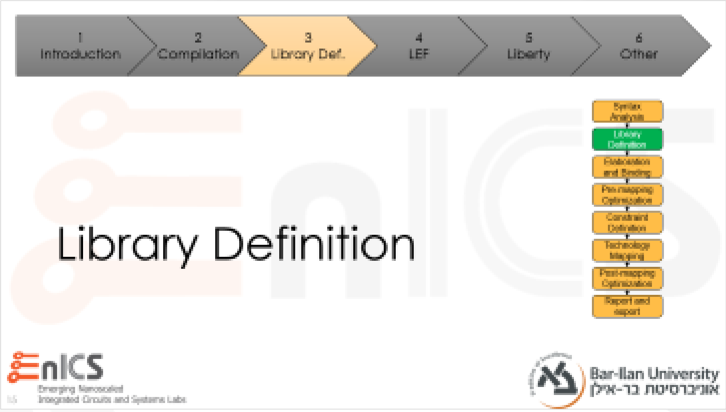
1 Introduction | 2 Compilation | 3 Library Def. | 4 LEF | 5 Liberty | 6 Other

Compilation

...but aren't we talking about synthesis?

Syntax Analysis · Library Definition · Elaboration and Binding · Pre-mapping Optimization · Constraint Definition · Technology Mapping · Post-mapping Optimization · Report and export

EnICS Emerging Nanoscaled Integrated Circuits and Systems Labs · Bar-Ilan University אוניברסיטת בר-אילן

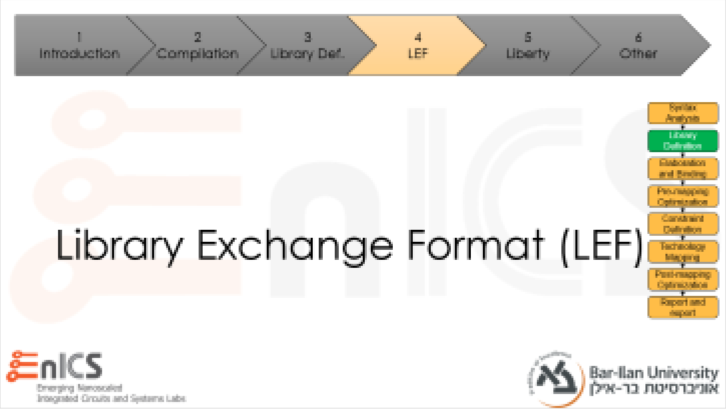


1 Introduction | 2 Compilation | 3 Library Def. | 4 LEF | 5 Liberty | 6 Other

Library Definition

Syntax Analysis · Library Definition · Elaboration and Binding · Pre-mapping Optimization · Constraint Definition · Technology Mapping · Post-mapping Optimization · Report and export

EnICS Emerging Nanoscaled Integrated Circuits and Systems Labs · Bar-Ilan University אוניברסיטת בר-אילן

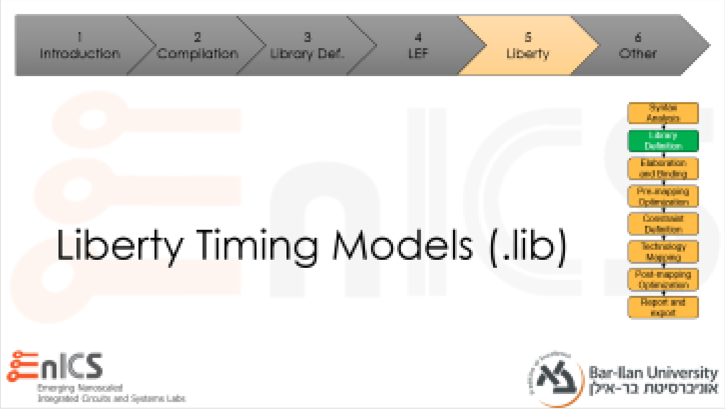


1 Introduction | 2 Compilation | 3 Library Def. | 4 LEF | 5 Liberty | 6 Other

Library Exchange Format (LEF)

Syntax Analysis · Library Definition · Elaboration and Binding · Pre-mapping Optimization · Constraint Definition · Technology Mapping · Post-mapping Optimization · Report and export

EnICS Emerging Nanoscaled Integrated Circuits and Systems Labs · Bar-Ilan University אוניברסיטת בר-אילן

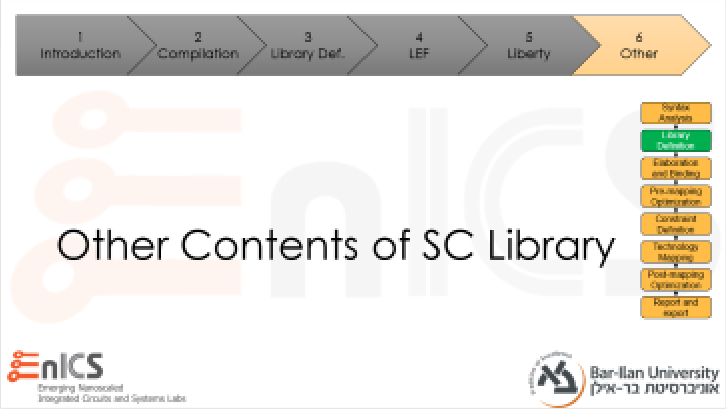


1 Introduction | 2 Compilation | 3 Library Def. | 4 LEF | 5 Liberty | 6 Other

Liberty Timing Models (.lib)

Syntax Analysis · Library Definition · Elaboration and Binding · Pre-mapping Optimization · Constraint Definition · Technology Mapping · Post-mapping Optimization · Report and export

EnICS Emerging Nanoscaled Integrated Circuits and Systems Labs · Bar-Ilan University אוניברסיטת בר-אילן



1 Introduction | 2 Compilation | 3 Library Def. | 4 LEF | 5 Liberty | 6 Other

Other Contents of SC Library

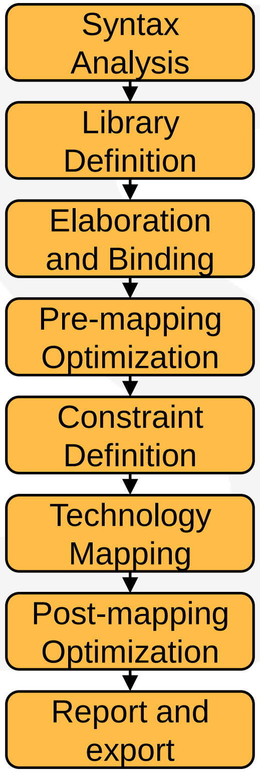
Syntax Analysis · Library Definition · Elaboration and Binding · Pre-mapping Optimization · Constraint Definition · Technology Mapping · Post-mapping Optimization · Report and export

EnICS Emerging Nanoscaled Integrated Circuits and Systems Labs · Bar-Ilan University אוניברסיטת בר-אילן

1 Introduction | 2 Compilation | 3 Library Def. | 4 LEF | 5 Liberty | 6 Other

Introduction

…what is logic synthesis?

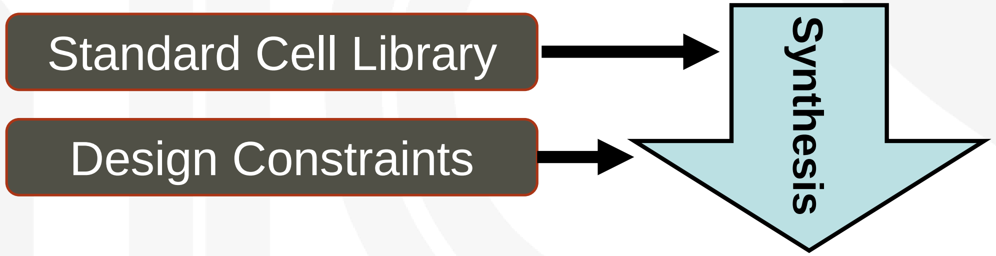


Syntax Analysis → Library Definition → Elaboration and Binding → Pre-mapping Optimization → Constraint Definition → Technology Mapping → Post-mapping Optimization → Report and export

What is Logic Synthesis?

- **Synthesis is the process that converts RTL into a technology-specific gate-level netlist, optimized for a set of pre-defined constraints.**
- **You start with:**
  - A behavioral RTL design
  - A standard cell library
  - A set of design constraints
- **You finish with:**
  - A gate-level netlist, mapped to the standard cell library
  - (For FPGAs: LUTs, flip-flops, and RAM blocks)
  - Hopefully, it’s also efficient in terms of speed, area, power, etc.

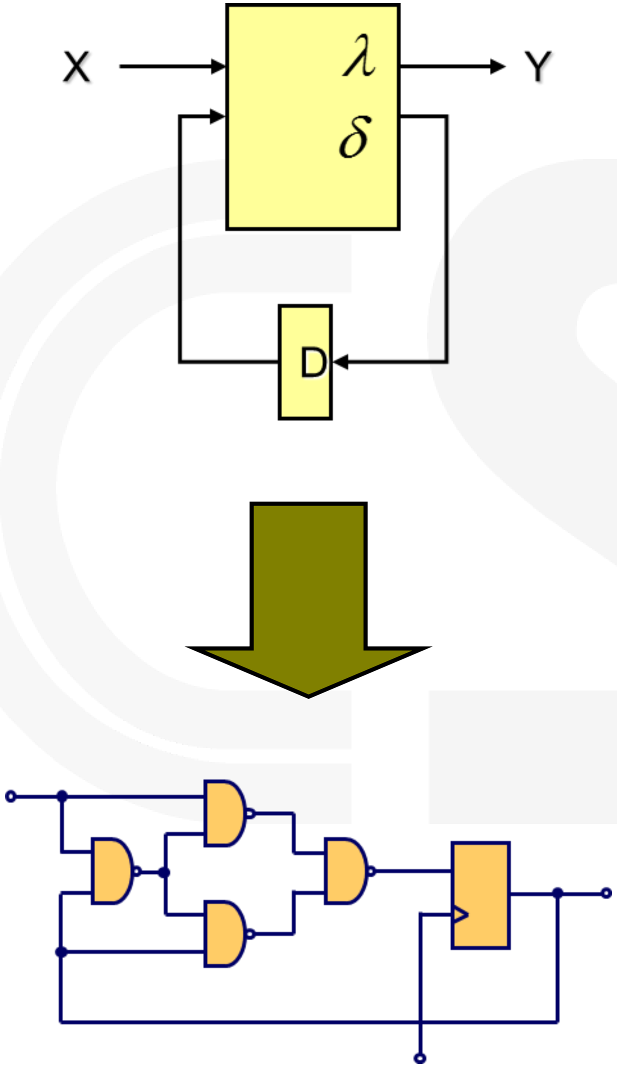
```
module counter(
     input clk, rstn, load,
     input  [1:0] in,
     output reg [1:0] out);
always @(posedge clk)
  if (!rstn) out <= 2'b0;
  else if (load) out <= in;
  else out <= out + 1;
endmodule
```



```
module counter ( clk, rstn, load, in, out );
  input [1:0] in;
  output [1:0] out;
  input clk, rstn, load;
  wire   N6, N7, n5, n6, n7, n8;

  FFPQ1 out_reg_1 (.D(N7),.CK(clk),.Q(out[1]));
  FFPQ1 out_reg_0 (.D(N6),.CK(clk),.Q(out[0]));
  NAN2D1 U8 (.A1(out[0]),.A2(n5),.Z(n8));
  NAN2D1 U9 (.A1(n5),.A2(n7),.Z(n6));
  INVD1 U10 (.A(load),.Z(n5));
  OA211D1 U11 (.A1(in[0]),.A2(n5),.B(rstn),.C(n8),.Z(N6));
  OA211D1 U12 (.A1(in[1]),.A2(n5),.B(rstn),.C(n6),.Z(N7));
  EXNOR2D1 U13 (.A1(out[1]),.A2(out[0]),.Z(n7));
endmodule
```

What is Logic Synthesis?

- **Given: Finite-State Machine $F(X, Y, Z, \lambda, \delta)$ where:**
  - X: Input alphabet
  - Y: Output alphabet
  - Z: Set of internal states
  - λ: $X \times Z \rightarrow Z$ (next state function)
  - δ: $X \times Z \rightarrow Y$ (output function)



- **Target: Circuit $C(G, W)$ where:**
  - G: set of circuit components
    $G = \{\text{Boolean gates, flip-flops, etc.}\}$
  - W: set of wires connecting G

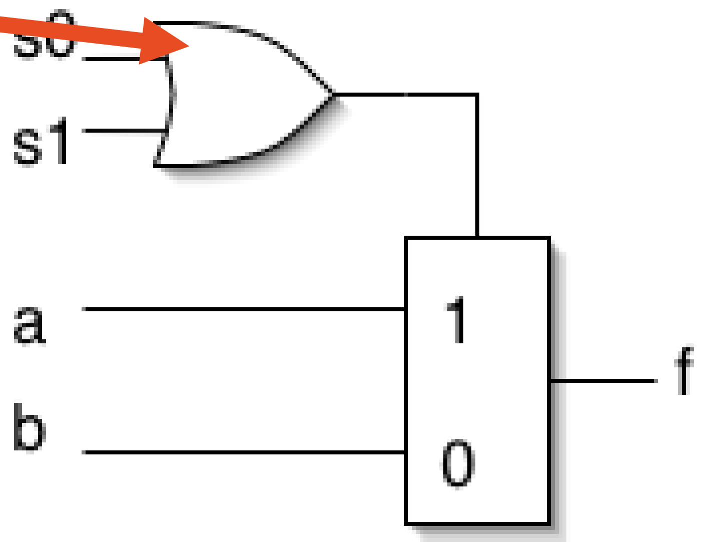
Motivation

- **Why perform logic synthesis?**
  - Automatically manages many details of the design process:
    - Fewer bugs
    - Improves productivity
    - Abstracts the design data (HDL description) from any particular implementation technology
    - Designs can be re-synthesized targeting different chip technologies;
      - E.g.: first implement in FPGA then later in ASIC
  - In some cases, leads to a more optimal design than could be achieved by manual means (e.g.: logic optimization)

- **Why not logic synthesis?**
  - May lead to less than optimal designs in some cases

Simple Example

```
module foo (a,b,s0,s1,f);
 input [3:0] a;
 input [3:0] b;
 input s0,s1;
 output [3:0] f;
 reg f;

 always @ (a or b or s0 or s1)
   if (!s0 && s1 || s0)
       f=a;
   else
       f=b;
endmodule
```



Goals of Logic Synthesis

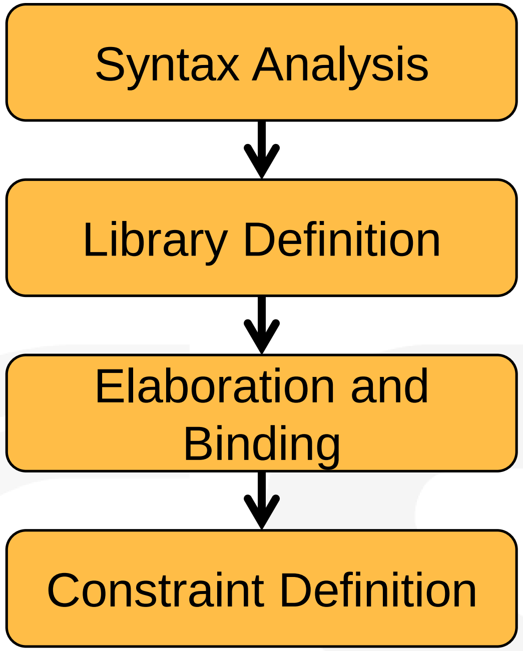
- **Minimize area**
  - In terms of literal count, cell count, register count, etc.
- **Minimize power**
  - In terms of switching activity in individual gates, deactivated circuit blocks, etc.
- **Maximize performance**
  - In terms of maximal clock frequency of synchronous systems, throughput for asynchronous systems
- **Any combination of the above**
  - Combined with different weights
  - Formulated as a constraint problem
    - "Minimize area for a clock speed > 300MHz"
- **More global objectives**
  - Feedback from layout
    - Actual physical sizes, delays, placement and routing

How does it work?

