Multi-valued logic and standard IEEE 1164

Jian-Jia Chen
(Slides are based on
Peter Marwedel)
TU Dortmund,
Informatik 12

© Springer, 2010

2015年 11 月 10 日

These slides use Microsoft clip arts. Microsoft copyright restrictions apply.

Abstraction of electrical signals

- Complete analog simulation at the circuit level would be time-consuming

☞ We try to use digital values and DE simulation as long as possible

☞ However, using just 2 digital values would be too restrictive (as we will see)

How many logic values for modeling ?

Two ('0' and '1') or more?

If real circuits have to be described, some abstraction of the driving strength is required.

☞ We introduce the distinction between:

- the **logic level** (as an abstraction of the voltage) and
- the **strength** (as an abstraction of the current drive capability) of a signal.

The two are encoded in **logic values**.

☞ CSA (connector, switch, attenuator) - theory [Hayes]

1 signal strength

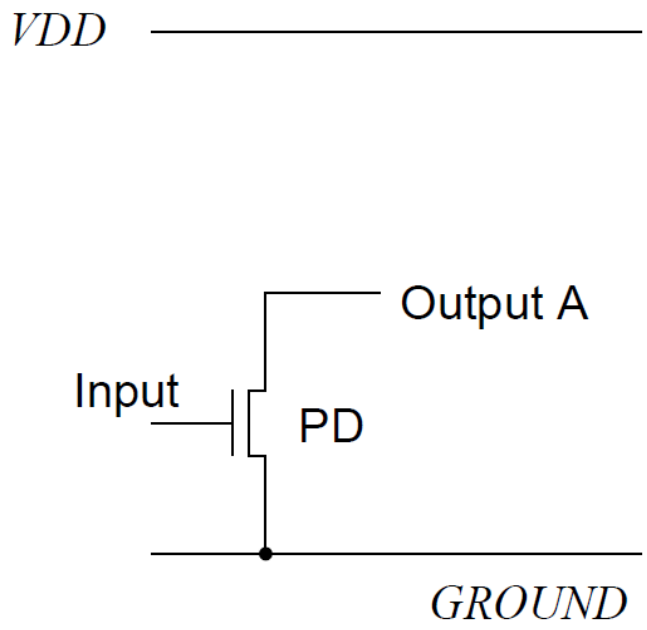
Logic values '0' and '1'.

Both of the same strength.

Encoding false and true, respectively.

2 signal strengths

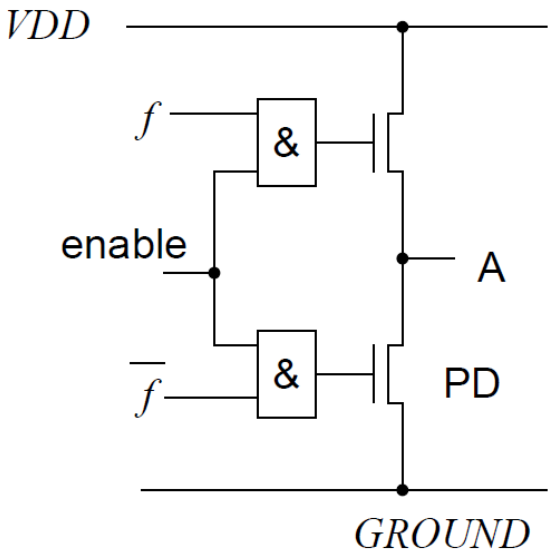
Many subcircuits can effectively disconnect themselves from the rest of the circuit (they provide “high impedance“ values to the rest of the circuit). Example: subcircuits with open collector or tri-state outputs.



