ASIC flow: mapping to cells

Rajit Manohar

Asynchronous VLSI and Architecture (AVLSI) Group
Computer Systems Lab, Yale University

`https://csl.yale.edu/~rajit/`
`https://avlsi.csl.yale.edu/act`

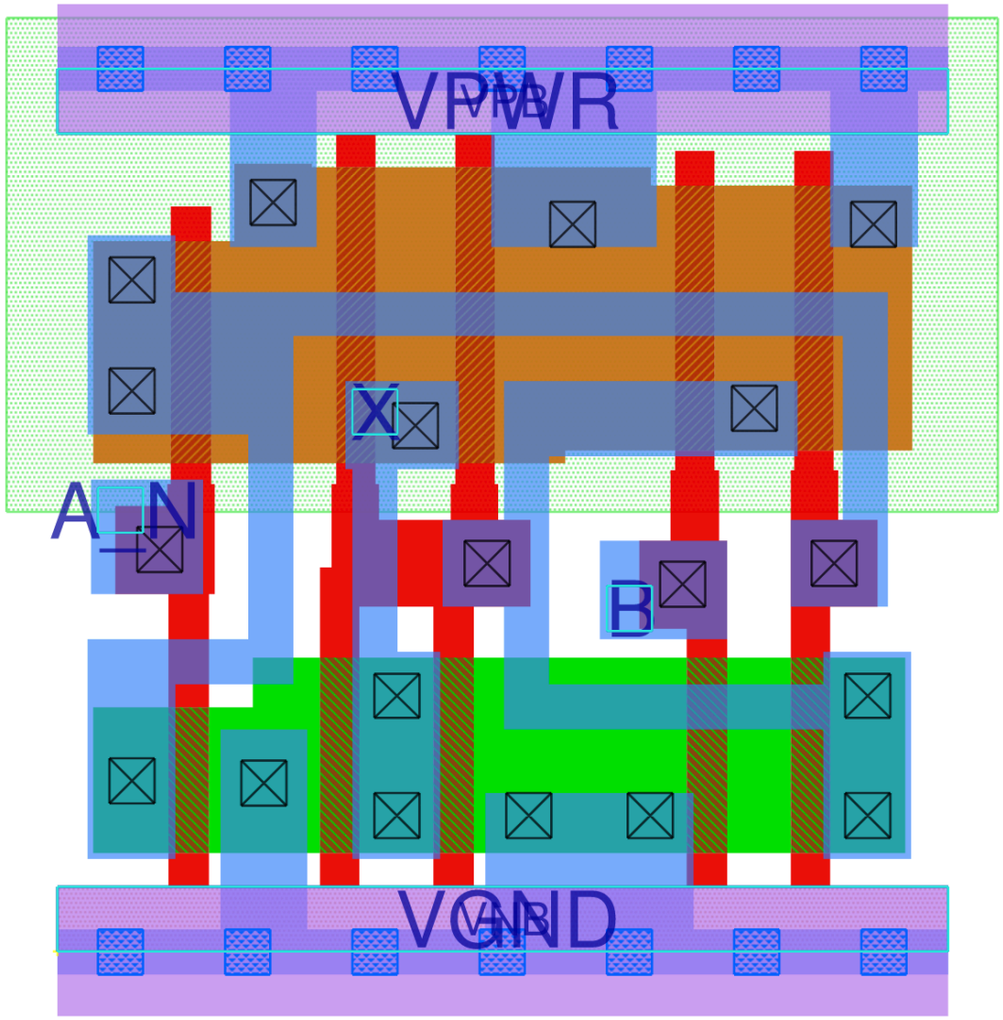
Yale

AVLSI

Basic idea

- Convert entire design into a set of pre-defined blocks (“cells”)
  - ❖ Examples
    - two-input NAND gate
    - two-input C-element
    - inverters with different sizing
    - … etc …

- Each cell is implemented at the transistor-level *once*
  - ❖ Rectangular geometry with input/output connection points (“pins”)



Example from Skywater 130 library for synchronous logic
https://antmicro-skywater-pdk-docs.readthedocs.io

Yale

AVLSI

Input: gate-level design in ACT

- Gate level design can have
  - ❖ Explicitly instantiated cells
  - ❖ Production rules

- Explicitly instantiated cells
  - ❖ We use your cell instances as specified
  - ❖ Each cell needs a physical implementation
  - ❖ Example:
    - ▸ Mapping arithmetic (e.g. “x + y”) using logic synthesis results in a collection of gates selected by the logic synthesis tool
  - ❖ A *technology-independent* cell library for combinational logic is available, based on James Stine’s open-source library in 180nm (“OSU library”)

Input: gate-level design in ACT

- Production rule mapping

```
prs {
    A & B #> Y-

    Y<30> -> W-
   ~Y<30> -> W+
}
```