Variety of general and ad-hoc (special case) methods:

- **Instantiation:**
  - Maintains a library of primitive modules (`AND`, `OR`, etc.) and user defined modules
- **“Macro expansion”/substitution:**
  - A large set of language operators (`+`, `-`, Boolean operators, etc.) and constructs (`if-else`, `case`) expand into special circuits
- **Inference:**
  - Special patterns are detected in the language description and treated specially (e.g.,: inferring memory blocks from variable declaration and read/write statements, FSM detection and generation from `always@(posedge clk)` blocks)
- **Logic optimization:**
  - Boolean operations are grouped and optimized with logic minimization techniques
- **Structural reorganization:**
  - Advanced techniques including sharing of operators, and retiming of circuits (moving FFs), and others

Basic Synthesis Flow



- **Syntax Analysis:**
  - Read in HDL files and check for syntax errors.

```
read_hdl –verilog sourceCode/toplevel.v
```

- **Library Definition:**
  - Provide standard cells and IP Libraries.

```
read_libs “/design/data/my_fab/digital/lib/TT1V25C.lib”
```

- **Elaboration and Binding:**
  - Convert RTL into Boolean structure.
  - State reduction, encoding, register infering.
  - Bind all leaf cells to provided libraries.

```
elaborate toplevel
```

- **Constraint Definition:**
  - Define clock frequency and other design constraints.

```
read_sdc sdc/constraints.sdc
```

Basic Synthesis Flow

- **Pre-mapping Optimization:**
  - Map to generic cells and perform additional heuristics.

```
syn_generic
```

- **Technology Mapping:**
  - Map generic logic to technology libraries.

```
syn_map
```

- **Post-mapping Optimization:**
  - Iterate over design, changing gate sizes, Boolean literals, architectural approaches to try and meet constraints.

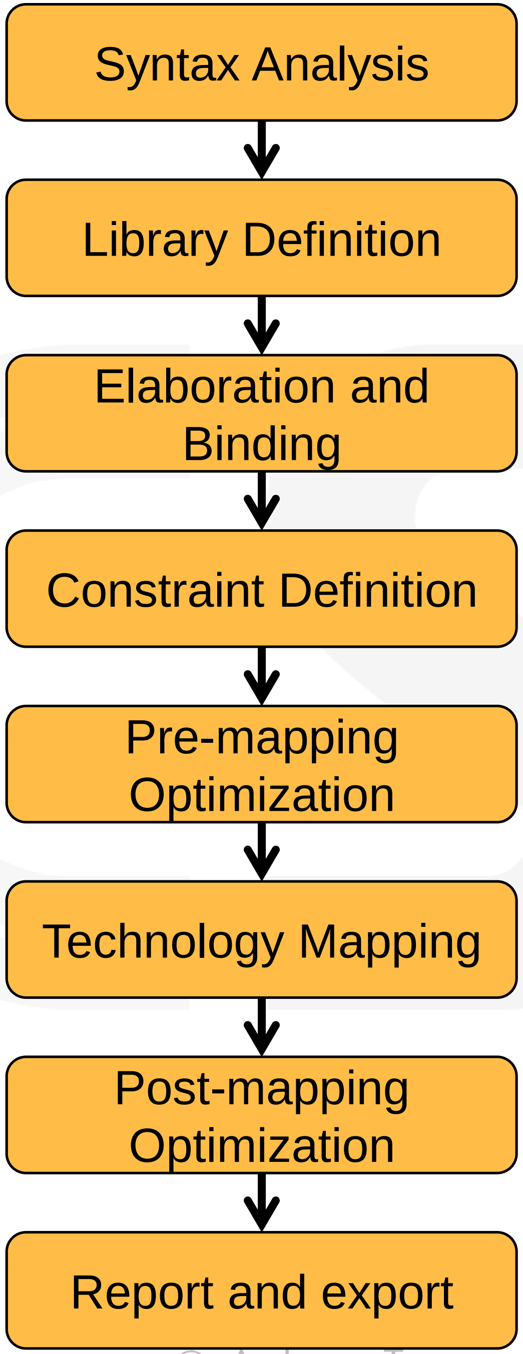
```
syn_opt
```

- **Report and export**
  - Report final results with an emphasis on timing reports.

```
report timing –num paths 10 > reports/timing_reports.rpt
```

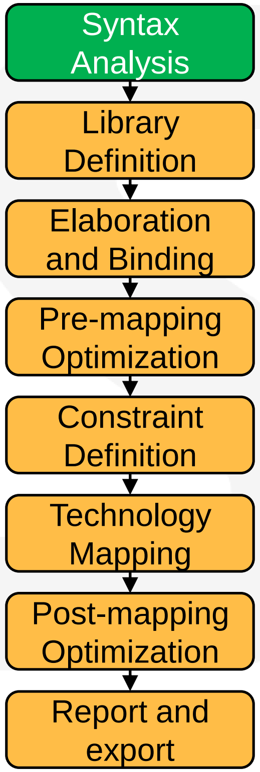
  - Export netlist and other results for further use.

```
write_hdl > export/netlist.v
```

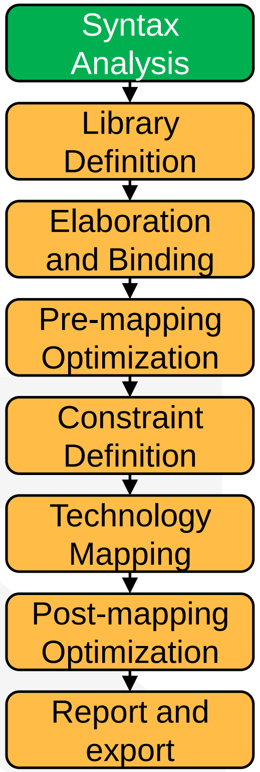


Compilation

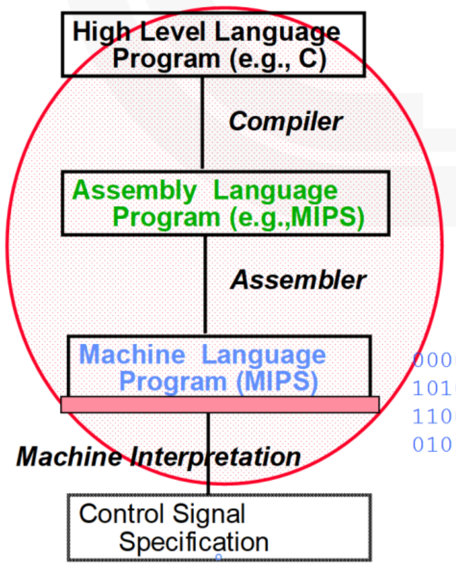
…but aren’t we talking about synthesis?



Compilation in the synthesis flow



- **Before starting to synthesize, we need to check the syntax for correctness.**
- **Synthesis vs. Compilation:**
  - Compiler
    - Recognizes all possible constructs in a formally defined program language
    - Translates them to a machine language representation of execution process
  - Synthesis
    - Recognizes a target dependent subset of a hardware description language
    - Maps to collection of concrete hardware resources
    - Iterative tool in the design flow


High Level Language Program (e.g., C)
Compiler
Assembly Language Program (e.g.,MIPS)
Assembler
Machine Language Program (MIPS)
Machine Interpretation
Control Signal Specification

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw $to,   0($2)
lw $t1,   4($2)
sw$t1,    0($2)
sw$t0,    4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

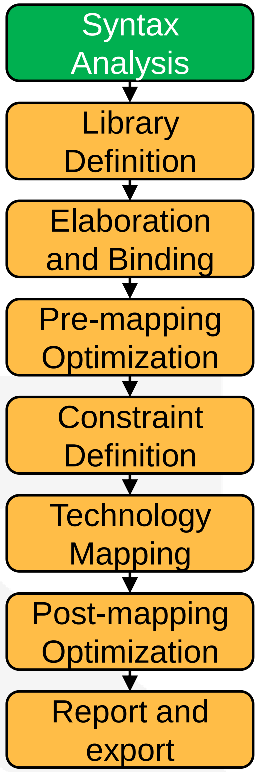
Compilation with NC-Verilog

- **To compile your Verilog code for syntax checking, use the NC-Verilog tool:**

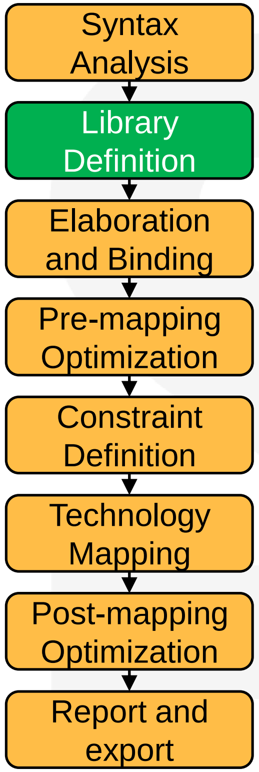
```
ncvlog <filename.v>
```

  - This will quickly run compilation on your Verilog source code and point you to syntax errors.
  - Alternatively, use the `irun` super command:

```
irun -compile <filename.v>
```



Library Definition


Syntax Analysis

Library Definition

Elaboration and Binding

Pre-mapping Optimization

Constraint Definition

Technology Mapping

Post-mapping Optimization

Report and export

It’s all about the standard cells…

- **The library definition stage tells the synthesizer where to look for *leaf cells* for binding and the *target library* for technology mapping.**
  - We can provide a list of *paths* to search for libraries in:

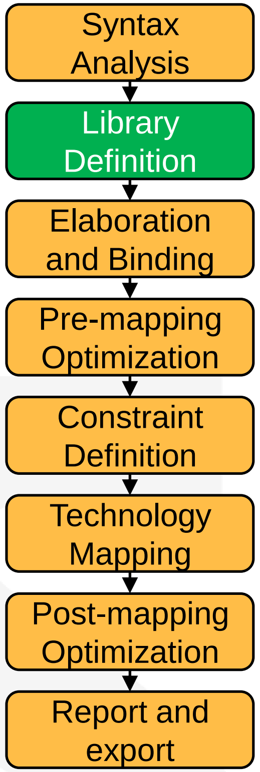
```
set_db init_lib_search_path “/design/data/my_fab/digital/lib/”
```

  - And we have to provide the name of a specific library, usually characterized for a single corner:

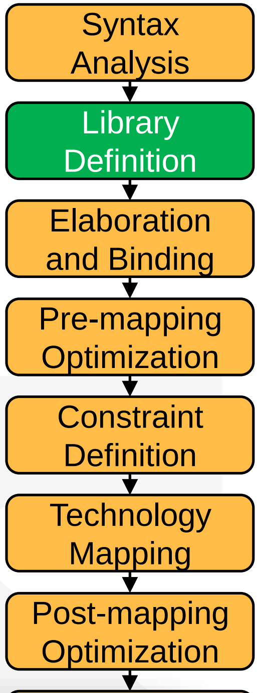
```
read_libs “TT1V25C.lib”
```

- **We also need to provide `.lib` files for IPs, such as memory macros, I/Os, and others.**

Make sure you understand all the warnings about the libs that the synthesizer spits out, even though you probably can’t fix them.


Syntax Analysis
Library Definition
Elaboration and Binding
Pre-mapping Optimization
Constraint Definition
Technology Mapping
Post-mapping Optimization
Report and export

But what is a library?

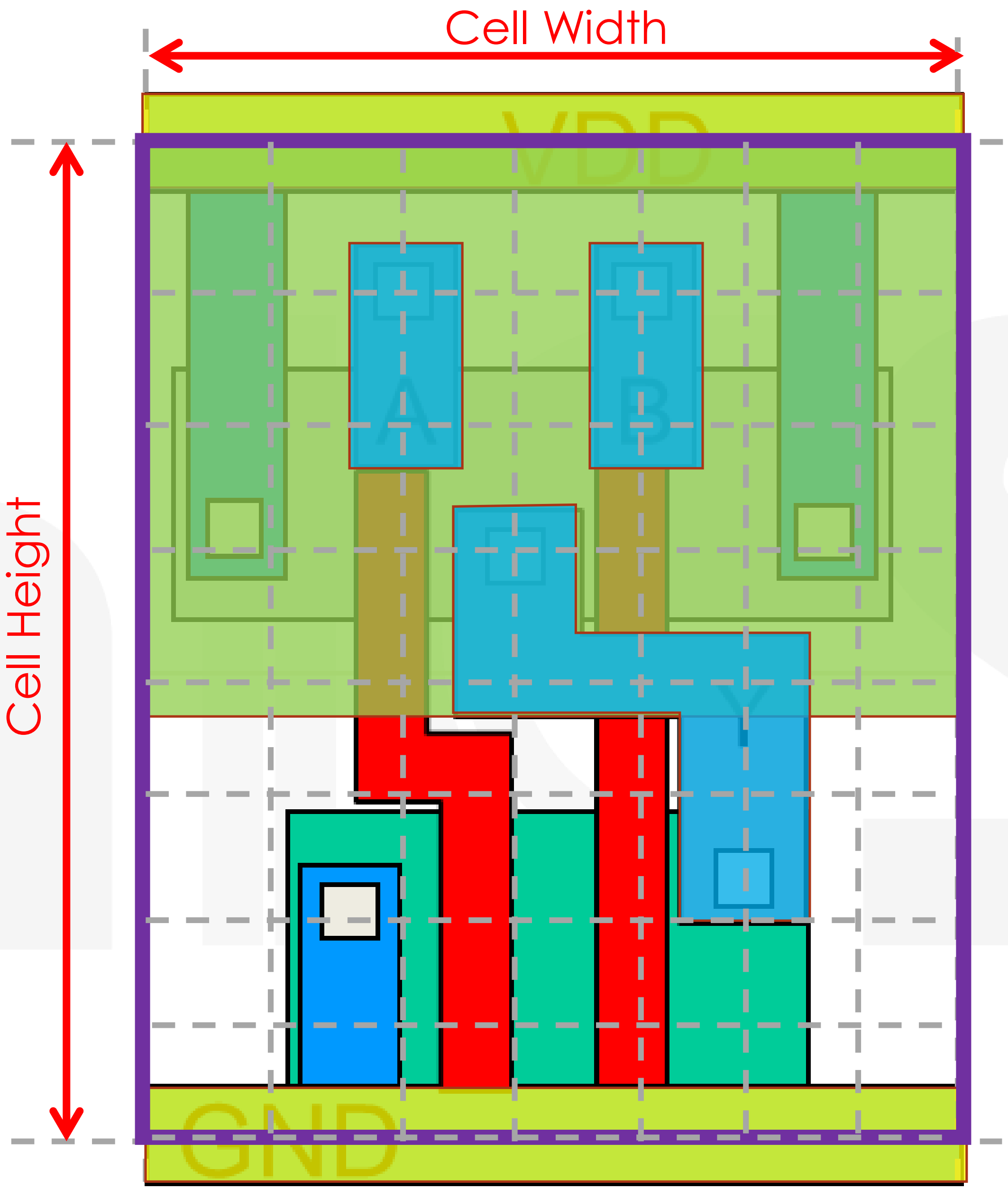


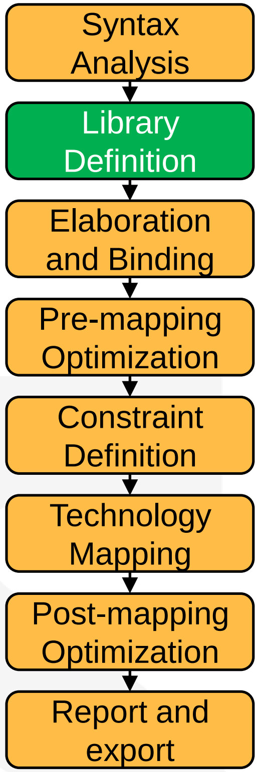
- A standard cell library is a collection of well defined and appropriately characterized logic gates that can be used to implement a digital design.
- Similar to LEGO, standard cells must meet predefined specifications to be flawlessly manipulated by synthesis, place, and route algorithms.
- Therefore, a standard cell library is delivered with a collection of files that provide all the information needed by the various EDA tools.

