3BA4 Design Rules

Design Rules for
Weste & Eshraghian CMOS Process ($\lambda = 0.5\,\mu m$)

```
/users/Public/BAICT/3BA4/3BA4-rules.txt
```

```
TITLE "Weste&Eshraghian CMOS (lambda=0.5micron) design rules"
```

These are design rule that match those in this textbook.

Key Concepts

- Idea: compute "error layers".
- Computational Tool: "Mask Operations"
  - Compute new (pseudo-)layers in terms of old ones
  - Maskop AND:
    l_1 AND l_2 gives a new layer that corresponds to the overlap of layers l_1 and l_2
  - Maskop OR:
    l_1 OR l_2 gives a new layer that corresponds to the total extent of both layers l_1 and l_2
  - Maskop NOT:
    l_1 NOT l_2 gives a new layer that corresponds to all of layer l_1 that is not covered by l_2.
- A design rules file specifies how to compute "error layers", that graphically show the extent of a particular error.
  The error layer definition has the form: *Error_Layer_Name* $\{$ error layer definition $\}$

Layer Definitions

```
substrate = ALL NOT Nwell
ndiff = Nplus AND substrate
vddn = Nwell AND Nplus
vssp = Pplus AND substrate
pdiff = Nwell AND Pplus
diff = Nplus OR Pplus
active = ndiff OR pdiff
ngate = Poly AND ndiff
pgate = Poly AND pdiff
gate = ngate OR pgate
wired = Nplus OR Pplus
wire = wired OR Poly
vmetal = Metal2 OR Metal3
```

General Rules

All lines must be vertical, horizontal or at forty-five degrees:

```
Skew_Edges{ DRAWN SKEW }
```

Poly should not overlap substrate contact diffusions:

```
badptran = Poly AND vssp
badntran = Poly AND vddn
Bad_Gates { badntran OR badptran }
```

Special Error Layer Operations

There are special mask operations used specifically to generate error layers for certain problems.

INTERNAL Moves all edges inward the amount specified — notes any contact with other edges as a violation

EXTERNAL Moves all edges outward the amount specified — notes any contact with other edges as a violation

COINCIDENT EDGE Flags any edges of two shapes lying on top of one another

ENCLOSURE Flags any edges of one layer not inside another by desired margin

NOT INSIDE Flags any layer shapes not inside shapes of another layer.

These operations typically take one or two layer names as parameters — to support both checks within one layer, and between two different layers.

Minimum Widths

(Letters and numbers refer to Weste&Eshraghian)

```
A1_Nwell_Width { INTERNAL Nwell < 5 }
B1_Active_Width {
  INTERNAL Nplus < 1.5
  INTERNAL Pplus < 1.5
}
C1_Poly_Width { INTERNAL Poly < 1 }
E1_Cut_Width { INTERNAL Ccut < 1 }
F1_Metal1_Width { INTERNAL Metal1 < 1.5 }
```

Minimum Separations (self)

```
// N-Wells at same potential
A2_Nwell_Sep { EXTERNAL Nwell < 3 }
// N-Wells at different potentials
A3_Nwell_PD  { EXTERNAL Nwell < 4 }
B2_Active_Sep {
  EXTERNAL Nplus < 1.5
  EXTERNAL Pplus < 1.5
}
C2_Poly_Sep { EXTERNAL Poly < 1 }
D4_Gate_Sep { EXTERNAL gate < 1.5 }
E2_Cut_Sep { EXTERNAL Ccut < 1 }
F2_Metal1_Sep { EXTERNAL Metal1 < 1.5 }
```

Minimum Separations (other)

```
B5_Nwell_ndiff_Sep {
 EXTERNAL Nwell ndiff < 2.5
 COINCIDENT EDGE Nwell ndiff }
B6_Nwell_vssp_Sep {
  EXTERNAL Nwell vssp < 1.5
  COINCIDENT EDGE Nwell vssp }
C3_Poly_Active_Sep {
  EXTERNAL Poly Nplus < 0.5
  EXTERNAL Poly Pplus < 0.5
  COINCIDENT EDGE Poly Nplus
  COINCIDENT EDGE Poly Pplus }
E7_gate_Cut_sep {
   EXTERNAL gate Ccut < 1
   COINCIDENT EDGE gate Ccut }
```

Transistor Extensions

```
C4_Gate_Ext {
  ENCLOSURE active Poly < 1
```

Contact Overlaps

```
B3_NW_pdiff_Overlap {
  ENCLOSURE pdiff Nwell < 2.5
  COINCIDENT EDGE pdiff Nwell
 }
B4_NW_vddn_Overlap {
  ENCLOSURE vddn Nwell < 1.5
  COINCIDENT EDGE vddn Nwell
 }
E4_wire_Cut_Ovl {
  ENCLOSURE Ccut wire < 1
  COINCIDENT EDGE Ccut wire
 }
E6_Metal1_Cut_Ovl {
  ENCLOSURE Ccut Metal1 < 0.5
  COINCIDENT EDGE Ccut Metal1
 }
```