- Unique cells are identified

```
  A & B -> Y-
~A & ~B -> Y+
```

```
 Y<30> -> W-
~Y<30> -> W+
```

- ACT is re-written to explicitly instantiate these cells
  - ❖ An ACT cell library is generated, consisting of *unique production rules* across the design
  - ❖ An existing ACT cell library can be re-used, and is extended if necessary

Yale

AVLSI

Open-source ACT based on OSU library

```
export defcell NOR2X1 (bool? A, B; bool! Y)
{
  prs {
     A | B => Y-
  }
  sizing { Y {-1} }
}
```

ACT cell

```
.subckt NOR2X1 vdd B gnd Y A
M0 a_9_54# A vdd vdd pfet w=4u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M1 Y B a_9_54# vdd pfet w=4u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M2 Y A gnd Gnd nfet w=1u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M3 gnd B Y Gnd nfet w=1u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
.ends NOR2X1
```

Original OSU cell

$ACT_HOME/act/std/cells.act

Yale

AVLSI

Open-source ACT based on OSU library

```
.subckt HAX1 vdd gnd YC A B YS
M0 vdd A a_2_74# vdd pfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M1 a_2_74# B vdd vdd pfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M2 vdd a_2_74# YC vdd pfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M3 a_41_74# a_2_74# vdd vdd pfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M4 a_49_54# B a_41_74# vdd pfet w=4u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M5 vdd A a_49_54# vdd pfet w=4u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M6 YS a_41_74# vdd vdd pfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M7 a_9_6# A gnd Gnd nfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M8 a_2_74# B a_9_6# Gnd nfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M9 gnd a_2_74# YC Gnd nfet w=1u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M10 a_38_6# a_2_74# gnd Gnd nfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M11 a_41_74# B a_38_6# Gnd nfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M12 a_38_6# A a_41_74# Gnd nfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M13 YS a_41_74# gnd Gnd nfet w=1u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
.ends HAX1
```

Original OSU cell

Yale

AVLSI

Open-source ACT based on OSU library

```
export defcell HAX1 (bool? A, B; bool! YC, YS)
{
  bool _YC, _YS;
  prs {
     A & B => _YC-
    _YC  => YC-

    _YC & (A | B) => _YS-
    _YS => YS-
  }
  sizing { _YC{-1}; YC{-1}; _YS{-1}; YS{-1} }
}
```

ACT cell

```
.subckt HAX1 vdd gnd YC A B YS
M0 vdd A a_2_74# vdd pfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M1 a_2_74# B vdd vdd pfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M2 vdd a_2_74# YC vdd pfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M3 a_41_74# a_2_74# vdd vdd pfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M4 a_49_54# B a_41_74# vdd pfet w=4u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M5 vdd A a_49_54# vdd pfet w=4u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M6 YS a_41_74# vdd vdd pfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M7 a_9_6# A gnd Gnd nfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M8 a_2_74# B a_9_6# Gnd nfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M9 gnd a_2_74# YC Gnd nfet w=1u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M10 a_38_6# a_2_74# gnd Gnd nfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M11 a_41_74# B a_38_6# Gnd nfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M12 a_38_6# A a_41_74# Gnd nfet w=2u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
M13 YS a_41_74# gnd Gnd nfet w=1u l=0.2u
+ ad=0p pd=0u as=0p ps=0u
.ends HAX1
```

Original OSU cell

Yale

AVLSI

Usage scenarios

- "I have all my cells and they have been instantiated already!"
  - ACT can use your cells as "black box" components
  - For each cell, we will need
    - Black-box: declare cell but do not provide a definition

```
export defcell MY_TWO_INPUT_CELL (bool? A, B; bool! Y);
```

    - LEF, GDS, timing information from .lib (as in a normal cell library)

