McGill

ECSE 421 Lecture 6: Discrete Event Models

ESD Chapter 2

Last Time

- Petri Nets
  - Condition/Event Nets
  - Place/Transition Nets
  - Predicate/Transition Nets

Where Are We?

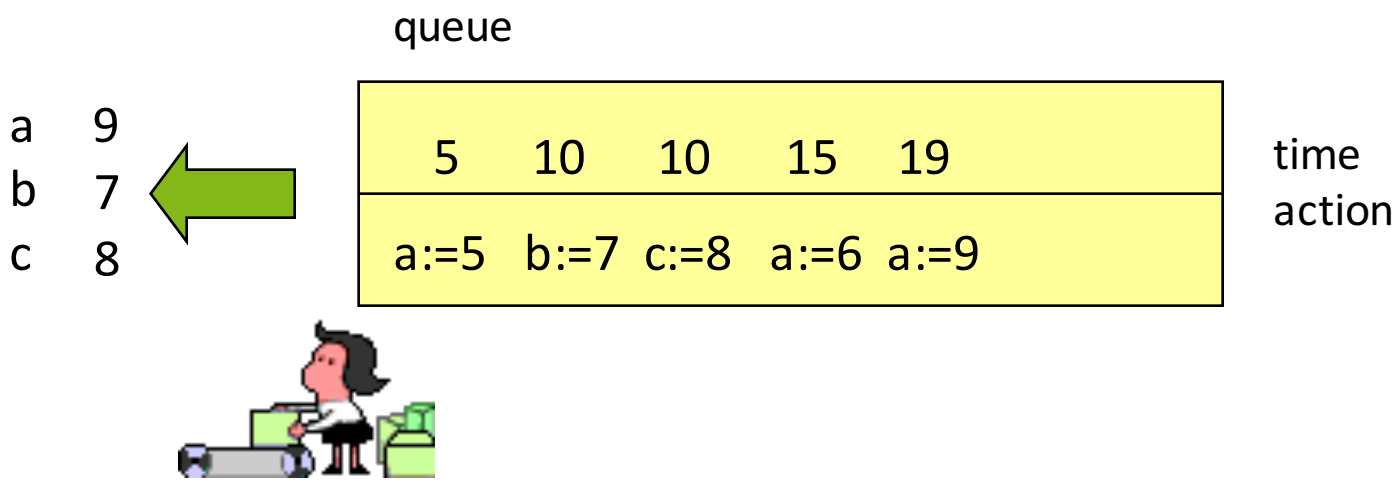
W	D	Date		Topic	ESD	PES	Out	In	Notes
1	T	12-Jan-2016	L01	Introduction to Embedded System Design	1.1-1.4				
	R	14-Jan-2016		Introduction to Embedded System Design	1.1-1.4				
2	T	19-Jan-2016	L02	Specifying Requirements / MoCs / MSC	2.1-2.3				
	R	21-Jan-2016	L03	CFSMs	2.4				
3	T	26-Jan-2016	L04	Data Flow Modeling	2.5	3.1-5,7	LA1		
	R	28-Jan-2016	L05	Petri Nets	2.6				Thru Slide 21
4	T	2-Feb-2016	L06	Discrete Event Models	2.7	4			G: Zaid Al-bayati
	R	4-Feb-2016	L07	DES / Von Neumann Model of Computation	2.8-2.10	5	LA2	LA1	
5	T	9-Feb-2016	L08	Sensors	3.1-3.2	7.3,12.1-6			
	R	11-Feb-2016	L09	Processing Elements	3.3	12.6-12			
6	T	16-Feb-2016	L10	More Processing Elements / FPGAs			LA3	LA2	
	R	18-Feb-2016	L11	Memories, Communication, Output	3.4-3.6				
7	T	23-Feb-2016	L12	Embedded Operating Systems	4.1				
	R	25-Feb-2016		***Midterm exam: in-class, closed book***			P	LA3	Chapters 1-3
	T	1-Mar-2016		***No class***					Winter break
	R	3-Mar-2016		***No class***					Winter break
8	T	8-Mar-2016	L13	Middleware	4.4-4.5				
	R	10-Mar-2016	L14	Performance Evaluation	5.1-5.2				
9	T	15-Mar-2016	L15	More Evaluation and Validation	5.3-5.8				
	R	17-Mar-2016	L16	Introduction to Scheduling	6.1-6.2.2				
10	T	22-Mar-2016	L17	Scheduling Aperiodic Tasks	6.2.3-6.2.4				
	R	24-Mar-2016	L18	Scheduling Periodic Tasks	6.2.5-6.2.6				

MoCs Considered in 421

Communication/ local computations	Shared memory	Message passing Synchronous	Asynchronous
Undefined components	Plain text, use cases | (Message) sequence charts		
Communicating finite state machines	StateCharts		SDL
Data flow	(Not useful)		Kahn networks, SDF
Petri nets		C/E nets, P/T nets, …	
Discrete event (DE) model	**VHDL*, Verilog*, SystemC*, …**	**Only experimental systems, e.g. distributed DE in Ptolemy**	
Von Neumann model	C, C++, Java	C, C++, Java with libraries CSP, ADA |	

Discrete event semantics

- Basic discrete event (DE) semantics
  - Queue of future actions, sorted by time
  - Loop
    - Fetch next entry from queue
    - Perform function as listed in entry
      - May include generation of new entries
    - Repeat until termination criterion is true