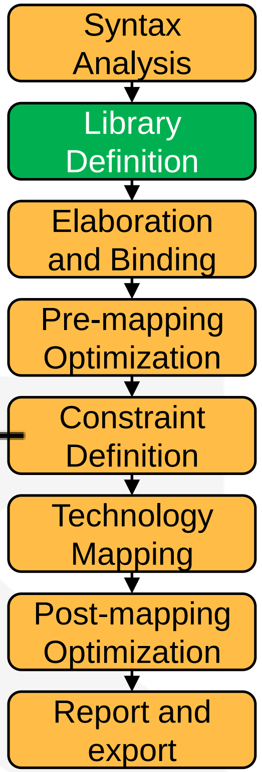
Example

- **NAND standard cell layout**
- **Pay attention to:**
  - Cell height
  - Cell width
  - Voltage rails
  - Well definition
  - Pin Placement
  - PR Boundary
  - Metal layers

Ideally, Standard Cells should be routed entirely in M1 !

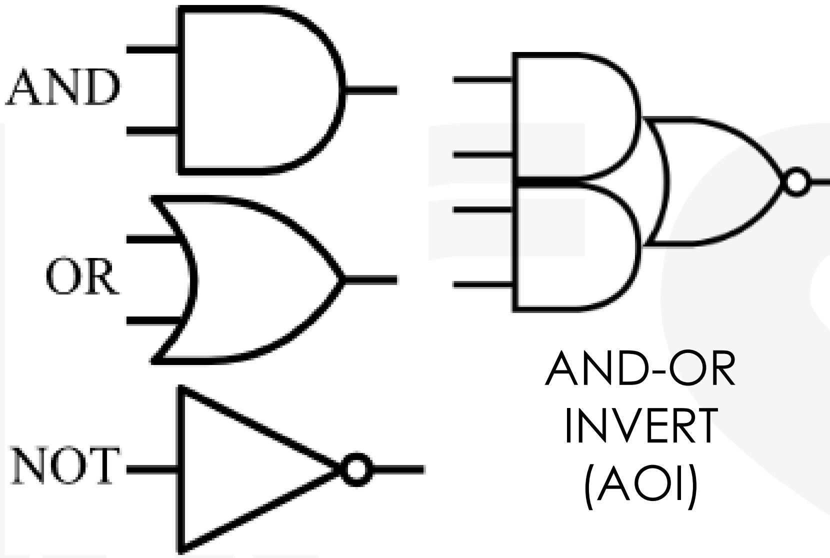


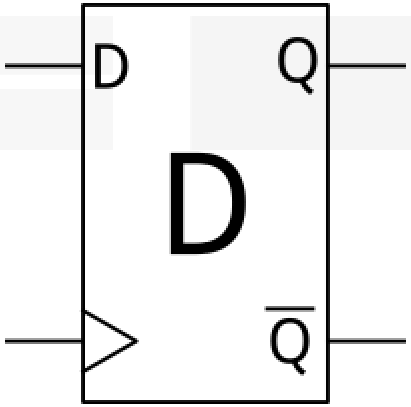




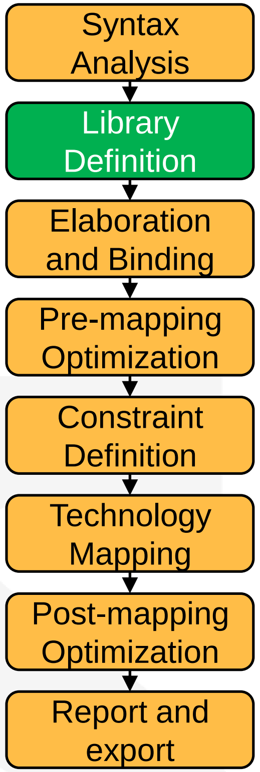
What cells are in a standard cell library?

- **Combinational logic cells (NAND, NOR, INV, etc.):**
  - Variety of drive strengths for all cells.
  - Complex cells (AOI, OAI, etc.)
  - Fan-In <= 4
  - ECO Cells
- **Buffers/Inverters**
  - Larger variety of drive strengths.
  - “Clock cells” with balanced rise and fall delays.
  - Delay cells
  - Level Shifters
- **Sequential Cells:**
  - Many types of flip flops: pos/negedge, set/reset, Q/QB, enable
  - Latches
  - Integrated Clock Gating cells
  - Scan enabled cells for ATPG.
- **Physical Cells:**
  - Fillers, Tap cells, Antennas, DeCaps, EndCaps, Tie Cells

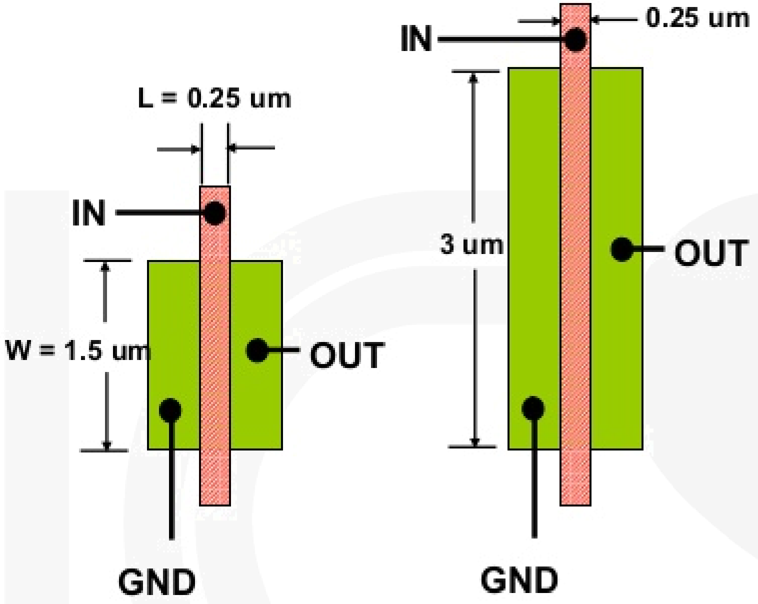




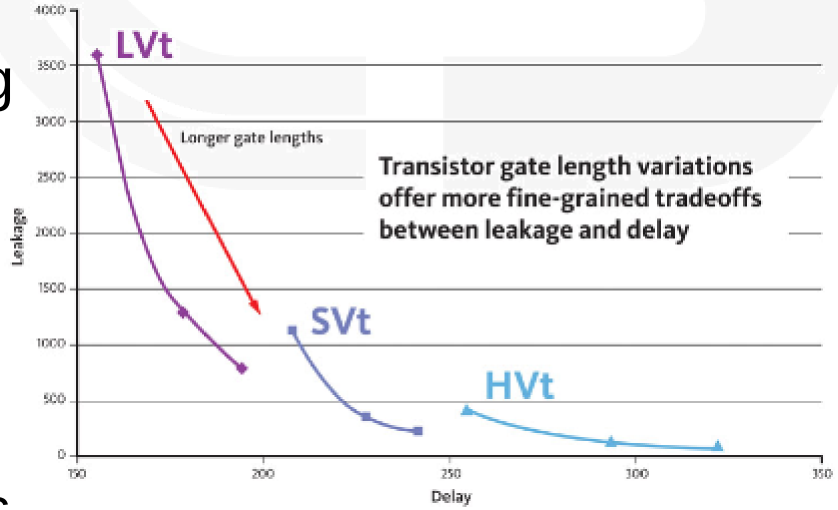
Multiple Drive Strengths and VTs



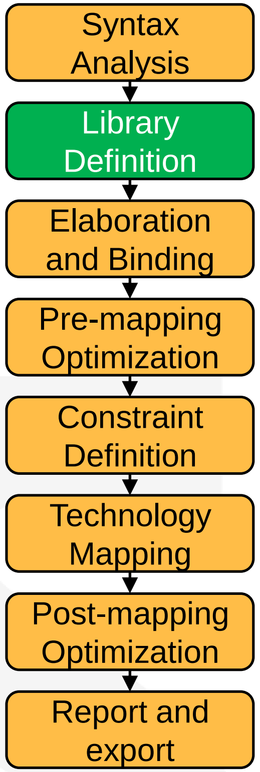
- **Multiple Drive Strength**
  - Each cell will have various sized output stages.
  - Larger output stage → better at driving fanouts/loads.
  - Smaller drive strength → less area, leakage, input cap.
  - Often called **X2**, **X3**, or **D2**, **D3**, etc.
- **Multiple Threshold (MT-CMOS)**
  - A single additional mask can provide more or less doping in a transistor channel, shifting the threshold voltage.
  - Most libraries provide equivalent cells with three or more VTs: **SVT**, **HVT**, **LVT**
    This enables tradeoff between speed vs. leakage.
  - All threshold varieties have same footprint and therefore can be swapped without any placement/routing iterations.



"1X" NMOS (W/L = 6) *"2X" NMOS (W/L = 12)*



Clock Cells


Syntax Analysis

Library Definition

Elaboration and Binding

Pre-mapping Optimization

Constraint Definition

Technology Mapping

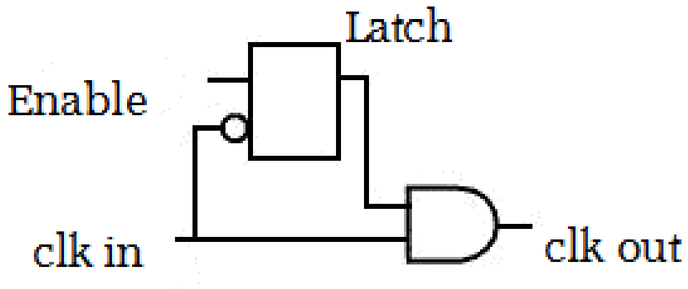
Post-mapping Optimization

Report and export

- **General standard cells are optimized for speed.**
  - That doesn't mean they're balanced…

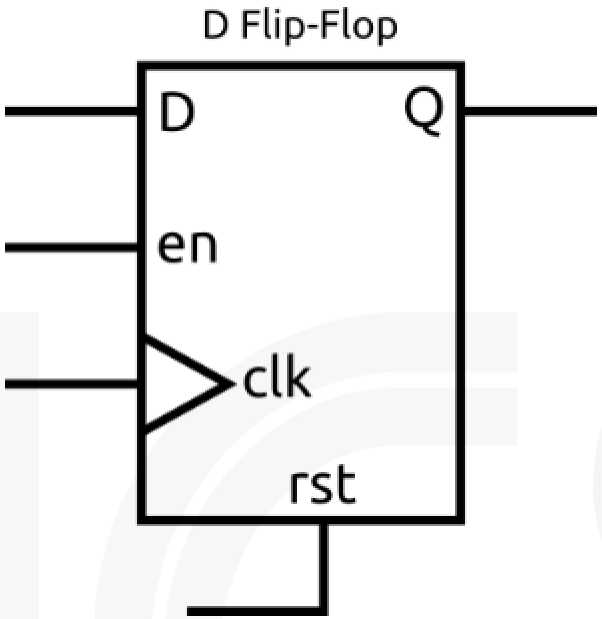
$$\min t_{pd} = \min\left(\frac{t_{p,LH} + t_{p,HL}}{2}\right) \not\Rightarrow t_{p,LH} = t_{p,HL}$$

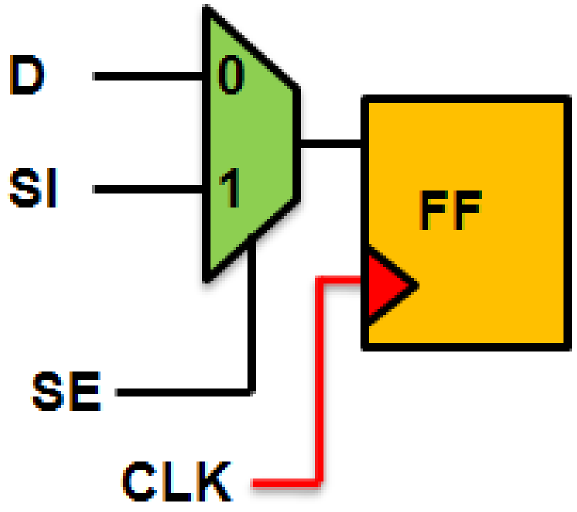
- **This isn't good for clock nets…**
  - Unbalanced rising/falling delays will result in unwanted skew.
  - Special "clock cells" are designed with balanced rising/falling delays to minimize skew.
  - These cells are usually less optimal for data and so should not be used.
- **In general, only buffers/inverters should be used on clock nets**
  - But sometimes, we need gating logic.
  - Special cells, such as *integrated clock gates*, provide logic for the clock networks.