Badly Formed Contacts

```
Cut_Conn {
  Ccut NOT INSIDE Metal1
  Ccut NOT INSIDE wire
}
Via_Conn {
  Via NOT INSIDE Metal1
  Via NOT INSIDE vmetal
}
```

Netlist Extraction and Comparison

How to check your layout against a netlist specification:

1. Obtain Reference netlist
   - presented in SPICE format
2. Extract Test netlist from your Layout
   - use ICTrace (M) Palette
   - save as HSPICE format, with .net extension
3. Use NetCheck program to compare them.
   - written in-House, in Haskell (`http://www.haskell.org`)
   - A bundle of “Literate Haskell Scripts” (.lhs)
   - Executed via Hugs interpreter

SPICE Format

Excerpt from *Reference.net*

```
m3 5 14 3 pmos l=1u w=1.5u
m4 8 3 7 nmos l=1u w=1.5u
```

Translation:

`m3` — MOSFET Number

`5 14 3` — Nodes connected to s/d (5), gate (14) and other s/d (3).

`pmos l=1u w=1.5u` — MOSFET type, length and width (`u`=μm)

In Haskell

```
{3 |-> MOS{d1=5,g=14,d2=3,t=P,l=1.0,w=1.5},
 4 |-> MOS{d1=8,g=3,d2=7,t=N,l=1.0,w=1.5}}
```

The Reference Network

```
m0 3 14 1 nmos l=1u w=1.5u
m1 3 15 1 pmos l=1u w=1.5u
m2 5 15 3 nmos l=1u w=1.5u
m3 5 14 3 pmos l=1u w=1.5u
m4 8 3 7 nmos l=1u w=1.5u
m5 6 3 8 pmos l=1u w=4.5u
m6 5 8 7 nmos l=1u w=1.5u
m7 6 8 5 pmos l=1u w=4.5u
m8 13 17 5 nmos l=1u w=1.5u
m9 13 16 5 pmos l=1u w=1.5u
```

A Good network

```
m1 2 3 9 nmos l=1.0u w=1.5u
m2 2 4 9 pmos l=1.0u w=1.5u
m3 0 9 10 nmos l=1.0u w=1.5u
m4 1 9 10 pmos l=1.0u w=4.5u
m5 0 10 8 nmos l=1.0u w=1.5u
m6 1 10 8 pmos l=1.0u w=4.5u
m7 9 4 8 nmos l=1.0u w=1.5u
m8 9 3 8 pmos l=1.0u w=1.5u
m9 8 5 7 nmos l=1.0u w=1.5u
m10 8 6 7 pmos l=1.0u w=1.5u
```

How to compare them?

The node numbers don’t matter — the overall connectivity does.

Showing that two graphs have the same shape is known as the Graph Isomorphism problem.

It is hard to solve in general (NP-Complete)

The NetCheck program works for small networks.

NetCheck processing

1. Get Reference netlist
2. Get Test netlist
3. Compare Connection Types
4. If different, report mismatches and quit
5. Otherwise, Compare Graphs
6. If different, report mismatches and quit/
7. Otherwise, report “TEST NETWORK OK”

Connection Types

For every node, we count:

g_no No. of gates to which this node is connected

n_sd_no No. of nMOS source/drain terminals to which this node is connected

p_sd_no No. of pMOS source/drain terminals to which this node is connected

In the program, a connection type is written out as:

```
ConnType{g_no=1,n_sd_no=3,p_sd_no=0}
```

Comparing Connection Types

We scan the netlist computing connection types for every circuit node.

```
{3 |-> ConnType{g_no=1,n_sd_no=3,p_sd_no=0},
,4 |-> ConnType{g_no=0,n_sd_no=1,p_sd_no=1},
,5 |-> ConnType{g_no=1,n_sd_no=3,p_sd_no=0}}
```

We then invert this so for all connection types present, we list the set of nodes with that type.

```
{ConnType{g_no=1,n_sd_no=3,p_sd_no=0} |-> {3,5},
,ConnType{g_no=0,n_sd_no=1,p_sd_no=1} |-> {4}}
```

We compare these for the Reference and Test netlists
— both netlists must have the same collection of connection types,
— and each should map to a set of the same size.

This test captures most mistakes!

Network Graphs

We map each node to a set of the nodes to which it is connected via source/drain regions, labelled with details of the gate node, and transistor type and size.

```
3 |-> {(5,CH 6 N 1.0 1.5),(7,CH 8 P 1.0 4.5)}
```

Here 3 connects to 5 via nMOS with gate node 6, and to 7 via pMOS with gate node 8.

Nodes with the same connection type are searched to see if we can find a match.

We report an error if no match found.