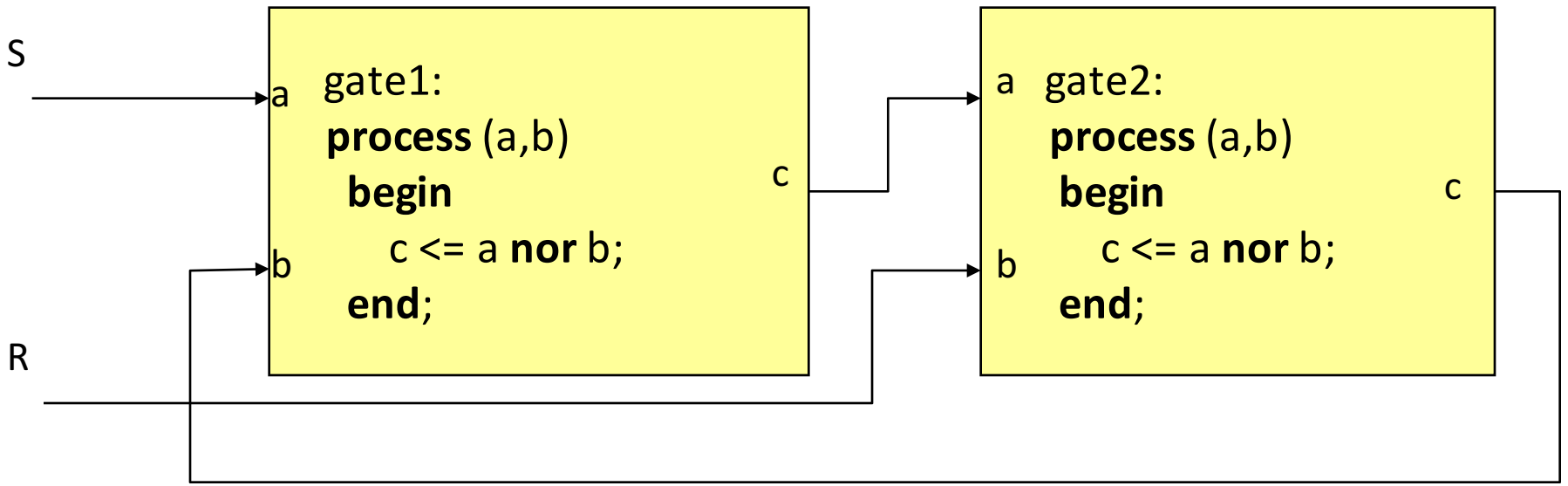
HDLs Use Discrete Event (DE) Semantics

- HW Description Languages (HDLs) must be able to describe concurrency
  - Many HW components are operating concurrently
  - Typically mapped to “processes”
  - These processes communicate via “signals”
- Example:
  - MIMOLA [Zimmermann/Marwedel], ~1975 …
  - …
  - VHDL (very prominent example in DE modeling)
    One of the three most important HDLs:
    VHDL, Verilog, SystemC

VHDL

- VHDL = VHSIC Hardware Description Language
- VHSIC = Very High Speed Integrated Circuit
- 1980: Def. started by US Dept. of Defense (DoD)
- 1984: First version of the language defined
  - Based on ADA,
  - Which in turn is based on PASCAL
- 1987: Revised version became IEEE standard 1076
- 1992: Revised IEEE standard
- 1999: VHDL-AMS: includes analog modeling
- 2006: Major extensions

Simple Example



- Processes wait for changes to values on their input ports
- If changes arrive …
  - Processes wake up,
  - Compute their code,
  - Record changes of output signals in the event queue, and
  - Wait for the next change
- If all processes wait …
  - Time is advanced, and
  - The next entry is taken from the event queue

VHDL Processes

Delays are allowed

Using a sensitivity list:

```
process (a,b)
    begin
        c <= a nor b after 10 ns;
    end;
```

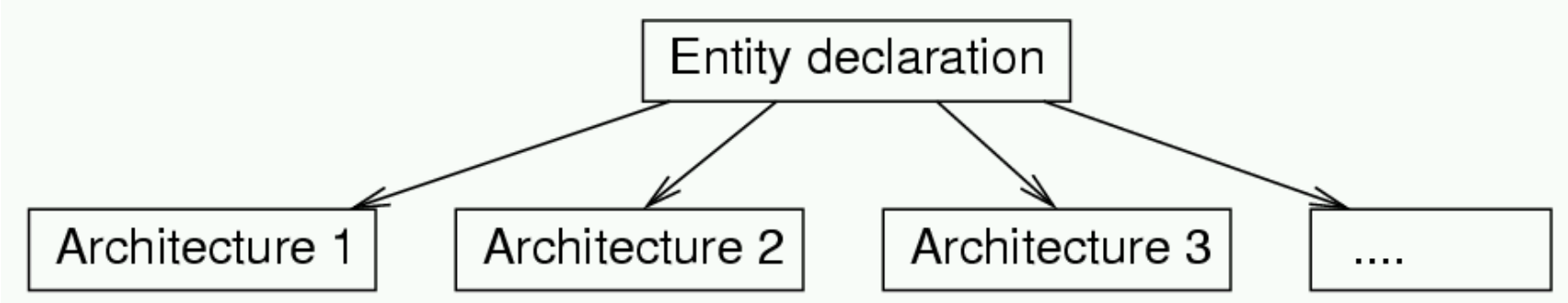
Is equivalent to:

```
process
    begin
        c <= a nor b after 10 ns;
        wait on a,b;
    end;
```

- <=: signal assignment op
- Signal assignments are events
  - Add entries to the projected waveform
  - Delay time is optional
- Implicit loop around the code in the body
- Sensitivity lists
  - Shorthand for a single *wait on* statement at the end of the process body

Entities and Architectures

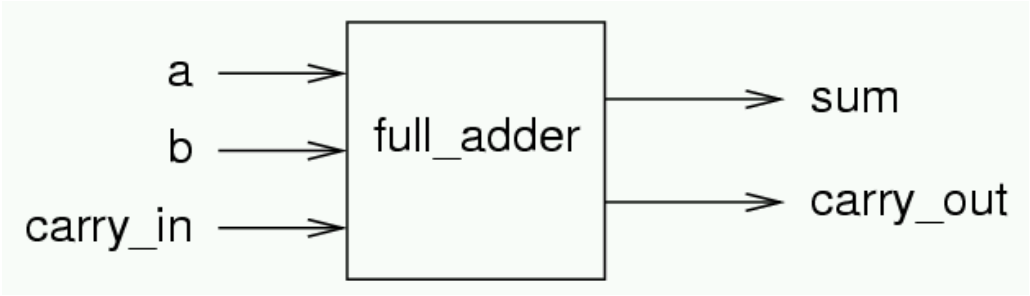
- In VHDL, HW components correspond to "entities"
  - Each design unit is called an entity
  - Entities contain one or more architectures and further entity declarations
  - Entities model concurrency with processes