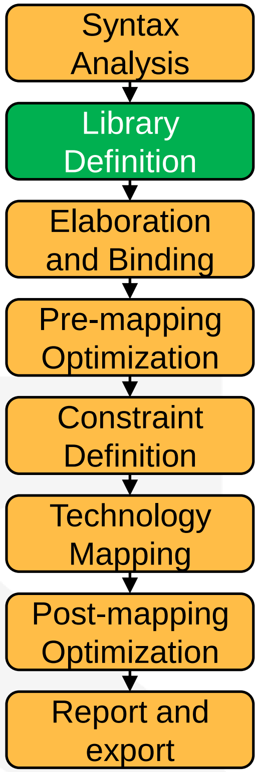


Sequentials

- **Flip Flops and Latches, including**
  - Positive/Negative Edge Triggered
  - Synchronous/Asynchronous Reset/Set
  - Q/QB Outputs
  - Enable
  - Scan
  - etc., etc.


D Flip-Flop




Syntax Analysis

Library Definition

Elaboration and Binding

Pre-mapping Optimization

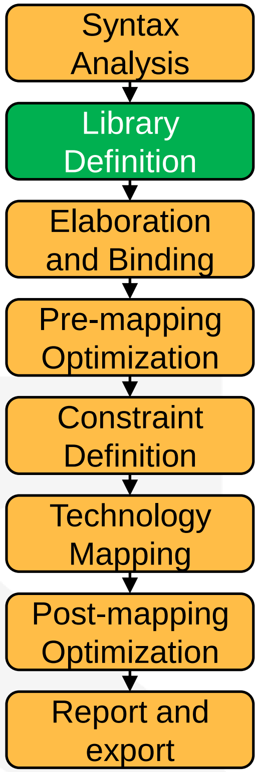
Constraint Definition

Technology Mapping

Post-mapping Optimization

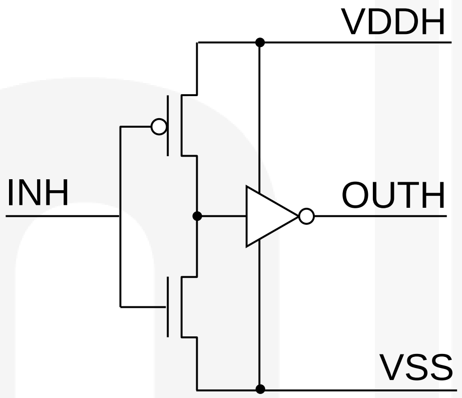
Report and export

Level Shifters

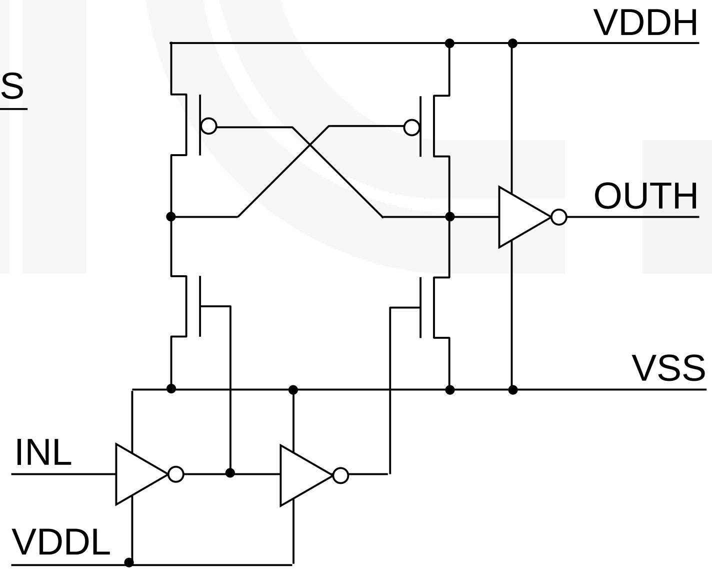


- **Level shifter cells are placed between voltage domains to pass signals from one voltage to another.**

- **HL (high-to-low) shifter**
  - Requires only one voltage
  - Single height cell

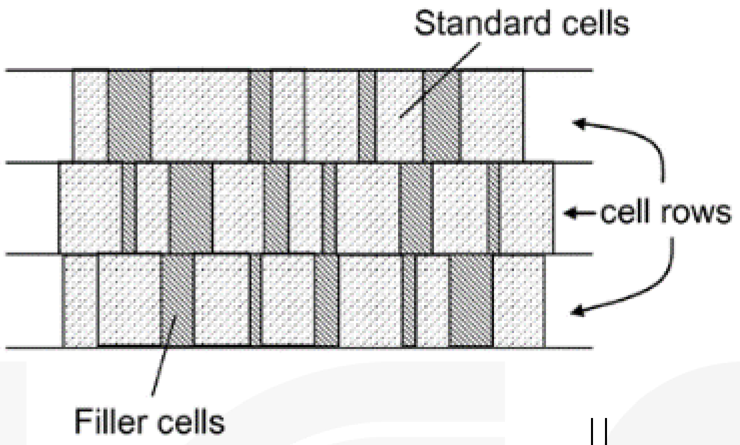


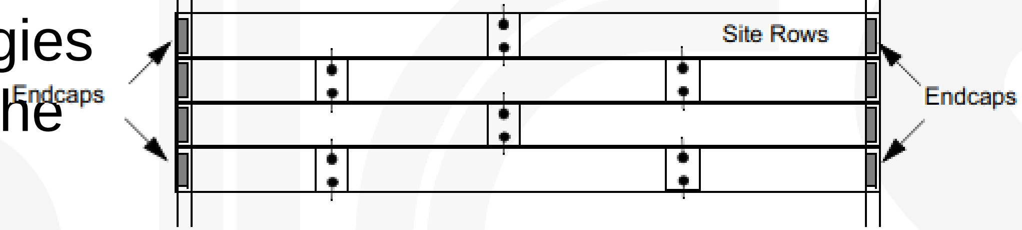
- **LH (low-to-high) shifter**
  - Needs 2 voltages
  - Often double height

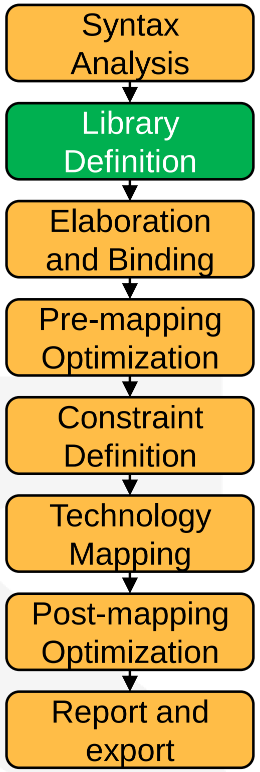


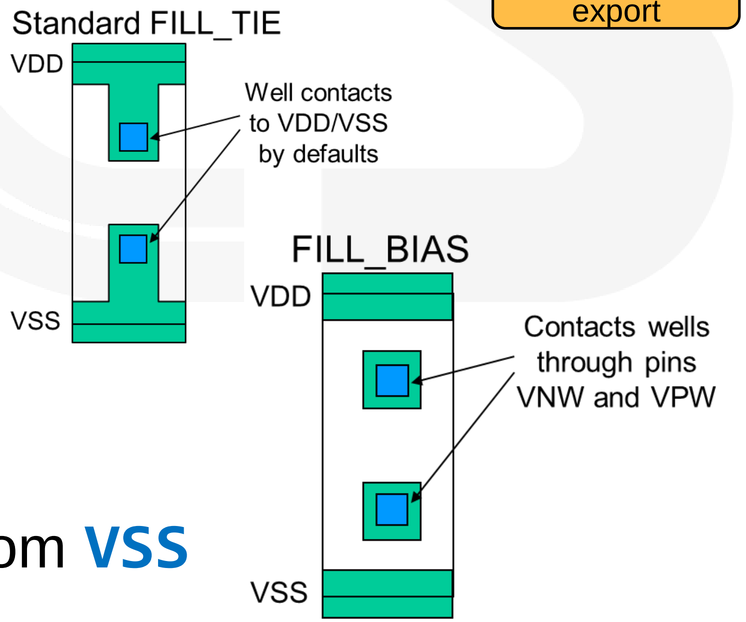
Filler and Tap Cells

- **Filler cells Must be inserted in empty areas in rows**
  - Ensure well and diffusion mask continuity
  - Ensure density rules on bottom layers
  - Provide dummy poly for scaled technologies
  - Sometimes, special cells are needed at the boundaries of rows - "End Caps"
  - Other fillers may include MOSCAPs between **VDD** and **GND** for voltage stability - "DeCAP cells"
- **Well Taps needed to ensure local body voltage**
  - Eliminate latch-up
  - No need to tap every single cell
- **Back or forward biasing for performance/leakage optimization**
  - **N-well** voltage different from **VDD**
  - Substrate or **P-well** (triple well process) voltage different from **VSS**
  - Bias voltage routed as signal pin or special power net








Standard FILL_TIE
VDD
VSS
Well contacts to VDD/VSS by defaults
FILL_BIAS
VDD
VSS
Contacts wells through pins VNW and VPW

Syntax Analysis

Library Definition

Elaboration

ing

ping tion

int on

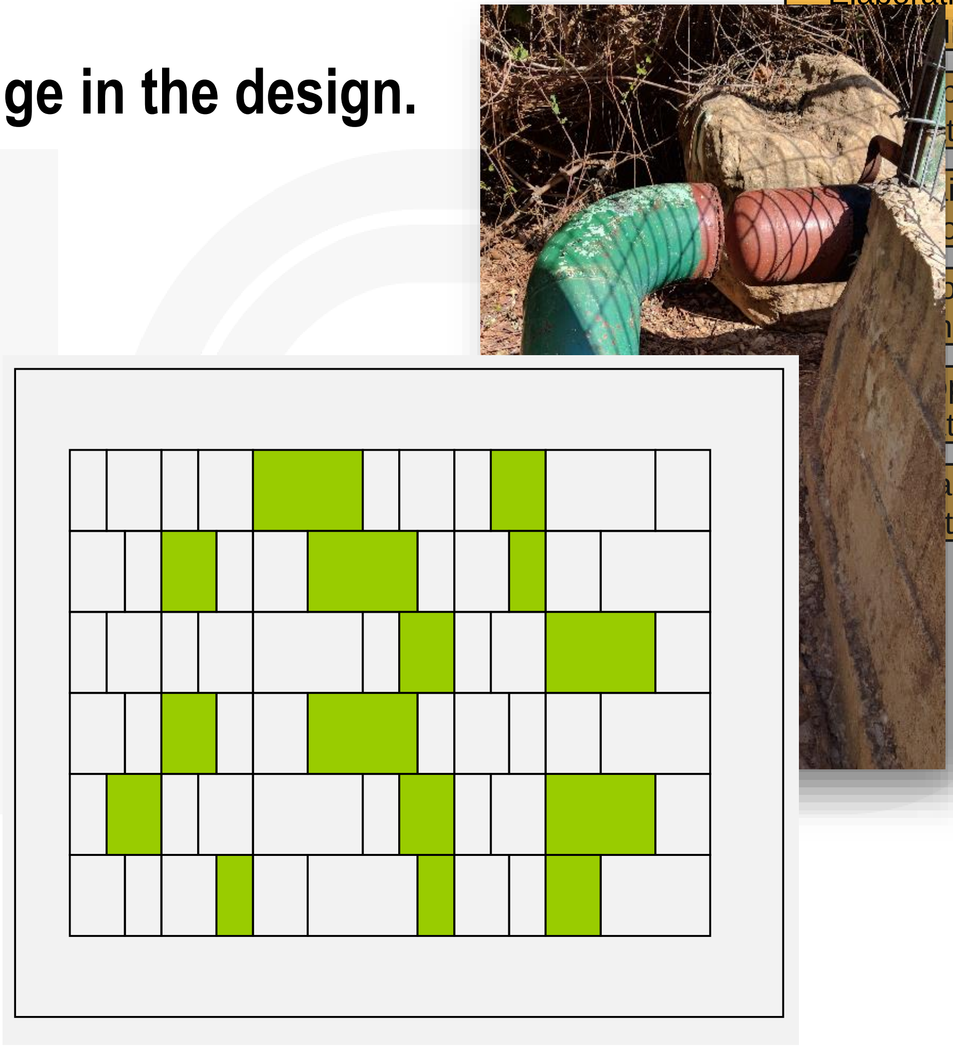
ogy ng

ping tion

nd t

Engineering Change Order (ECO) Cells

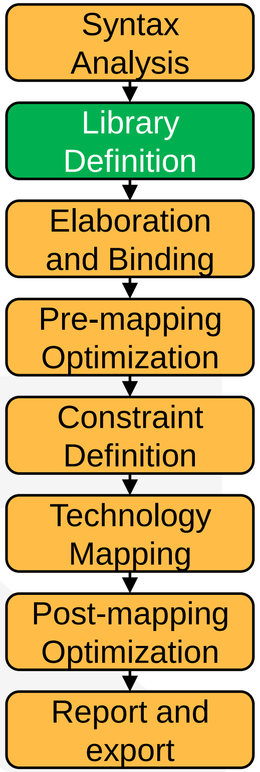
- **An Engineering Change Order (ECO) is a very late change in the design.**
  - ECOs usually are done after place and route.
  - However, re-spins of a chip are often done without recreating all-masks. This is known as a “Metal-Fix”.
- **ECOs usually require small changes in logic.**
  - How can we do this after placement?
  - Or worse – after tapeout???
- **Solution – Spare (Bonus) Cells!**
  - Cells without functionality
  - Cells are added during design (fill)
  - In case of problems (after processing) new metal and via mask → cells get their wanted functionality
  - Cell combinations can create more complex functions
    - Ex. **AND**,**NAND**,**NOR**,**XOR**,**FF**,**MUX**,**INV**,..



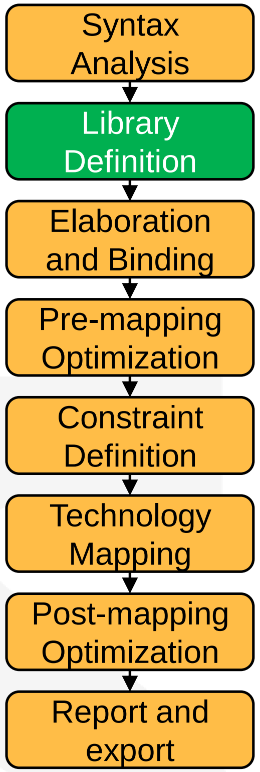
- **Special standard cells are used to differentiate from real cells.**

My favorite word… ABSTRACTION!

- **So, what is a cell?**
  - I guess that the detailed layout is sufficient to know (guess) anything and everything about a standard cell.
  - Or it would be easier, if we got the whole Open Access database of the cell…
- **But do we really need to know everything?**
  - For example, does logic simulation need to know if your inverter is CMOS or Pseudo-NMOS?
  - And does a logic synthesizer need to know what type of transistors you used?
- **No!**
  - To make life (and calculations) simpler, we will *abstract away* this info.
  - Each tool will get only the data it really needs.



What files are in a standard cell library?