Input = ’0’ –> A disconnected

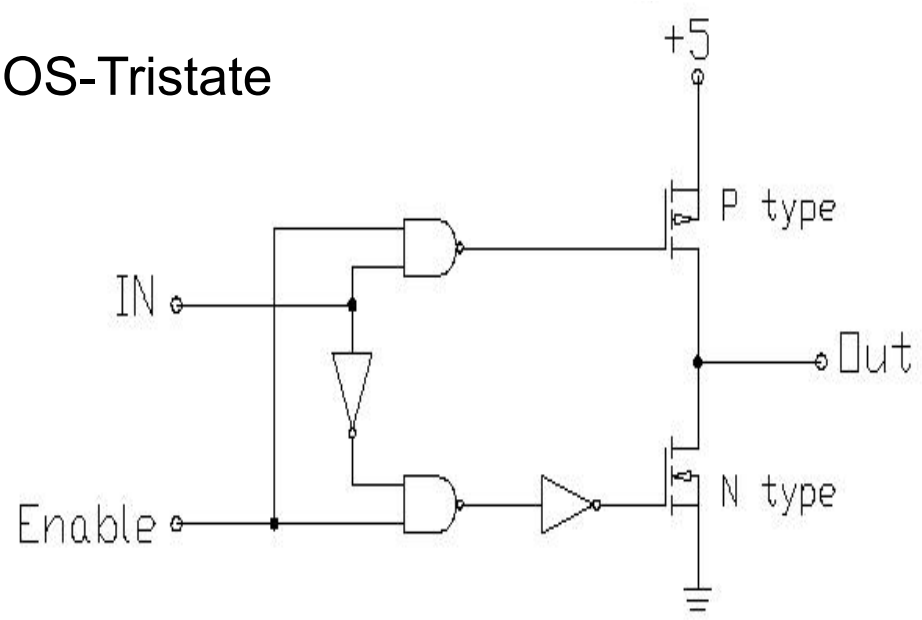
TriState circuits

nMOS-Tristate



enable = ’0’ –> A disconnected

CMOS-Tristate

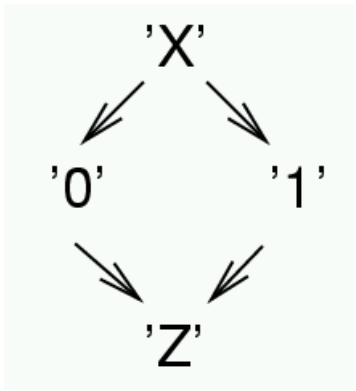


Source: http://www-unix.oit.umass.edu/~phys532/lecture3.pdf

☞ We introduce signal value 'Z', meaning “high impedance“

2 signal strengths (cont'ed)

We introduce an operation #, which generates the effective signal value whenever two signals are connected by a wire. #('0','Z')='0'; #('1','Z')='1'; '0' and '1' are "stronger" than 'Z'