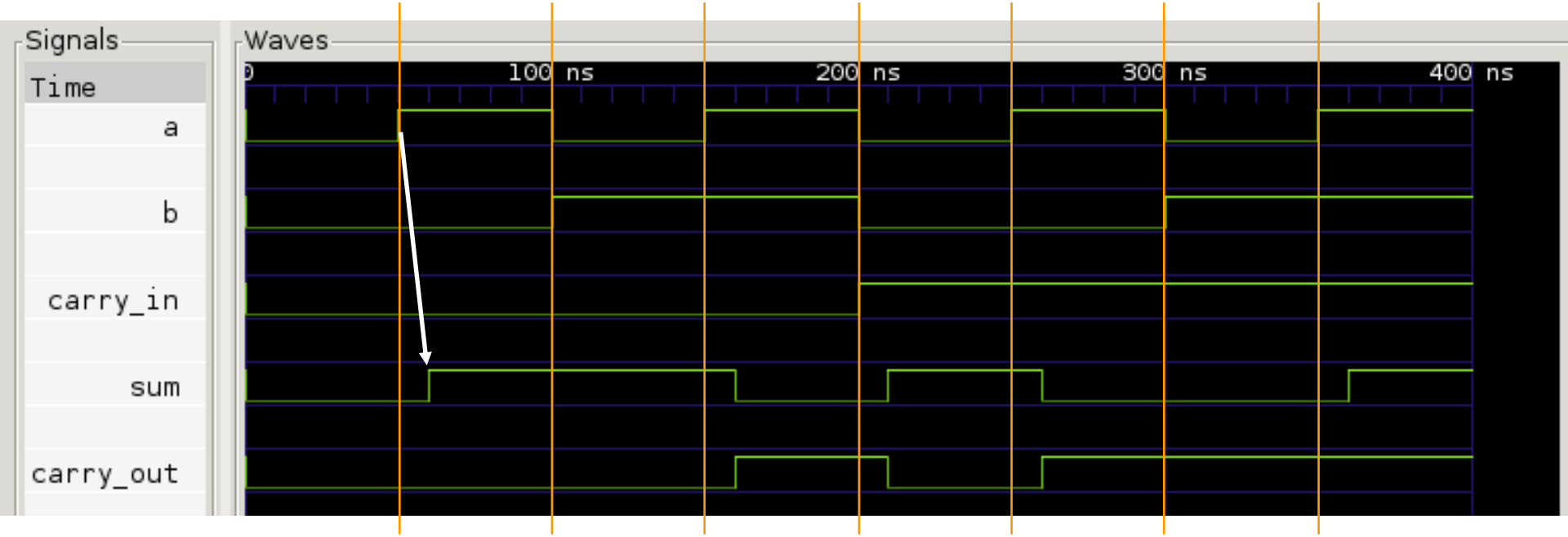
- Each architecture includes a model of the entity
- By default, the most recently analyzed architecture is used
- Another architecture can be requested in a **configuration**

Example: Full Adder

```
entity full_adder is
    port(a, b, carry_in: in Bit;  -- input ports
         sum,carry_out: out Bit); -- output ports
    end full_adder;
architecture behavior of full_adder is
begin
    sum       <= (a xor b) xor carry_in     after 10 ns;
    carry_out <= (a and b) or (a and carry_in) or
                 (b and carry_in)           after 10 ns
end behavior;
```



Example: Full Adder Simulation Results



Assignments

- **Variable assignments**
  Syntax: *variable* := *expression;*
- **Signal assignments**
  Syntax:
  signal <= *expression*;
  signal <= *expression* **after** *delay*;
  signal <= **transport** expression **after** delay;
  signal <= **reject** time **inertial** expression **after** delay;
- One process may make several assignments to the same signal
- For each signal there is one **driver** per process
  The driver stores information about the **future** of signal, the so-called **projected waveform**

Extending the Projected Waveform

- Each executed signal assignment will result in **adding** entries in the projected waveform, as indicated by the delay time, *e.g.*:

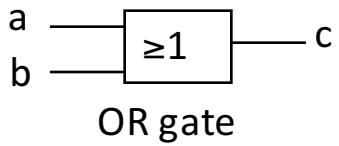
output <= ‘0’ **after** 5 ns, ‘1’ **after** 10 ns;

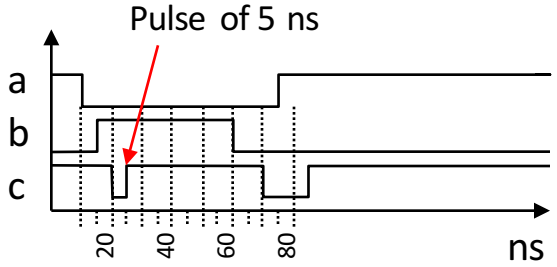
1. Transport Delay

signal <= **transport** *expression* **after** *delay*;

- This corresponds to models for simple wires

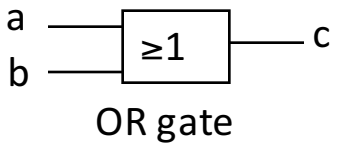
- Pulses are observed, no matter how short
- **Example:**
  - *c* <= **transport** *a* or *b* **after** 10 ns;

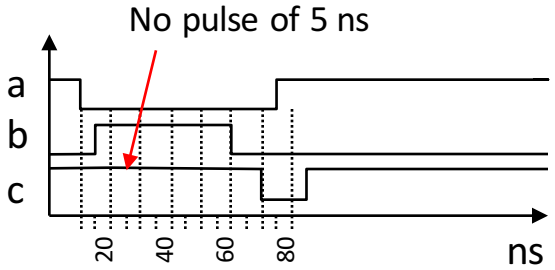




2. Inertial Delay

- By default, inertial delay is assumed
  - Inertial delay models the behavior of gates
- Suppression of all "glitches" shorter than the specified delay
  - Events are removed from the projected waveform
- Example:
  - *c* <= *a* or *b* **after** 10 ns