Syntax Analysis

Library Definition

Elaboration and Binding

Pre-mapping Optimization

Constraint Definition

Technology Mapping

Post-mapping Optimization

Report and export

- **Behavioral Views:**
  - `Verilog` (or `Vital`) description used for simulation, logic equivalence.
- **Physical Views:**
  - Layout of the cells (`GDSII` format) for DRC, LVS, Custom Layout.
  - Abstract of the cells (`LEF` format) for P&R, RC extraction.
- **Transistor Level:**
  - `Spice`/`Spectre` netlist for LVS, transistor-level simulation.
  - Often provided both with parasitics (post-layout) and without.
- **Timing/Power:**
  - `Liberty` files with characterization of timing and power for STA.
- **Power Grid Views:**
  - Needed for IR Drop analysis.
- **Others:**
  - Symbols for displaying the cells in various tools.
  - `OA Libraries` for easy integration with Virtuoso.

Behavioral (.v)

Abstract (.lef)

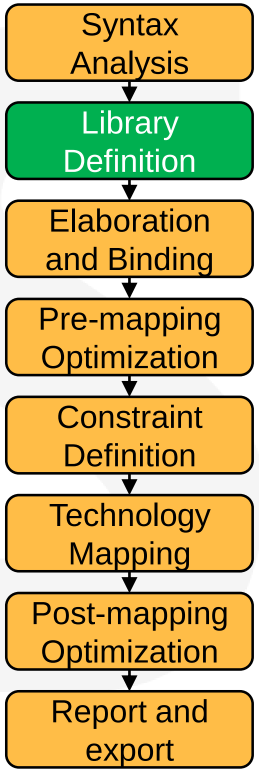
Layout (.gds)

Spice (.spi, .cdl)

Timing (.lib)

Open Access (.oa)

Library Exchange Format (LEF)



Syntax Analysis

Library Definition

Elaboration and Binding

Pre-mapping Optimization

Constraint Definition

Technology Mapping

Post-mapping Optimization

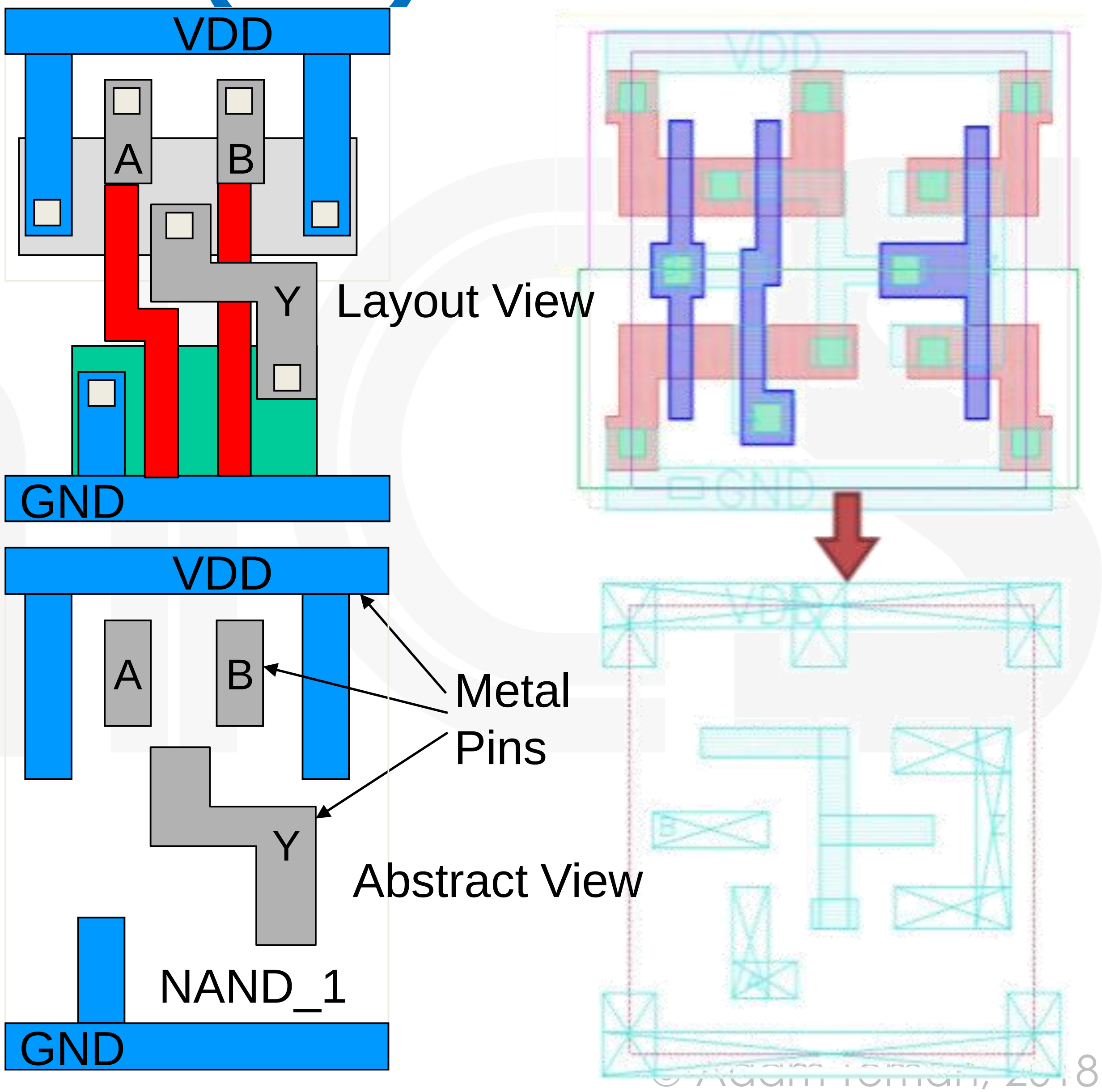
Report and export

Library Exchange Format (LEF)

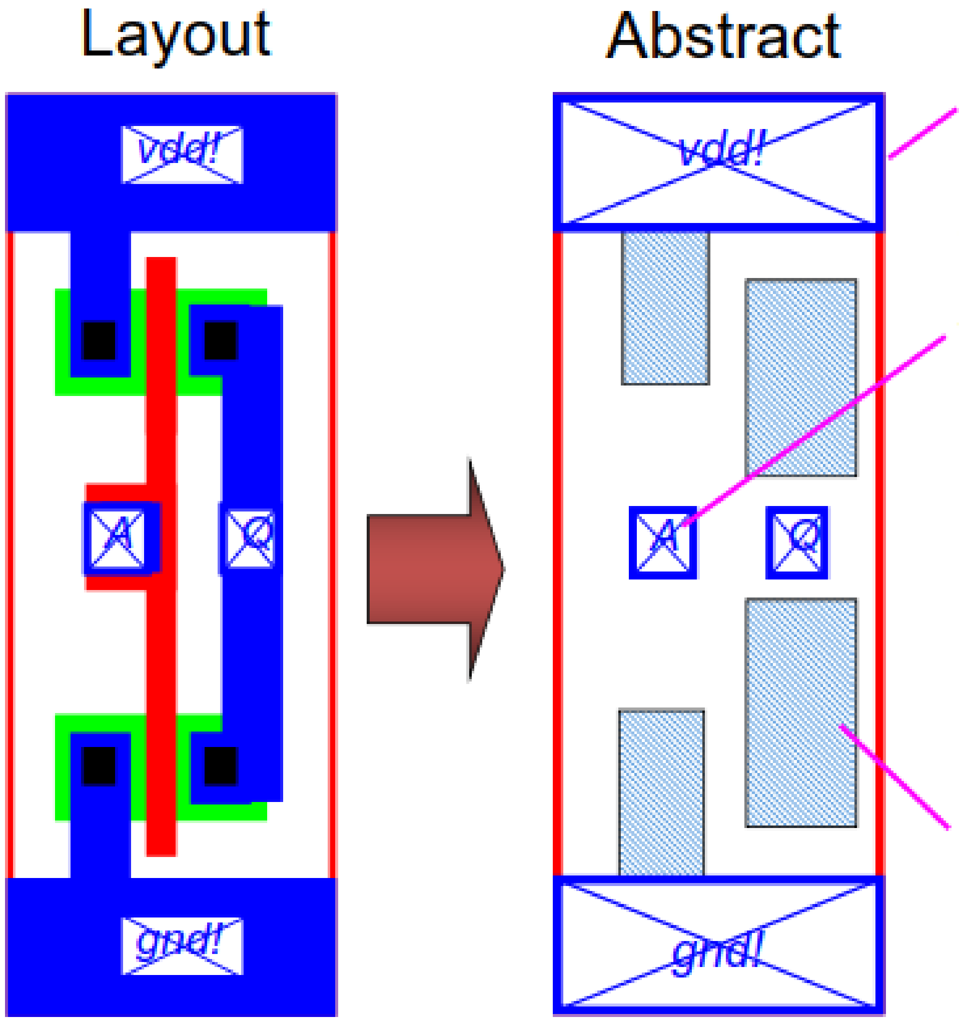
- **Abstract description of the layout for P&R**
  - Readable ASCII Format.
  - Contains detailed PIN information for connecting.
  - **Does not include** front-end of the line (poly, diffusion, etc.) data.

- **Abstract views only contain the following:**
  - **Outline** of the cell (size and shape)
  - **Pin** locations and layer (usually on `M1`)
  - Metal **blockages**

    (Areas in a cell where metal of a certain layer is being used, but is not a pin)



Library Exchange Format (LEF)



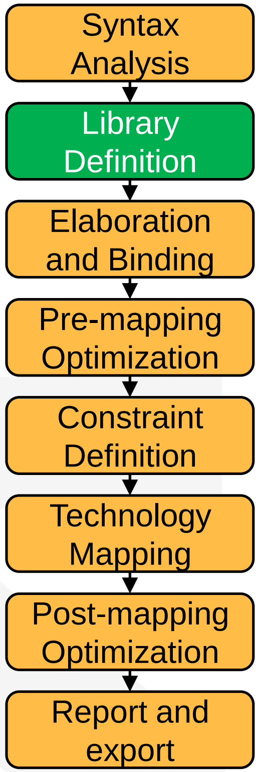
Physical cell size

Terminals with physical placement

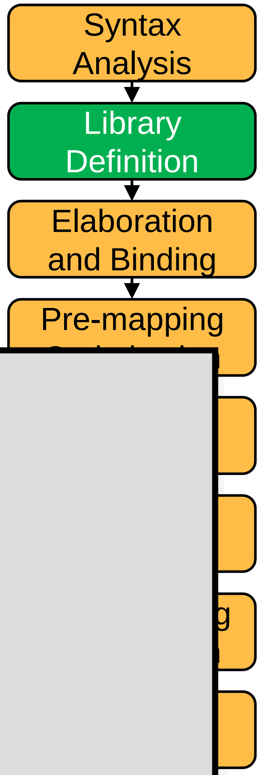
Obstructions

LEF

```
MACRO IV
    CLASS CORE ;
    FOREIGN IV 0.000 0.000 ;
    ORIGIN 0.00 0.00 ;
    SIZE 3.00 BY 12.00 ;
    SYMMETRY x y  ;
    SITE CORE ;
    PIN A
        DIRECTION INPUT ;
        ANTENNASIZE 1.4 ;
        PORT
        LAYER metal1 ;
        RECT  0.50 5.00 1.00 5.50
    ;
        END
    END A
              ⋮

    OBS
        LAYER metal1 ;
        RECT  1.90 6.50 2.60 7.20
    ;
        RECT  0.40 4.90 1.00 5.60
    ;
```



Technology LEF



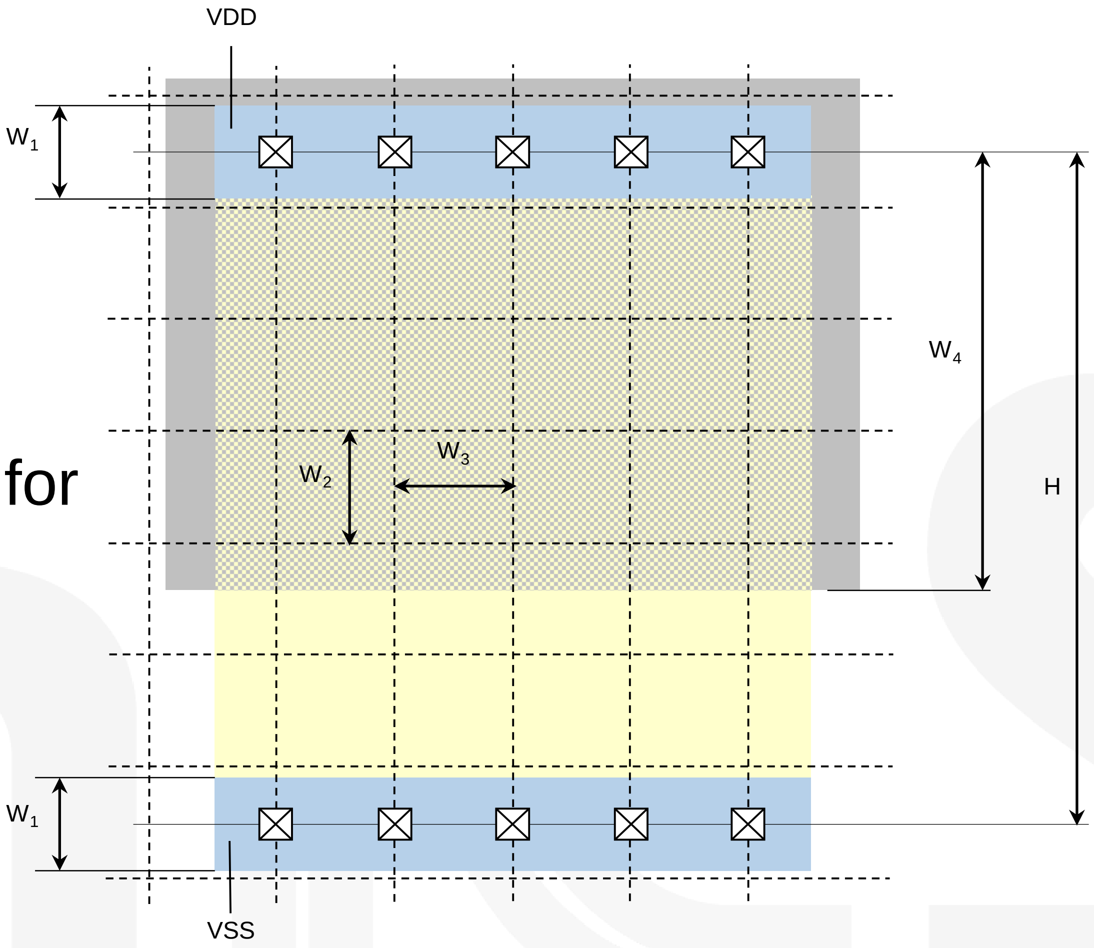
- **Technology LEF** **Files contain (simplified) information about the technology for use by the placer and router:**
  - **Layers**
    - Name, such as **M1**, **M2**, etc.
    - Layer type, such as routing, cut (**via**)
    - Electrical properties (**R**, **C**)
    - Design Rules
    - Antenna data
    - Preferred routing direction
  - **SITE** (**x** and **y** grid of the library)
    - **CORE** sites are minimum standard cell size
    - Can have site for double height cells!
    - IOs have special **SITE**.
  - **Via** definitions
  - **Units**
  - **Grids** for layout and routing

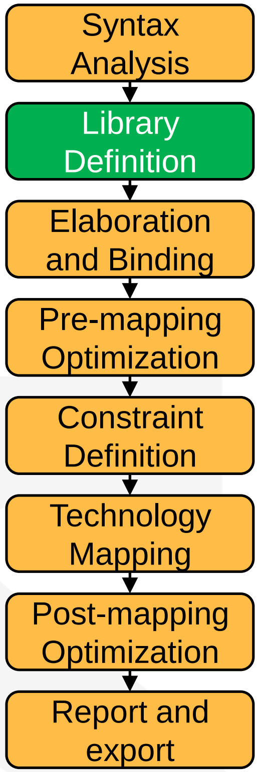
```
SITE CORE
  CLASS CORE;
  SIZE 0.2 X 12.0;
END CORE

LAYER MET1
  TYPE ROUTING ;
  PITCH 3.5 ;
  WIDTH 1.2 ;
  SPACING 1.4 ;
  DIRECTION HORIZONTAL ;
  RESISTANCE RPERSQ .7E-01 ;
  CAPACITANCE CPERSQDIST .46E-04 ;
END MET1

LAYER VIA
  TYPE CUT ;
END VIA
```

Additional files provide parasitic extraction rules. These can be basic (“cap tables”) or more detailed (“QRC techfile). These may be provided as part of the PDK.