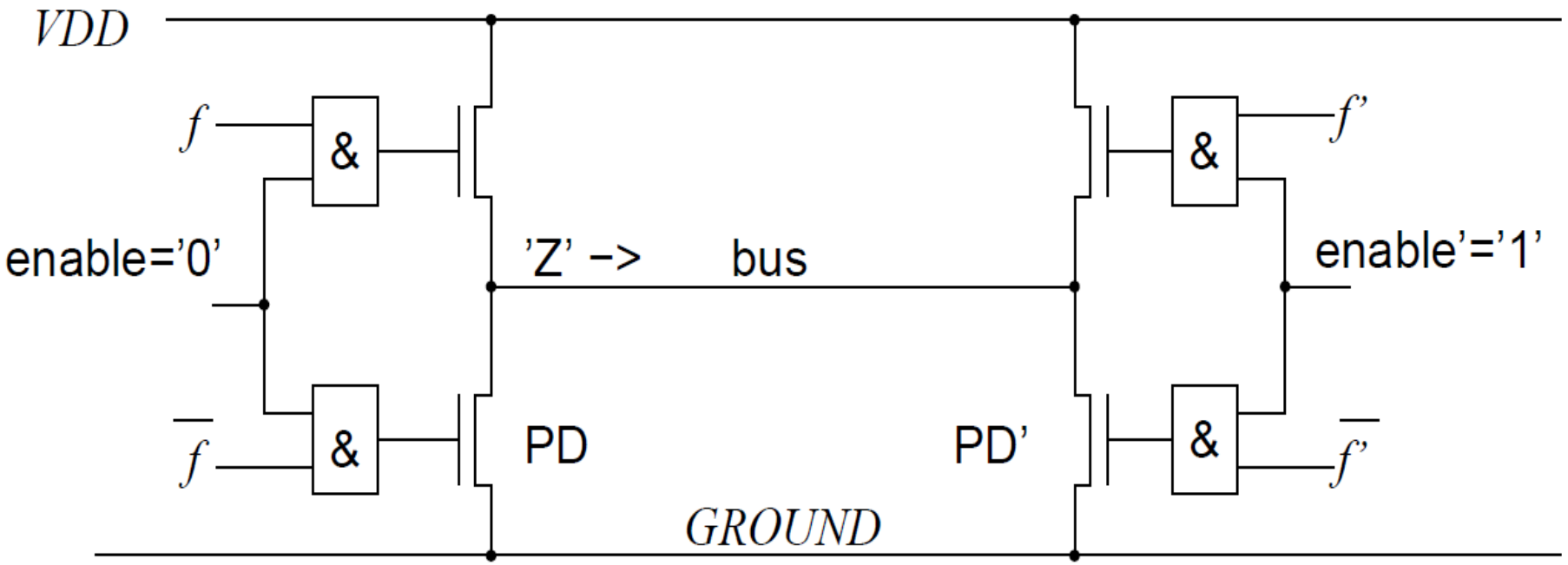
1 strength

Hasse diagram

According to the partial order in the diagram, # returns the smallest element at least as large as the *two arguments* ("Sup").

In order to define #('0','1'), we introduce 'X', denoting an undefined signal level.
'X' has the same strength as '0' and '1'.

Application example



signal value on bus = #(value from left subcircuit, value from right subcircuit)
#('Z', value from right subcircuit)
value from right subcircuit

"as if left circuit were not there".

IEEE 1164

VHDL allows user-defined value sets.

- ☞ Each model could use different value sets (unpractical)
- ☞ Definition of standard value set according to standard IEEE 1164:

{'0', '1', 'Z', 'X', 'H', 'L', 'W', 'U', '-'}

First seven values as discussed previously.

☞: Everything said about 7-valued logic applies.

☞: Combination of pre-charging and depletion transistors cannot be described in IEEE 1164.

'U': un-initialized signal; used by simulator to initialize all not explicitly initialized signals.

Input don‘t care

'–' denotes **input don't care.**

Suppose:

$f(a,b,c) = a\bar{b} + bc$ except for a=b=c='0' where f is undefined

Then, we could like specifying this in VHDL as

```
f <= select a & b & c
     '1' when "10-"    -- first term
     '1' when "-11"    -- second term
     'X' when "000"    -- 'X' ≙ ('0' or '1') here (output don't care)
     '0' otherwise;
```

Simulator would check if a & b & c = "10-", i.e. if c='-'.

Since c is never assigned a value of '-', this test would always fail. Simulator does not know that '–' means either '1' or '0', since it does not include any special handling for '–',

(*at least not for pre-VHDL'2006*).

Function std_match

Special meaning of '–' can be used in special function std_match.

if std_match(a&b&c,"10-")
is true for any value of **c**, but this does not enable the use of the compact **select** statement.