Wait Statements

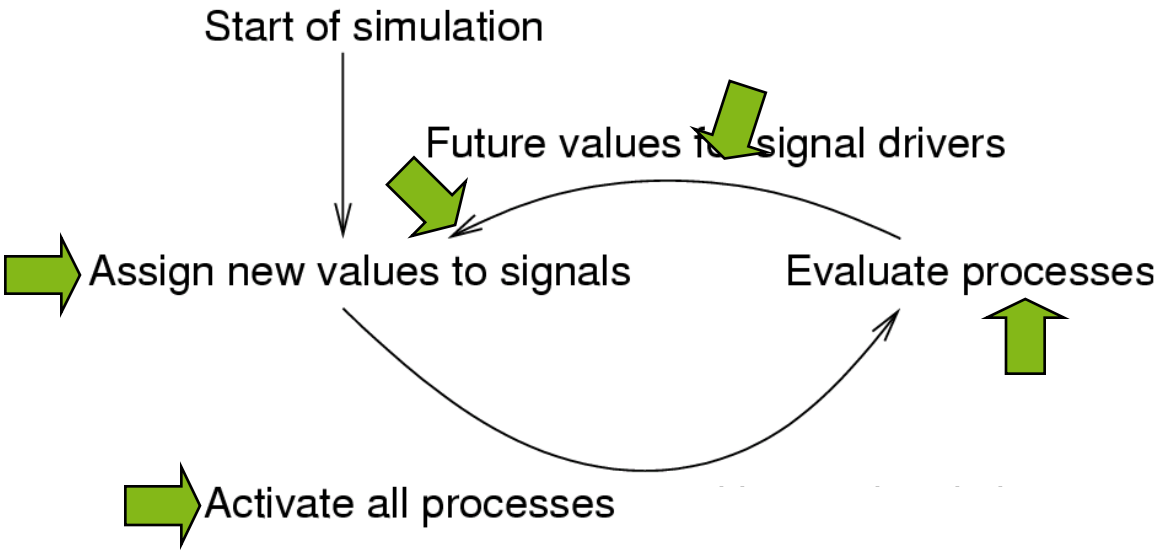
- Four possible kinds of wait-statements:
  - **wait on** *signal* list;
    - wait until signal changes;
    - Example: **wait on** a;
  - **wait until** *condition*;
    - wait until condition is met;
    - Example: **wait until** c='1';
  - **wait for** *duration*;
    - wait for specified amount of time;
    - Example: **wait for** 10 ns;
  - **wait**;
    - suspend indefinitely

VHDL Semantics: Global Control

- According to the original VHDL standard:
  - The execution of a model consists of
    - An initialization phase, followed by
    - The repetitive execution of process statements in the description of that model
  - The initialization phase executes each process once

VHDL Semantics: Initialization

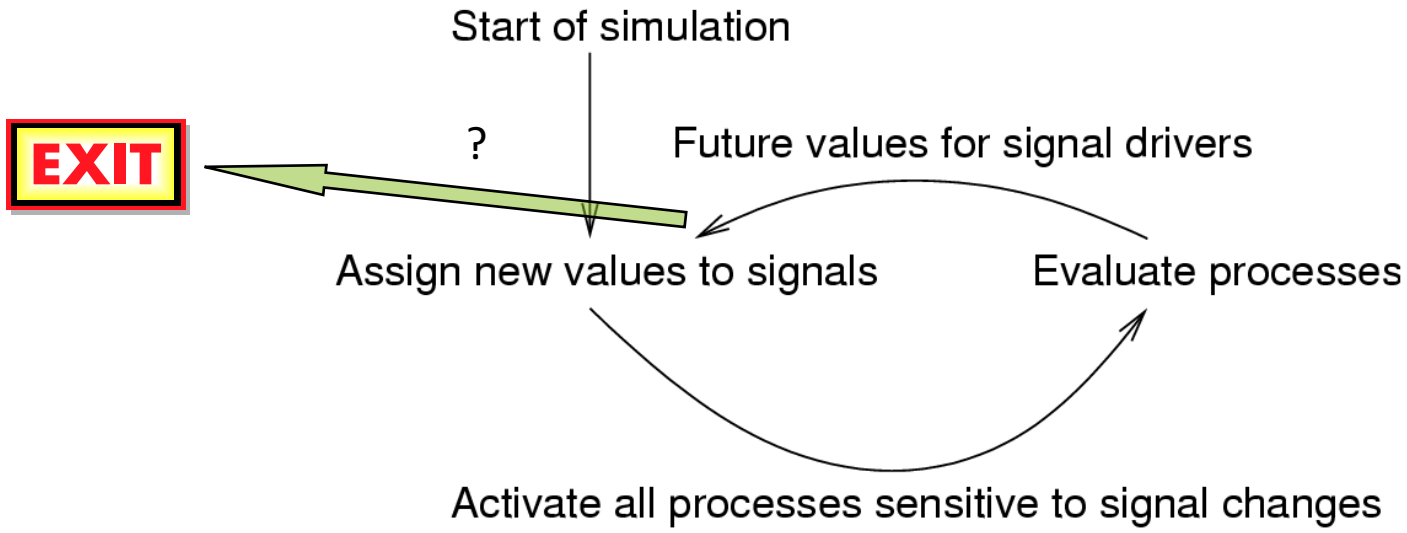
- At the beginning of initialization, the current time, T_c, is 0 ns.
  - *The … effective value of each explicitly declared signal are computed, and the current value of the signal is set to the effective value. …*
  - *Each … process … is executed until it suspends.*
  - *The time of the next simulation cycle (… in this case … the 1st cycle), T_n is calculated according to the rules of step **f** of the simulation cycle, below.*



VHDL Semantics: Simulation Cycle (1)

- According to the standard, the simulation cycle proceeds as follows:

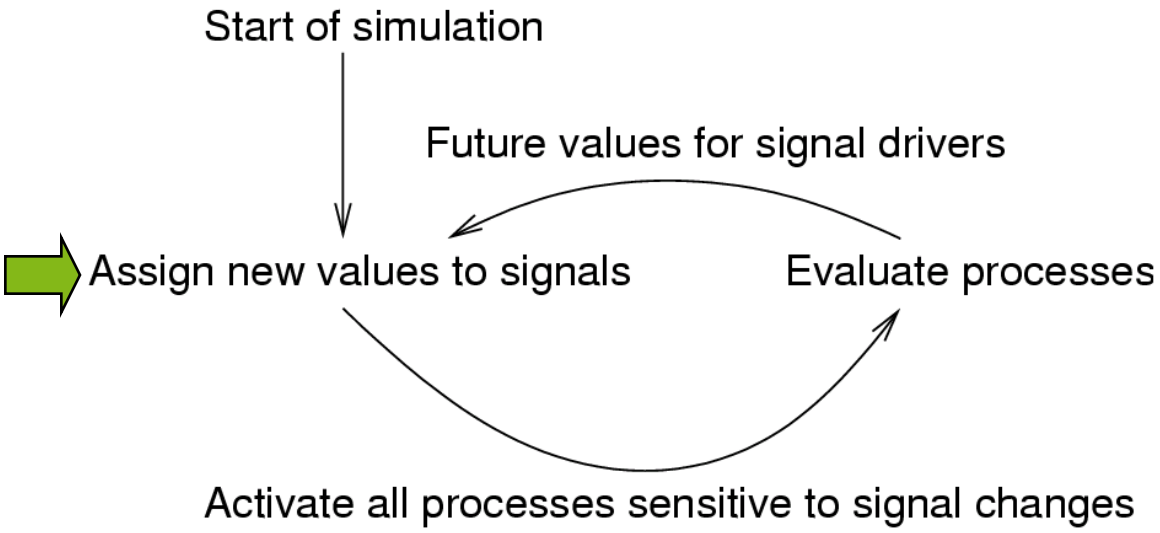
a) The current time, T_c, is set to T_n. Stop if
T_n = TIME'HIGH and
"nothing else is to be done" at T_n.



VHDL Semantics: Simulation Cycle (2)

b) Each active explicit signal in the model is updated. (Events may occur as a result.)
Previously computed entries in the queue are now assigned if their time corresponds to the current time T_c.
New values of signals are not assigned before the next simulation cycle, at the earliest.

Signal value changes result in events, enabling the execution of processes sensitive to that signal.

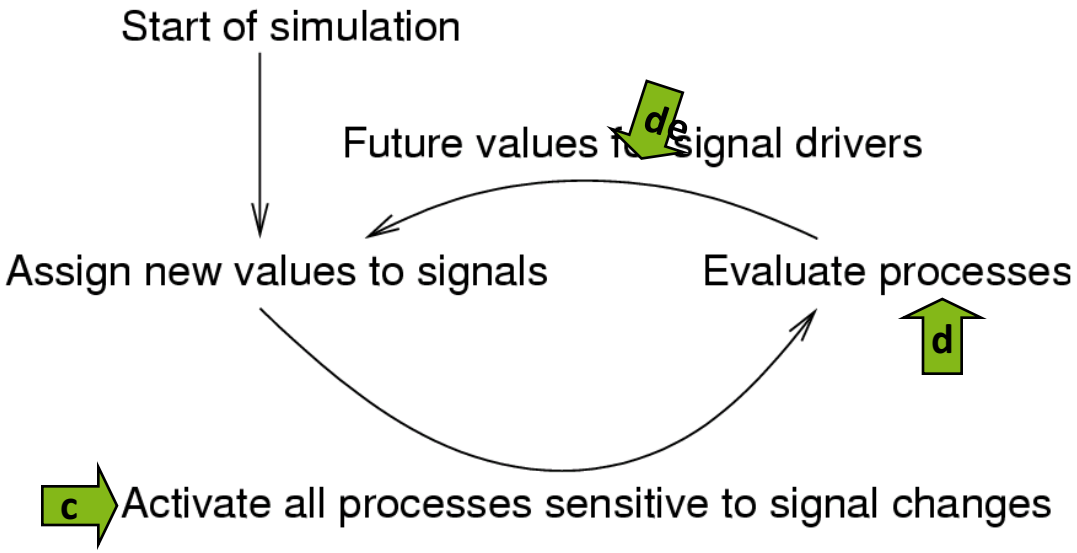


VHDL Semantics: Simulation Cycle (3)

c) $\forall P$ sensitive to s: if there is an event on s in the current cycle: P resumes.

d) Each process that has resumed in the current simulation cycle is executed until it suspends*.

*Generating future values for signal drivers.

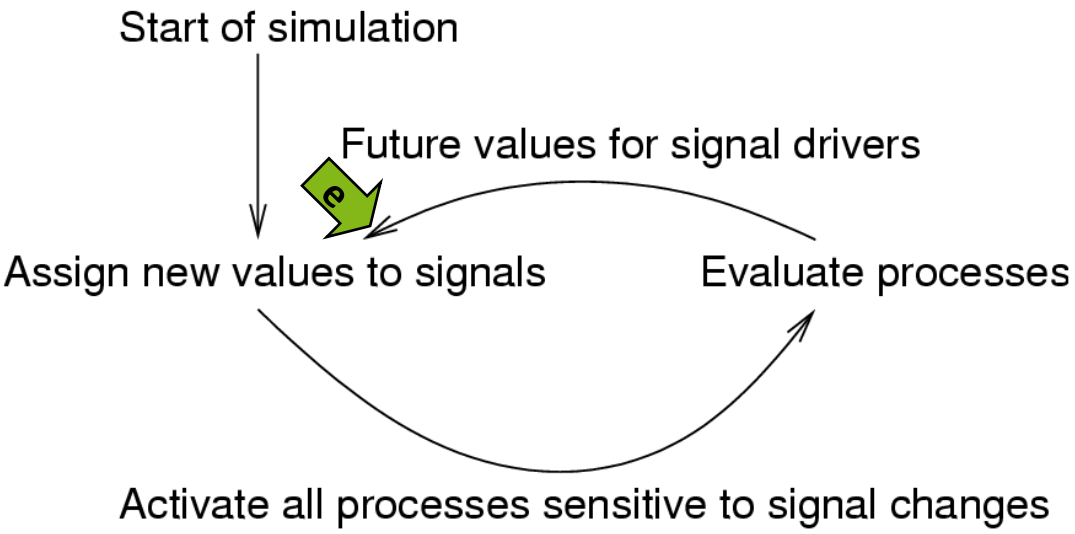


VHDL Semantics: Simulation Cycle (4)