Technology LEF

- **Cell height is measured in *Tracks***
  - A Track is one `M1` pitch
  - E.g., An `8-Track Cell` has room for 8 horizontal `M1` wires.
- **The more tracks, the wider the transistors, the faster the cells.**
  - 7-8 *low-track* libraries for area efficiency
  - 11-12 *tall-track* libraries for performance, but have high leakage
  - 9-10 *standard-track* libraries for a reasonable area-performance tradeoff



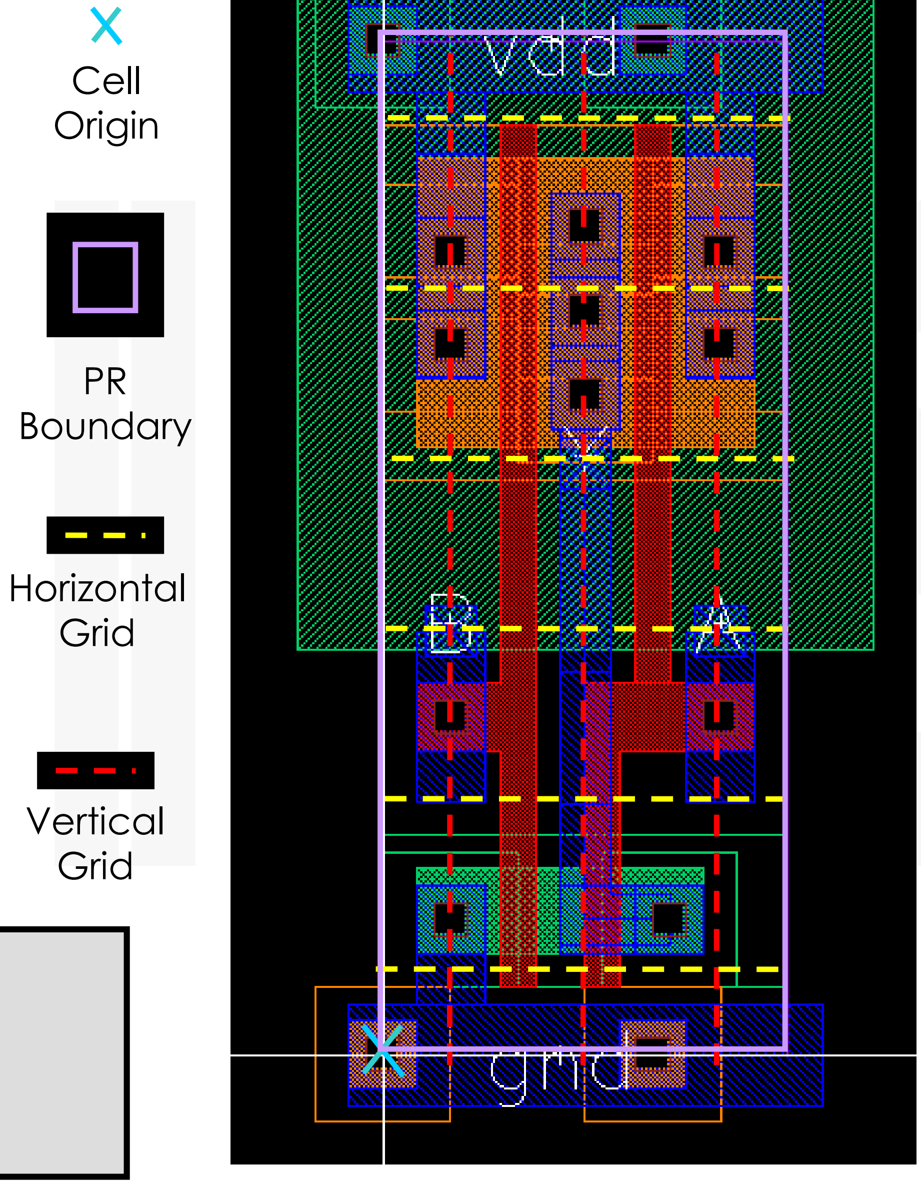


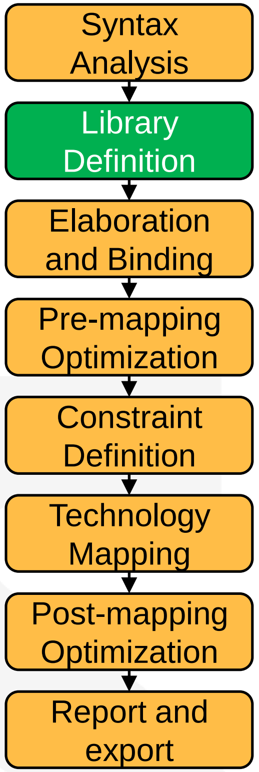
Parameter	Symbol
Cell height (# tracks)	H
Power rail width	W_1
Vertical grid	W_2
Horizontal grid	W_3
N-Well height	W_4

Technology LEF

- **Cells must fit into a predefined grid**
  - The minimum Height X Width is called a **SITE**.
  - Must be a multiple of the minimum X-grid unit and row height.
  - Cells can be double-height, for example.
- **Pins should coincide with routing tracks**
  - This enables easy connection of higher metals to the cell.

```
SITE CORE
  CLASS CORE;
  SYMMETRY X Y;
  SIZE 0.2 X 12.0;
END CORE
```

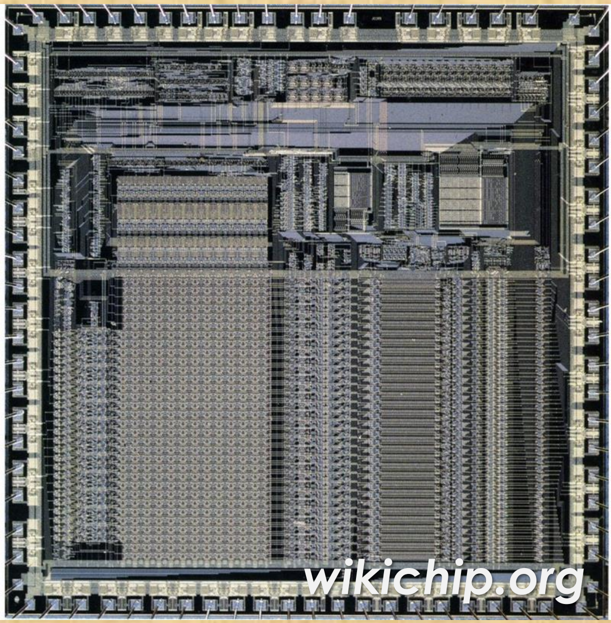




The Chip Hall of Fame

- **After checking out two Intel chips, we better not forget**

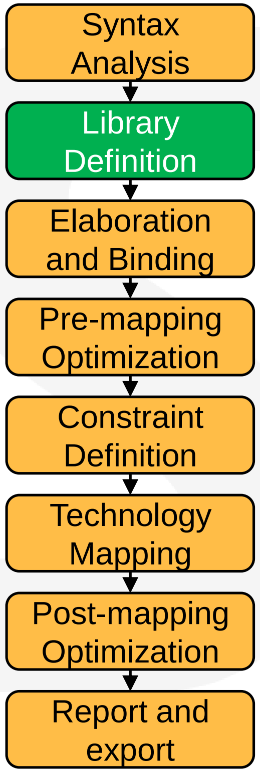
Acorn Computers
ARM1 Processor



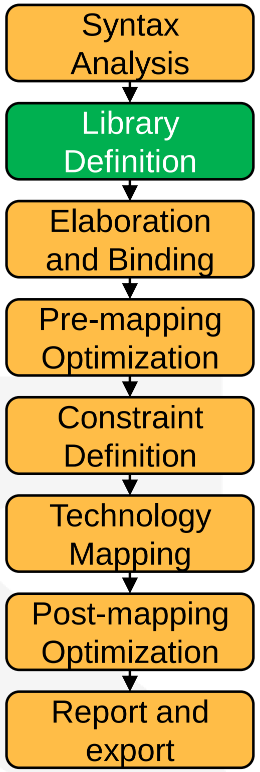
- Racking up Kahoot points on your smartphone?
  Then you probably should pay tribute to the granddaddy of that chip inside.
- Release date: April 1985 Manufactured by VLSI Technology
- Transistor Count: 25,000 Process: 3 um CMOS
- 32-bit ARMv1 architecture
- ARM stands for “Acorn RISC Machine”
- The reference design was written in 808 lines of BASIC!
- Never sold as a commercial product, but as a co-processor for BBC Micro.

2017 Inductee to the IEEE Chip Hall of Fame

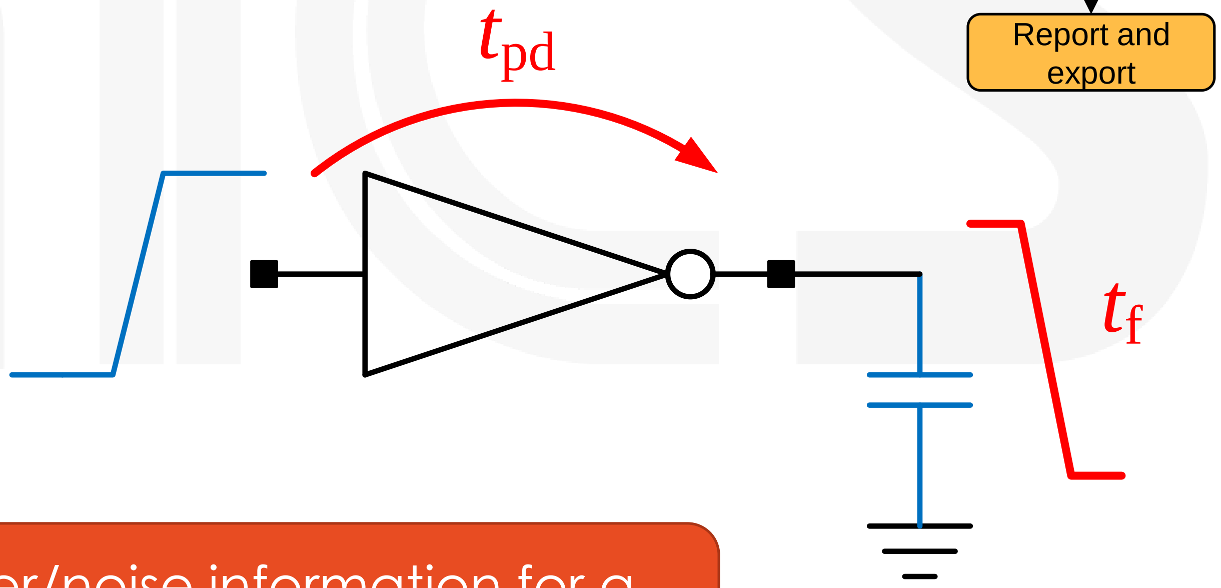
Liberty Timing Models (.lib)



Liberty (.lib): Introduction



- **How do we know the delay through a gate in a logic path?**
  - Running **`SPICE`** is way too complex.
  - Instead, create a *timing model* that will simplify the calculation.
- **Goal:**
  - For every timing arc, calculate:
    - Propagation Delay (t_{pd})
    - Output transition (t_{rise}, t_{fall})
  - Based on:
    - Input net transition (t_{rise}, t_{fall})
    - Output Load Capacitance (C_{load})

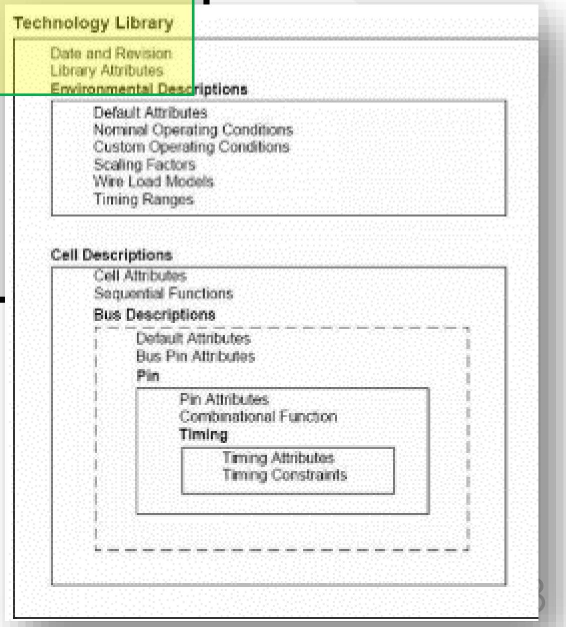


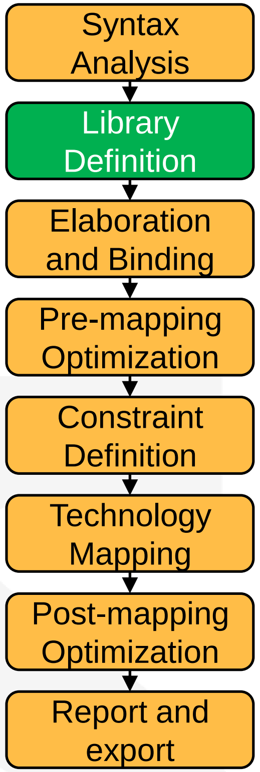
Note that every `.lib` will provide timing/power/noise information for a single corner, i.e., **process**, **voltage**, **temperature**, **RCX**, etc.

Liberty (.lib): General

- **Timing data of standard cells is provided in the Liberty format.**
  - Library:
    - General information common to all cells in the library.
    - For example, operating conditions, wire load models, look-up tables
  - Cell:
    - Specific information about each standard cell.
    - For example, function, area.
  - Pin:
    - Timing, power, capacitance, leakage, functionality, etc. characteristics of each pin in each cell.