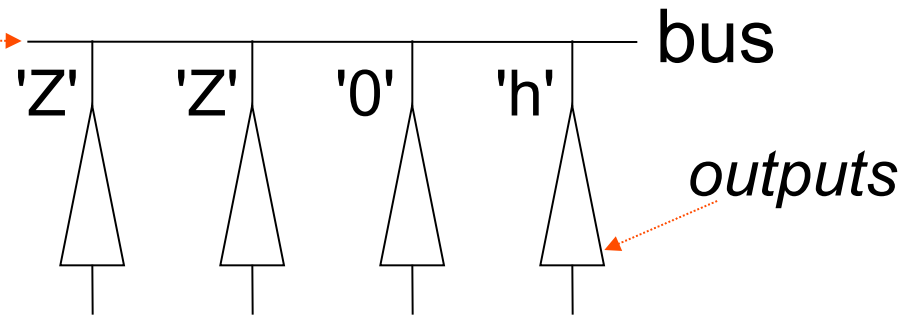
- ☞ The flexibility of VHDL comes at the price of less convenient specifications of Boolean functions.

VHDL'2006 has changed this: '–' can be used in the "intended" way in case selectors

Outputs tied together

In hardware, connected outputs can be used:

Resolution function used for assignments to bus, if bus is declared as std_logic.



Modeling in VHDL: resolution functions

type std_ulogic **is** ('U', 'X','0', '1', 'Z', 'W', 'l', 'h', '-');

subtype std_logic **is** resolved std_ulogic;
-- involve function resolved for assignments to std_logic

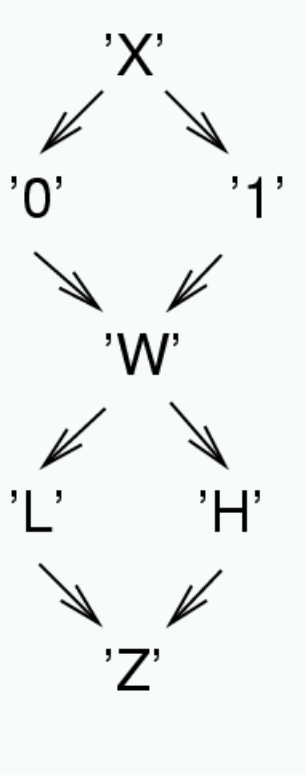
Resolution function for IEEE 1164

```
type std_ulogic_vector is array(natural range<>)of std_ulogic;

function resolved (s:std_ulogic_vector) return …
 variable result: std_ulogic:='Z';   --weakest value is default
  begin
    if (s'length=1) then return s(s'low) --no resolution
    else for i in s'range loop
      result:=resolution_table(result,s(i))
    end loop
    end if;
   return result;
  end resolved;
```

Using # (=sup) in resolution functions

```
constant resolution_table : stdlogic_table := (
--U    X    0    1    Z   W      L     H   –
('U',  'U',  'U', 'U', 'U', 'U',    'U', 'U', 'U'),   --| U |
('U',  'X',  'X', 'X', 'X', 'X',    'X', 'X', 'X'),   --| X |
('U',  'X',  '0', 'X', '0', '0',    '0', '0', 'X'),   --| 0 |
('U',  'X',  'X', '1', '1', '1',    '1', '1', 'X'),   --| 1 |
('U',  'X',  '0', '1', 'Z', 'W',    'L', 'H', 'X'),   --| Z |
('U',  'X',  '0', '1', 'W', 'W',    'W', 'H', 'X'),   --| W |
('U',  'X',  '0', '1', 'L', 'W',    'L', 'W', 'X'),   --| L |
('U',  'X',  '0', '1', 'H', 'W',    'W', 'H', 'X'),   --| H |
('U',  'X',  'X', 'X', 'X', 'X',    'X', 'X', 'X')    --| - |
);
```



This table would be difficult to understand without the partial order

Summary

Discrete event models

- Queue of future events, fetch and execute cycle, commonly used in HDLs
- processes model HW concurrency
- signals model communication
- **wait**, sensitivity lists
- the VHDL simulation cycle
  - ∀ δ cycles, determinate simulation

Multiple-valued logic

- General CSA approach
- Application to IEEE 1164