e) Time T_n of the next simulation cycle = earliest of

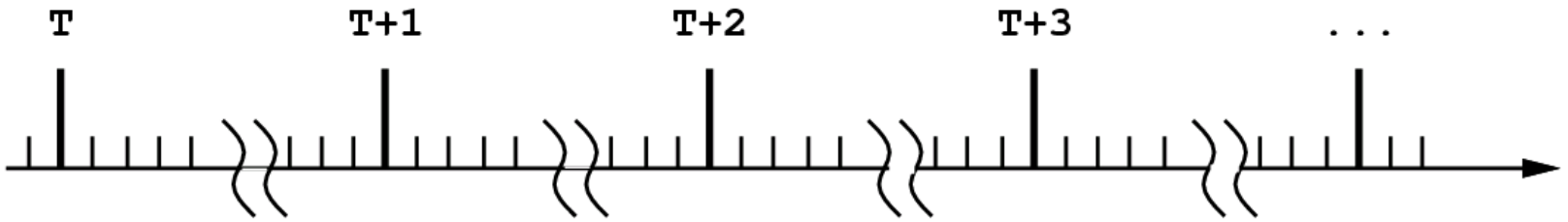
1. TIME'HIGH (end of simulation time)
2. The next time at which a driver becomes active
3. The next time at which a process resumes (determined by **wait for** statements)

Next cycle (if any) will be a delta cycle if $T_n = T_c$

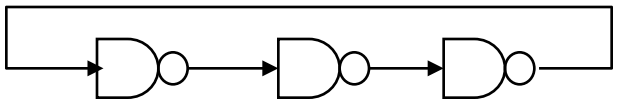


δ-Simulation Cycles

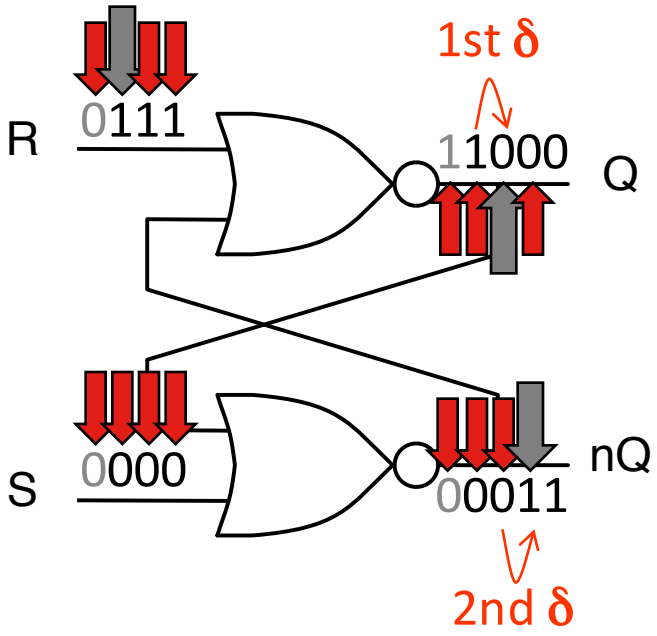
- Delta cycles are generated for delay-less models
- There is an arbitrary number of δ cycles between any 2 physical time instants:



- Simulation of delay-less hardware loops might not terminate (or even advance T_c)



δ-Cycle Example: RS NOR Flip Flop



	0ns	0ns+δ	0ns+2δ	0ns+3δ
R	1	1	1	1
S	0	0	0	0
Q	1	0	0	0
nQ	0	0	1	1

```
gate1:
    process (S,Q)
    begin
        nQ <= S nor Q;
    end;
gate2:
    process (R,nQ)
    begin
        Q <= R nor nQ;
    end;
```

δ cycles reflect the fact that no real gate comes with zero delay.

⇒ should delay-less signal assignments be allowed at all?

δ-Cycles and Determinate Simulation

- Is VDHL determinate?

```
a <= b;
b <= a;
```

- Yes, separation into two simulation phases results in determinate semantics
  - Like StateMate

VHDL: Evaluation

- Behavioral hierarchy
  - Procedures and functions
- Structural hierarchy
  - Through structural architectures,
  - But no nested processes
- No specification of non-functional properties
- No object-orientation
- Static number of processes
- Complicated simulation semantics
- Too low level for initial specification
- Good as an intermediate “Esperanto” or “assembly” language for hardware generation

Abstraction of Electrical Signals

- Complete analog simulation at the circuit level?
  - Impossibly time consuming!
- Use digital values and DES as long as possible
  - Two digital values? Too restrictive!

How Many Logic Values?