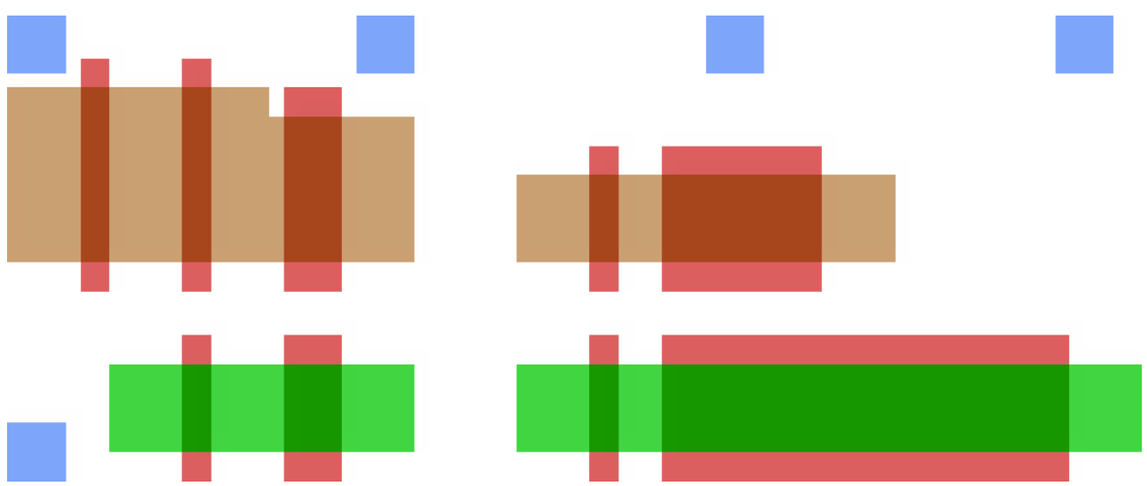
- Using an existing physical design flow? (e.g. a commercial tool)
  - Map everything to cells
  - Export Verilog netlist or DEF file
  - Use your cell library

Yale

AVLSI

Usage scenarios

- “I don’t have any cells!”
  - ❖ The ACT flow can get you started…
    - ‣ Initial placement of transistors, ready to be wired up
    - ‣ Key requirement: “Skyline” layout

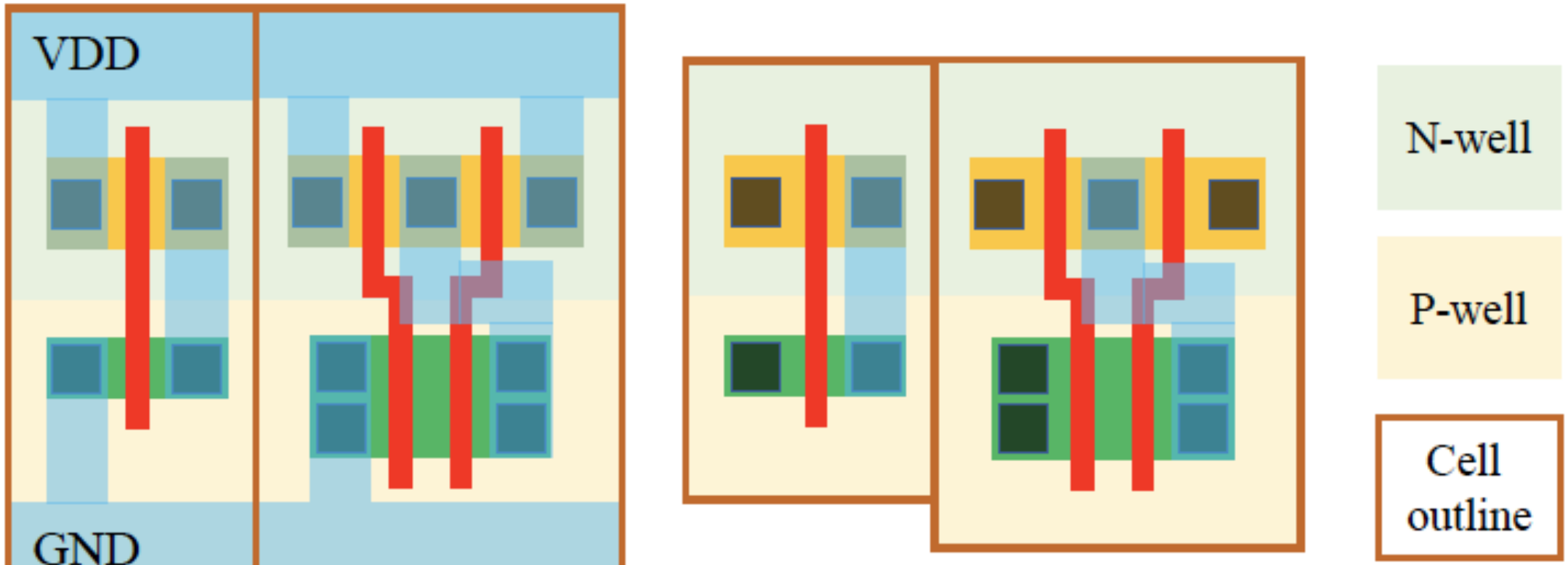


- “I have some cells, but not sure if I have them all”
  - ❖ ACT will let you know if there are any missing cells when mapping to the design

Yale

AVLSI

ACT supports *gridded* cells



"Standard" cells

most cells are fixed height (e.g. 12 tracks), width is integer multiple of tracks; some "multi-height" cells

"Gridded" cells

cells width and height is integer multiple of tracks; can have "multi-deck" cells

Yale

AVLSI