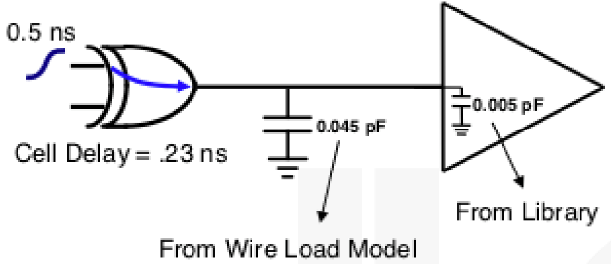
```
library (nameoflibrary) {
... /* Library level simple and complex attributes */

/* Cell definitions */
cell (cell_name) {
   ... /* cell level simple attributes */

   /* pin groups within the cell */
   pin(pin_name) {
       ... /* pin level simple attributes */

       /* timing group within the pin level */
       timing(){
            ... /* timing level simple attributes */ }
       ... /* additional timing groups */

   } /* end of pin */
   ... /* more pin descriptions */
} /* end of cell */
... /* more cells */

} /* end of library */
```





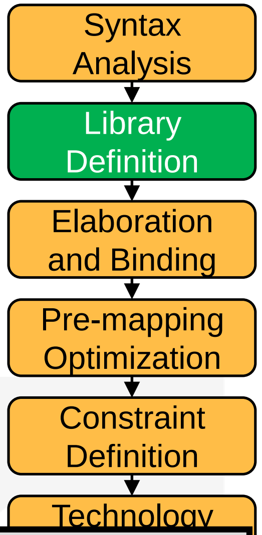
Liberty (.lib): Timing Models

- **Non-Linear Delay Model (NLDM)**
  - Driver model:
    - Ramp voltage source
    - Fixed drive resistance
  - Receiver model:
    - Min/max rise/fall input caps
  - Very fast
  - Doesn’t model cap variation during transition.
  - Loses accuracy beyond 130nm



Input Trans (ns)	Output Load (pF): .005	.05	.10	.15
0.0	.1	.15	.2	.25
0.5	.15	.23	.3	.38
1.0	.25	.4	.55	.75

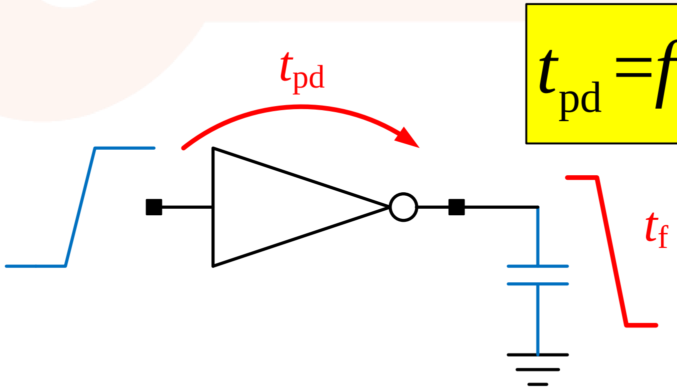
Cell Delay (ns)



```
lu_table_template(delay_template_5x5) {
  variable_1 : input_net_transition;
  variable_2 : total_output_net_capacitance;
  index_1 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
  index_2 ("1000.0, 1001.0, 1002.0, 1003.0, 1004.0");
}
```

```
cell (INVX1) {
  pin(Y) {
    timing() {
      cell_rise(delay_template_5x5) {
        values ( \
          "0.147955, 0.218038, 0.359898, 0.922746, 1.76604", \
          "0.224384, 0.292903, 0.430394, 0.991288, 1.83116", \
          "0.365378, 0.448722, 0.584275, 1.13597, 1.97017", \
          "0.462096, 0.551586, 0.70164, 1.24437, 2.08131", \
          "0.756459, 0.874246, 1.05713, 1.62898, 2.44989"); }
```

$$t_{pd} = f\left(t_{input}, C_{load}\right)$$



Liberty (.lib): Timing Models

- **Non-Linear Delay Model (NLDM)**
  - Delay calculation interpolation

Cell Fall

Cap\Tr	0.05	0.2	0.5
0.01	0.02	0.16	0.30
0.5	0.04	0.32	0.60
2.0	0.08	0.64	1.20

0.178

Cell Rise

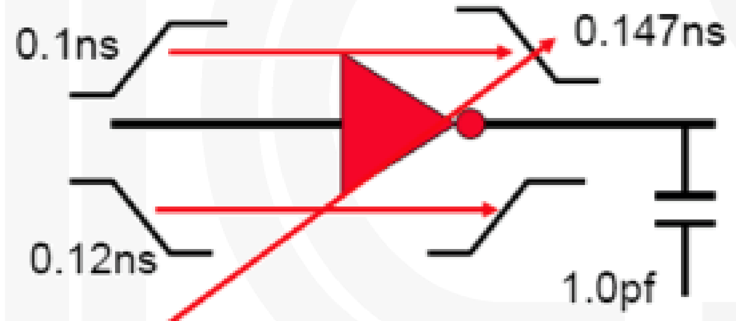
Cap\Tr	0.05	0.2	0.5
0.01	0.03	0.18	0.33
0.5	0.06	0.36	0.66
2.0	0.09	0.72	1.32

0.261

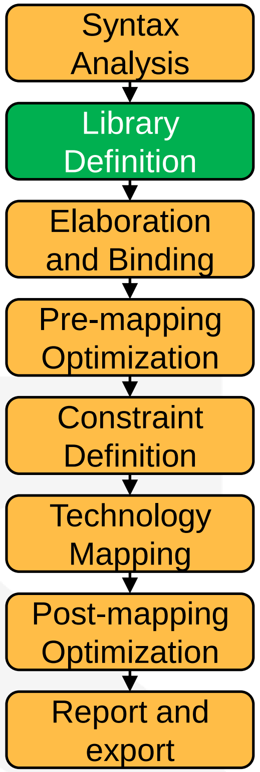
Fall Transition

Cap\Tr	0.05	0.2	0.5
0.01	0.01	0.09	0.15
0.5	0.03	0.27	0.45
2.0	0.06	0.54	0.90

0.147

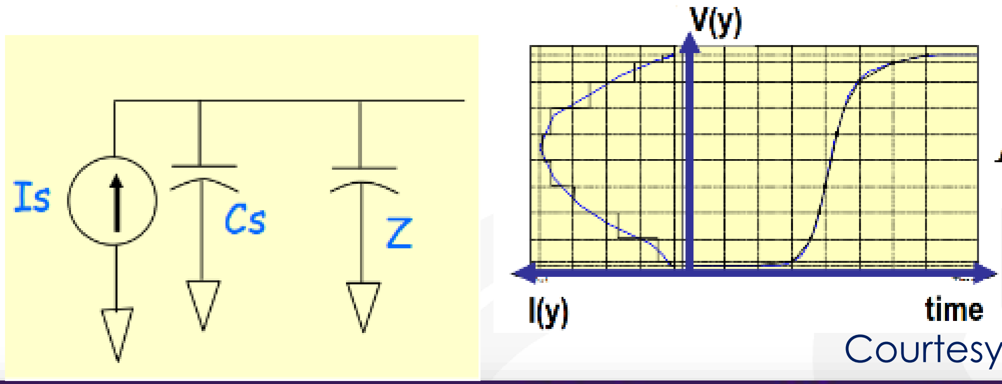


Fall delay = 0.178ns
Rise delay = 0.261ns
Fall transition = 0.147ns
Rise transition = …



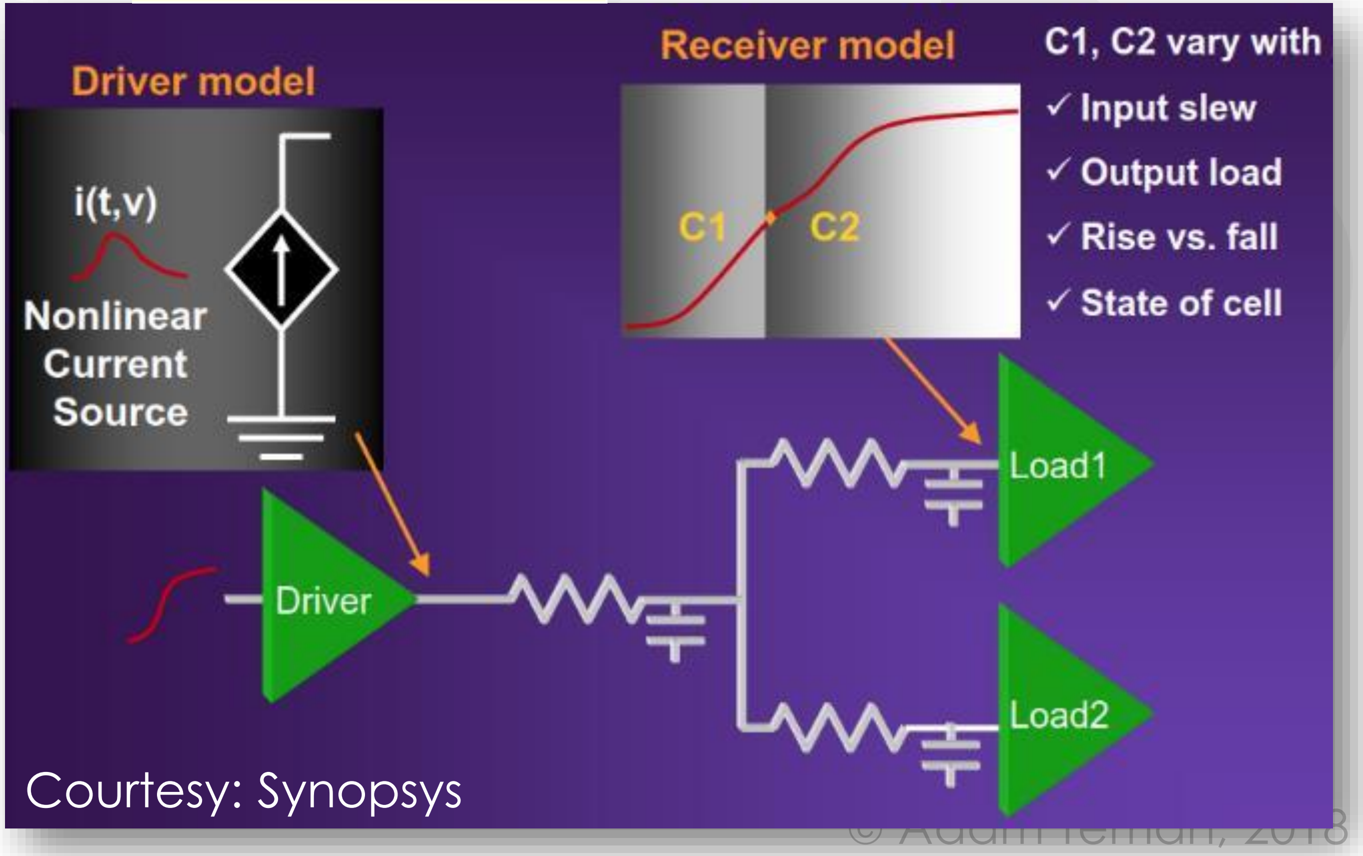
Liberty (.lib): Timing Models

- **Current Source Models (CCS, ECSM)**
  - Model a cell's nonlinear output behavior as a current source
  - Driver model:
    - Nonlinear current source
  - Receiver model:
    - Changing capacitance
  - Requires many more values
  - Requires a bit more calculation
  - Essential under 130nm
  - Within 2% of SPICE.



$$I = C\frac{\Delta V}{\Delta t}$$

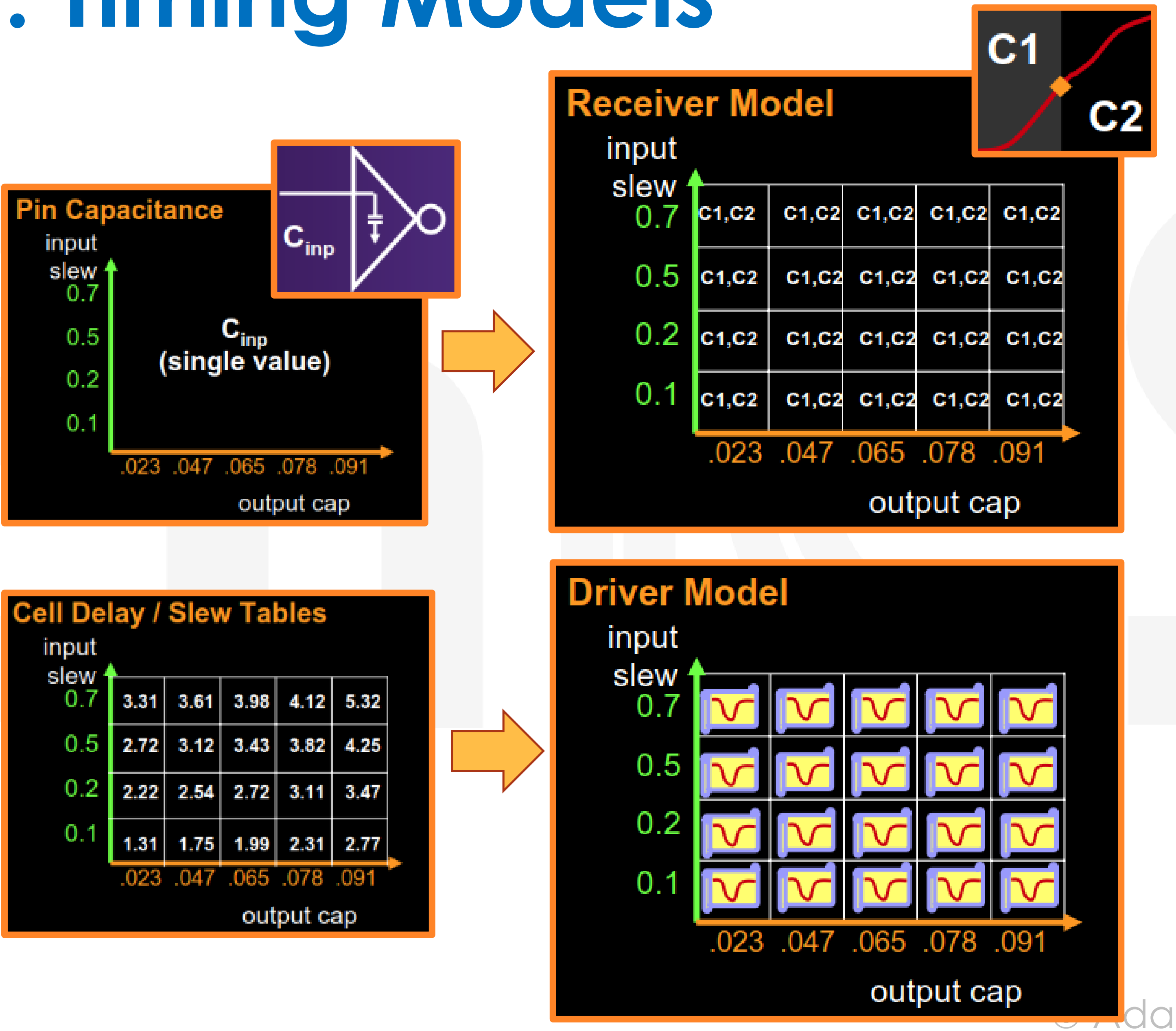
Courtesy: Cadence



Courtesy: Synopsys

Liberty (.lib): Timing Models

- **NLDM vs CCS/ECSM**

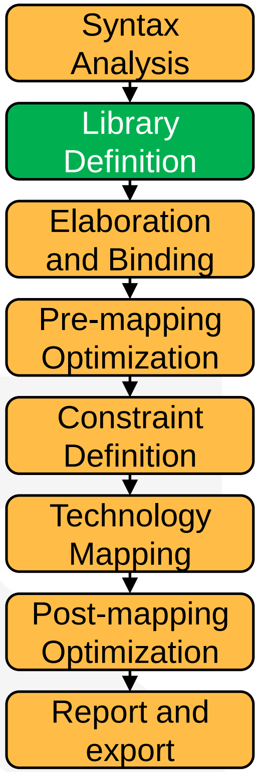


Cell Delay / Slew Tables

input slew \ output cap	.023	.047	.065	.078	.091
0.7	3.31	3.61	3.98	4.12	5.32
0.5	2.72	3.12	3.43	3.82	4.25
0.2	2.22	2.54	2.72	3.11	3.47
0.1	1.31	1.75	1.99	2.31	2.77