- Two ('0' and '1') or more?
- To describe real circuits, some abstraction of the *driving strength* is required

Logic Level and Drive Strength

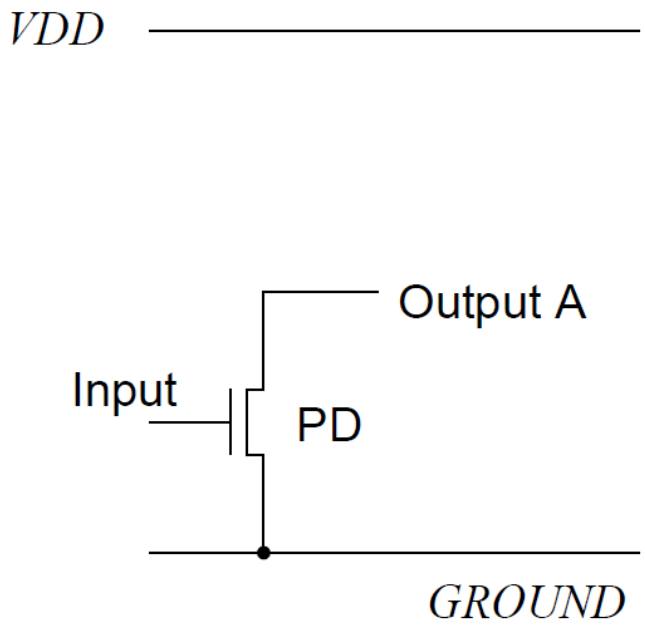
- We therefore distinguish between
  - the **logic level** (as an abstraction of the voltage) and
  - the **strength** (as an abstraction of the current drive capability) of a signal.
- Encoded in logic values
- Output resolution using CSA (connector, switch, attenuator) theory [Hayes]

One Signal Strength

- Logic values ‘0’ and ‘1’
- Both of the same strength
- Encoding false and true, respectively

Two Signal Strengths

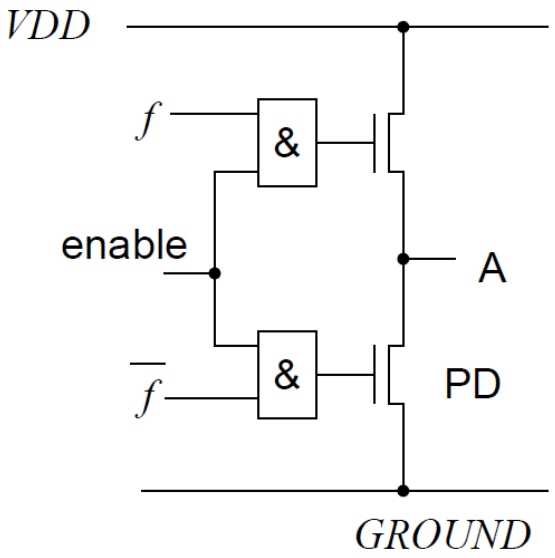
- Many sub-circuits can disconnect themselves
  - "High impedance" output
- Examples:
  - Sub-circuits with open collector
  - Tri-state outputs



Input = '0' -> A disconnected

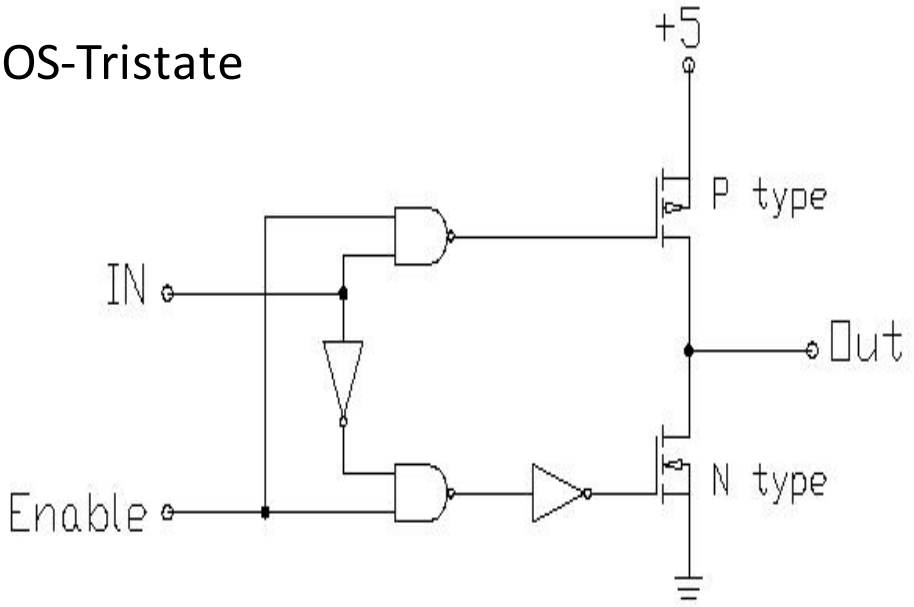
Tri-State Circuits

nMOS-Tristate



enable = '0' –> A disconnected

CMOS-Tristate

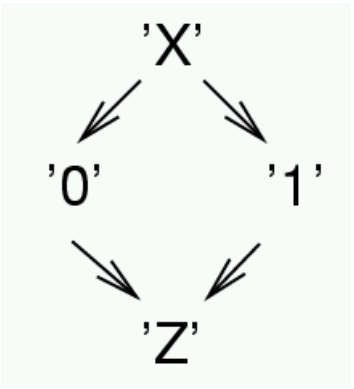


Source: http://www-unix.oit.umass.edu/~phys532/lecture3.pdf

⇒ We introduce signal value 'Z', meaning "high impedance"

Two Signal Strengths, Cont'd

- When a signal is driven at different strengths and with different values, what "wins"?
- We introduce an operation #
  - Generates the effective signal value
  - Useful when two signals are connected by a wire
- #('0','Z')='0'; #('1','Z')='1'; '0' and '1' are "stronger" than 'Z'

