static and dynamic principle

static principle:
output determined by a connection with "strong" node

dynamic principle:
output (sometimes) determined by a "weak" (floating) node

charging:
$C_s$ is being charged up to $V_+ - \text{level}$

evaluation:
$C_s$ is being discharged \textbf{when} there is a conducting path to mass
realisation of dynamic logic

interchange network and evaluation switch

"precharge" switch

evaluation switch

control with levels

toggle switch becomes selector

precharge switch

evaluation switch

pass network

control with levels

pass network

pass network

pass network

pass network
cmos-realisation of dynamic logic

φ low: \( C_s \) is being charged up to \( V_+ \) - level ("precharge")

φ high: \( C_s \) is being discharged when the nmos network has a conducting path; else: \( v_{\text{out}} \) shifts slowly towards an intermediate level
evaluation speed

Dynamic logic is faster, but what about precharge?
“hiding” precharge time

slave clock

T_S

slave transparant

precharge

T_P

p-e clock

master transparant

master clock

T_M

slave latches

combinatorial logic

master latches

static logic:

T_M + T_S + max \sum_{\text{path}} T_b(\text{GATE}_S)

dynamic logic:

T_M + \max (T_S, T_P) + max \sum_{\text{path}} T_D(\text{GATE}_D)
the cascade problem

solutions:
1. "dummy" imitates the slowest gate ("self timing")

φ' must come sufficiently later than φ
otherwise spurious discharge occurs (als a=b=c=d=1)
no-race logic (nora)

solutions:
1. "dummy" imitates the slowest gate ("self timing")
2. alternate nmos and pmos
   - more complicated logic synthesis

- noise sensitivity because of possibly large floating nodes
- an extra clock line -> area
- more pmos network (and therefore slower)
- dead time for "skew" protection (more delay, unless . . . . )
solutions:
1. "dummy" imitates the slowest gate ("self timing")
2. alternate nmos and pmos
3. invert the output
   • only unate logic possible

• less noise sensitivity because
  - output from strong logic
  - concentrated floating node
  - effect only when sense node changes $\gg V_t$
domino logic

precharge:
"sense node" is being charged
\( v_{\text{sense}} \to V_+ \); \( v_{\text{out}} \to 0 \) V;
nmos transistors in fanout
\( \to \) do not conduct!

evaluation:
if "sense node" is being charged
\( v_{\text{sense}} \to 0 \) V; \( v_{\text{out}} \to V_+ \);
transistors in fanout will open
else \( v_{\text{out}} \) stays low;
nmos transistors in fanout
still do not conduct

solutions:
1. "dummy" imitates the slowest gate ("self timing")
2. alternate nmos and pmos
3. invert the output
   • only unate logic possible

• less noise sensitivity because
  - output from strong logic
  - concentrated floating node
  - effect only when sense node changes \( \gg V_t \)
charge sharing in domino logic

\[ t_0 < t < t_i : \quad v_S = V_+ \]
\[ Q_{\text{tot}} = C_S V_+ \]

\[ t >> t_i : \quad Q_{\text{tot}} = C_S v_S + (C_1 + C_2) v_S \]

\text{when} \ v_S < v_i - V_t

\[ t \to \infty : \quad v_S \to \frac{C_S V_+}{C_S + C_1 + C_2} = \frac{V_+}{1 + \frac{C_1 + C_2}{C_S}} \]

\[ v_S \geq V_{IH} \Rightarrow \frac{C_1 + C_2}{C_S} < \frac{V_+ - V_{IH}}{V_{IH}} \]
charge sharing in domino logic

reduce charge sharing by:
- enlarging $C_s$
- precharging internal nodes
- bleeders

all measures delay evaluation!
complementary network pairs in nmos

A cascode switch is a network between two decision nodes and a strong node with the property that always exactly one decision node is electrically connected with the strong node.

Example: pull-down cascode switch (strong node is low).

“Pull-down cascode switches consists of two complementary networks (n-type switches), but now with the complementary signals controlled.”
differential cascode-switches

various pull-up possibilities for pull-down cascode switches

attention: low levels strong, but high levels are weak!

if \( \phi \) low, than \( D_1 \) and \( D_2 \) high!

if \( \phi \) high, than either \( D_1 \) or \( D_2 \) low!
differential cascode switch

here too
a complementary network!

but now:
- both nmost
- complementary
  signals needed

→ more wires, better testability
from table to decision diagram

representations of logic functions
• minterms or maxterms
• truth table
• sum of products
  or product of sums
• factored expressions
• ...........
• tree structures

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
from table to decision diagram

representations of logic functions

- minterms or maxterms
- truth table
- sum of products
  or product of sums
- factored expressions
- ........
- tree structures

--- each path from the root to a leaf is a minterm
--- function value “in” leaf

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
from table to decision diagram

representations of logic functions
- minterms or maxterms
- truth table
- sum of products or product of sums
- factored expressions
- tree structures

-- each path from the root to a leaf is a minterm
-- function value “in” leaf
from table to decision diagram

eliminate variables with two identical sub trees!

variable $d$ has in the green sub trees no impact: “does not discriminate”
from table to decision diagram

eliminate variables with two identical sub trees!

variable d is in the green sub trees eliminated!
from table to decision diagram

eliminate variables with two identical sub trees!

variable d is in the green sub trees eliminated!

variable b has in the green sub tree no impact: “does not discriminate”
from table to decision diagram

eliminate variables with two identical sub trees!

variable b is in the green sub tree eliminated
from table to decision diagram

1. eliminate variables with two identical sub trees!
2. identify identical sub trees!

the two green sub trees are (inclusive labels) identical!
from table to decision diagram

1. eliminate variables with two identical sub trees!
2. identify identical sub trees!

the two green sub trees are (inclusive labels) identified!
from table to decision diagram

1. eliminate variables with two identical sub trees!
2. identify identical sub trees!

the diagram is no longer a "tree"!
("dag" : directed acyclic graph)

variable c has in the green sub tree no impact: "does not discriminate"
from table to decision diagram

1. eliminate variables with two identical sub trees!
2. identify identical sub trees!

the diagram is no longer a "tree"!
("dag" : directed acyclic graph )

variable c is in the green sub tree eliminated!
from table to decision diagram

1. eliminate variables with two identical sub trees!
2. identify identical sub trees!
3. identify the 0- and 1-leafs!

the diagram is no longer a "tree"!
("dag" : directed acyclic graph)

```
0 0 0
1 1 1
```

```
d c b a
0 0 0
1 1 1
```
from table to decision diagram

1. eliminate variables with two identical sub trees!
2. identify identical sub trees!
3. identify the 0- and 1-leafs!

the diagram is no longer a “tree”!
("dag" : directed acyclic graph )

after identification of leafs
the diagram has only two leafs left!
from decision diagram to pass network

a realisation is obtained by replacing every node by a 2-selector (or a pair of complementary switches)
from decision diagram to pass network