Receiver Model

input slew \ output cap	.023	.047	.065	.078	.091
0.7	C1,C2	C1,C2	C1,C2	C1,C2	C1,C2
0.5	C1,C2	C1,C2	C1,C2	C1,C2	C1,C2
0.2	C1,C2	C1,C2	C1,C2	C1,C2	C1,C2
0.1	C1,C2	C1,C2	C1,C2	C1,C2	C1,C2

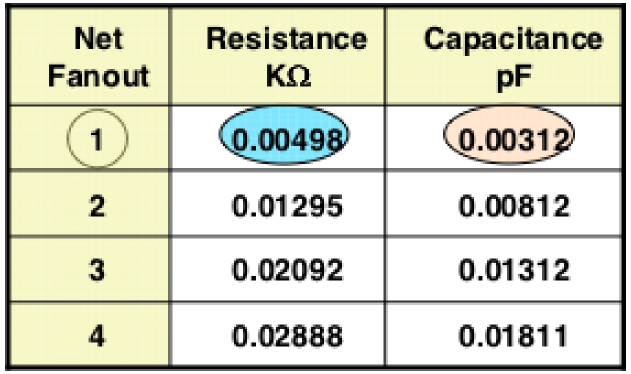


Syntax Analysis → Library Definition → Elaboration and Binding → Pre-mapping Optimization → Constraint Definition → Technology Mapping → Post-mapping Optimization → Report and export

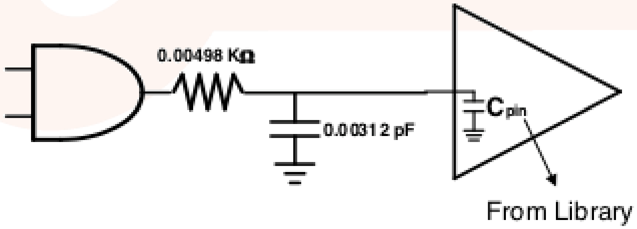
Courtesy: Synopsys

Liberty (.lib): Wire Load Models

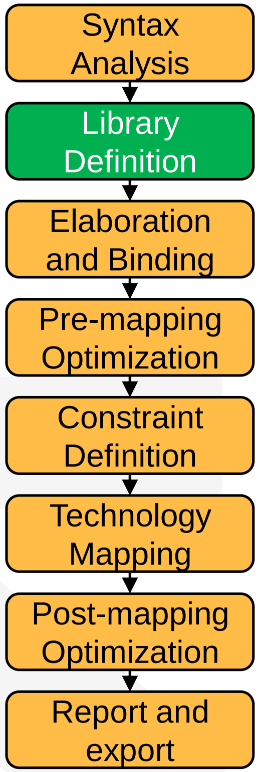
- How do you estimate the parasitics (RC) of a net before placement and routing?
- Wire Load Models estimate the parasitics based on the *fanout* of a net.



Net Fanout	Resistance KΩ	Capacitance pF
1	0.00498	0.00312
2	0.01295	0.00812
3	0.02092	0.01312
4	0.02888	0.01811

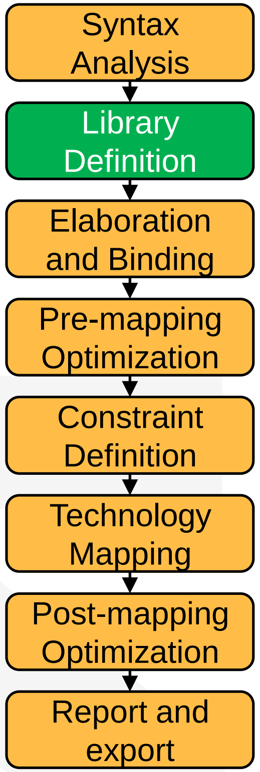


```
library (myLib) {
  wire_load(“WLM1”)
    resistance: 0.0006 ; // R per unit length
    capacitance: 0.0001 ; // C per unit length
    area : 0.1 ; // Area per unit length
    slope : 1.5 ; // Used for linear extrapolation
    fanout_length(1, 0.002) ; // for fo=1, Lwire=0.002
    fanout_length(2, 0.006) ; // for fo=2, Lwire=0.006
    fanout_length(3, 0.009) ; // for fo=3, Lwire=0.009
    fanout_length(4, 0.015) ; // for fo=4, Lwire=0.015
    fanout_length(5, 0.020) ; // for fo=5, Lwire=0.020
    fanout_length(6, 0.028) ; // for fo=6, Lwire=0.028
  }
} /* end of library */
```



http://www.vlsi-expert.com/

Physical-Aware Synthesis



Syntax Analysis

Library Definition

Elaboration and Binding

Pre-mapping Optimization

Constraint Definition

Technology Mapping

Post-mapping Optimization

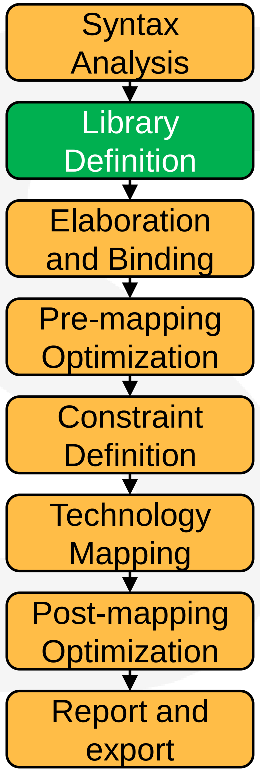
Report and export

- **Due to the lack of accuracy, wireload models lead to very poor correlation between synthesis and post-layout in nanometer technologies.**
- **Instead, use physical information during synthesis**
  - Synopsys calls this “Topographical Mode”
  - Cadence calls this “Physical Synthesis”
- **Physical-Aware Synthesis basically runs placement inside the synthesizer to obtain more accurate parasitic estimation:**
  - Without a floorplan, just using .lef files
  - After first iterations, import a floorplan .def to the synthesizer.

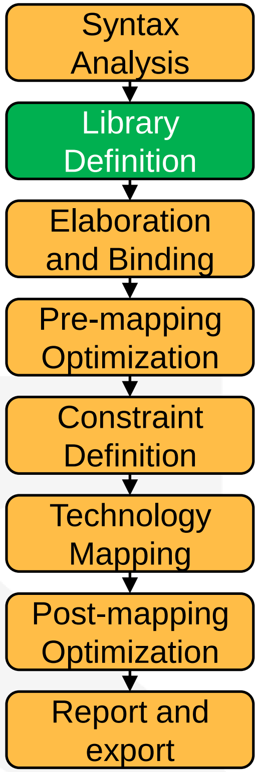
```
syn_opt -physical
```

1 Introduction | 2 Compilation | 3 Library Def. | 4 LEF | 5 Liberty | 6 Other

Other Contents of SC Library


Syntax Analysis
Library Definition
Elaboration and Binding
Pre-mapping Optimization
Constraint Definition
Technology Mapping
Post-mapping Optimization
Report and export

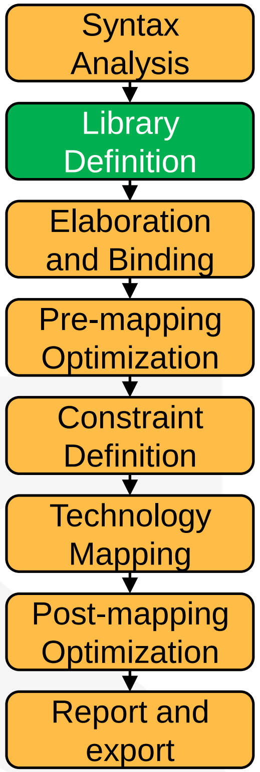
Other contents of SC Library



- **Many other files and formats may be provided as part of a standard cell library:**
  - GDS
  - Verilog
  - ATPG
  - Power Grid Models
  - OA Databases
  - Spice Models
  - etc.

Documentation and Datasheets

- **So, are we just supposed to look through and see what the vendor decided to provide us with?**
  - Yes!
  - However they probably supplied some PDFs describing the library.
  - And usually there are data sheets with numbers for each corner.



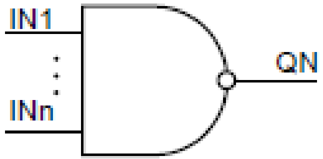
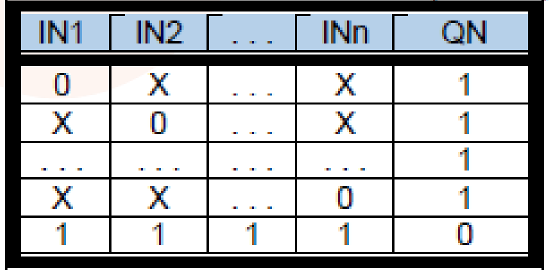


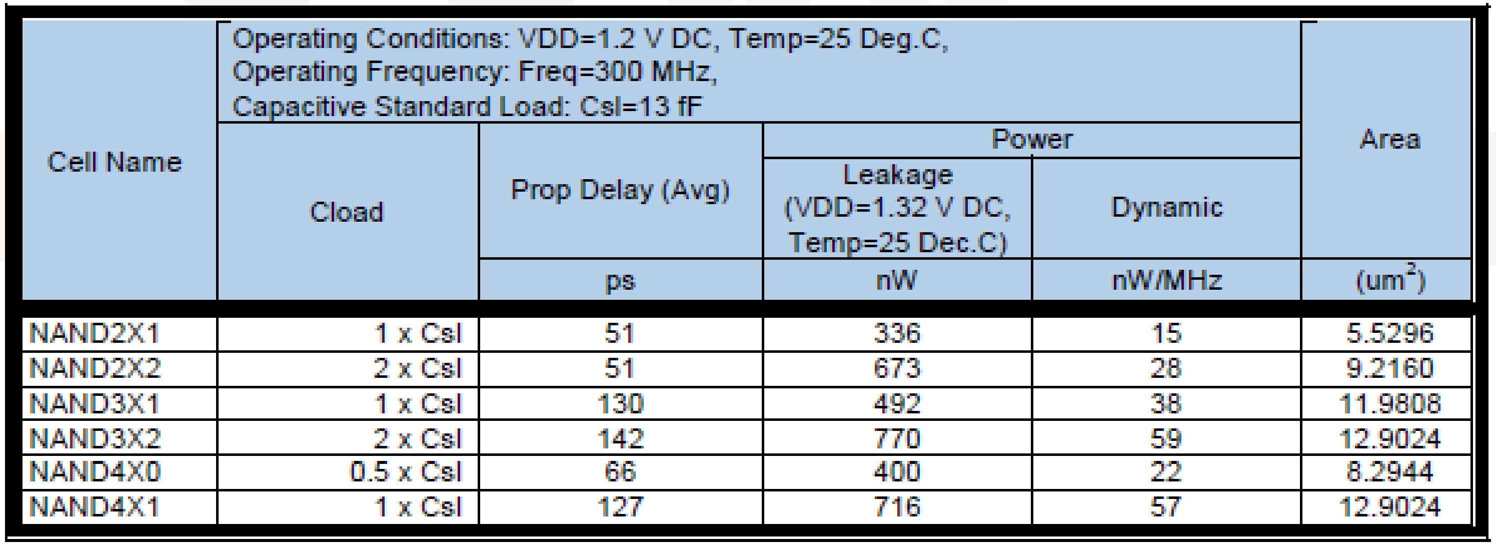
Figure 9.6. Logic Symbol of NAND

Table 9.11. NAND Truth Table (n=2,3,4)



IN1	IN2	. . .	INn	QN
0	X	. . .	X	1
X	0	. . .	X	1
. . .	. . .	. . .	. . .	1
X	X	. . .	0	1
1	1	1	1	0

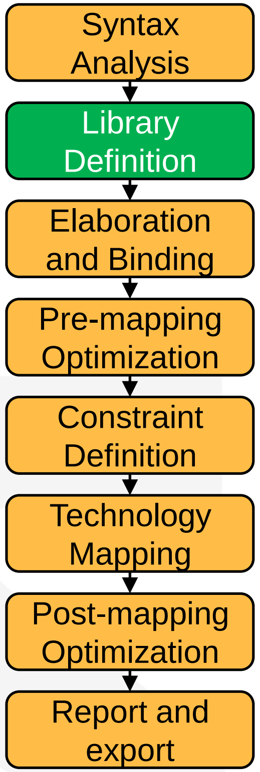
Table 9.12. NAND Electrical Parameters and Areas



Cell Name	Operating Conditions: VDD=1.2 V DC, Temp=25 Deg.C, Operating Frequency: Freq=300 MHz, Capacitive Standard Load: Csl=13 fF				Area
	Cload	Prop Delay (Avg)	Power		
			Leakage (VDD=1.32 V DC, Temp=25 Dec.C)	Dynamic	
		ps	nW	nW/MHz	(um^2)
NAND2X1	1 x Csl	51	336	15	5.5296
NAND2X2	2 x Csl	51	673	28	9.2160
NAND3X1	1 x Csl	130	492	38	11.9808
NAND3X2	2 x Csl	142	770	59	12.9024
NAND4X0	0.5 x Csl	66	400	22	8.2944
NAND4X1	1 x Csl	127	716	57	12.9024

And what about other IPs?

- All IPs will be provided as a library, including most of the ***views*** a standard cell library will have.
- These are required for integration of the hard macros in the standard design flow (**simulation**, **synthesis**, **P&R**, **verification**, etc.)
- Memories (SRAMs) are a special case, as they usually come with a ***memory compiler*** that generates the particular memory cut the designer requires.



Syntax Analysis

Library Definition

Elaboration and Binding

Pre-mapping Optimization

Constraint Definition

Technology Mapping

Post-mapping Optimization

Report and export