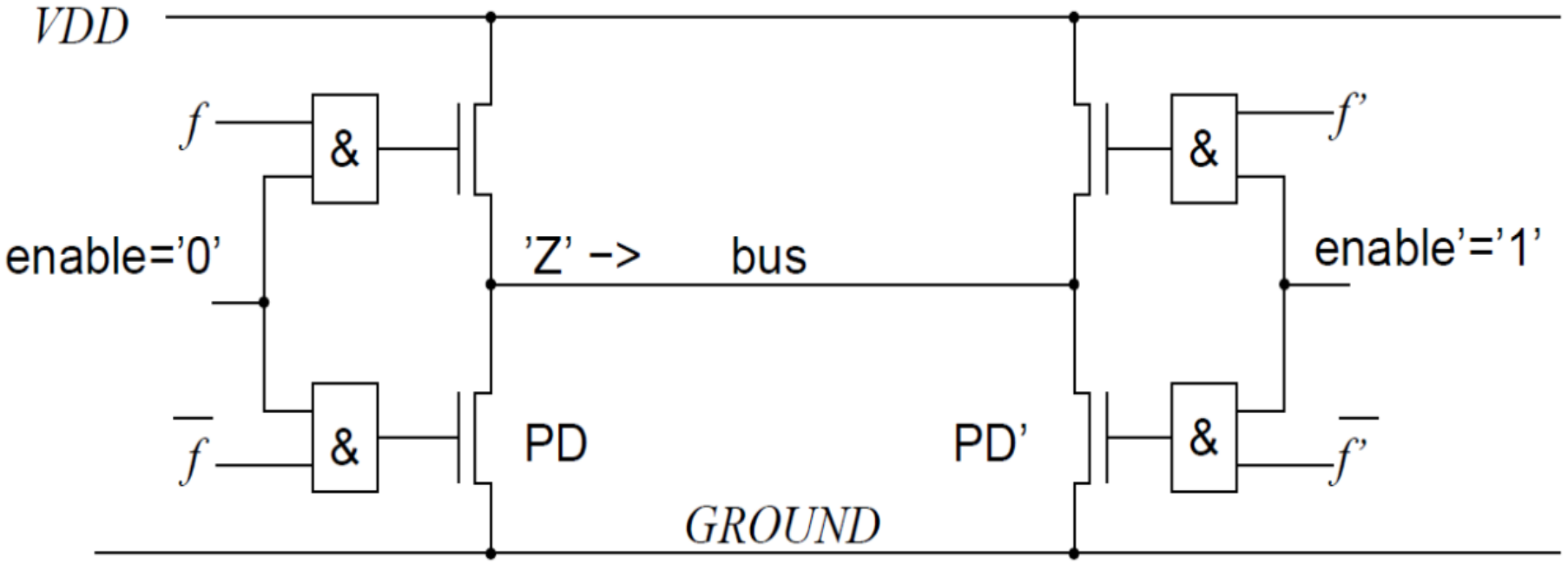


1 strength

Hasse diagram

- # returns the smallest element at least as large as the *two arguments* ("Sup")
- To define #('0','1'), we introduce 'X', denoting an undefined signal level
- 'X' has the same strength as '0' and '1'

Application Example

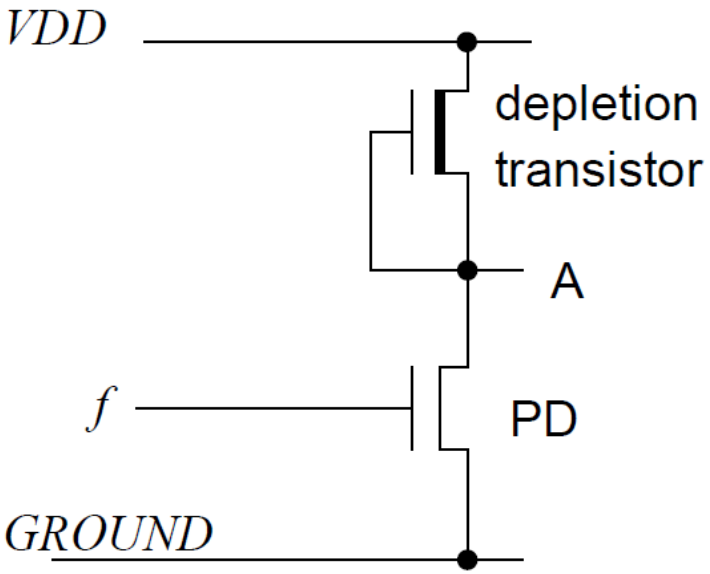


signal value on bus = #(value from left, value from right)
#('Z', value from right)
value from right

Three Signal Strengths

- Current values insufficient for describing real circuits
- Depletion transistor
  - Contributes a weak signal value to A
- We therefore introduce ‘H’
  - A weak signal of level ‘1’
- #('H', '0')='0'; #('H','Z') = 'H'

Inverter implemented in NMOS

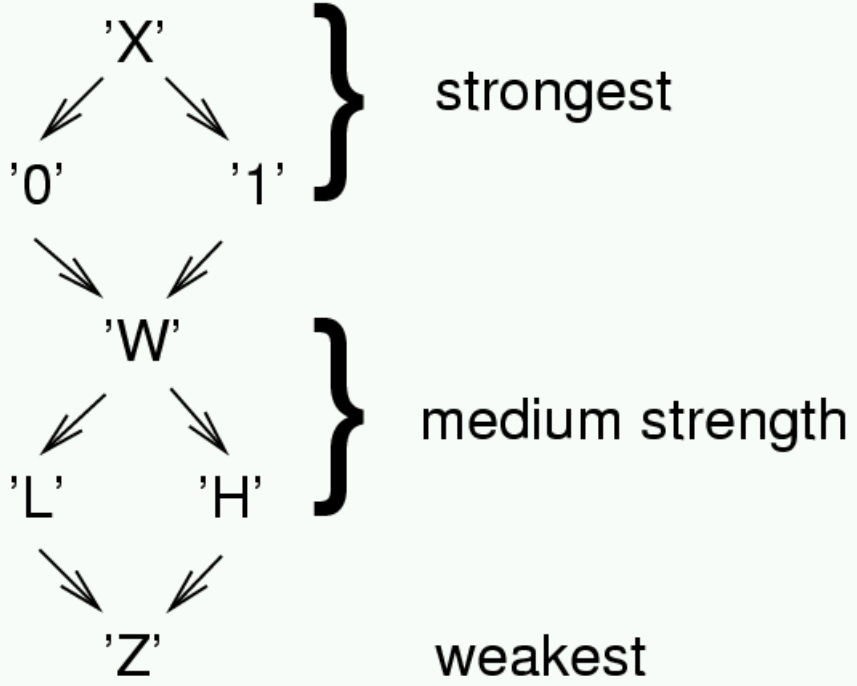


Three Signal Strengths, Cont'd

- Weak signals of level '0'

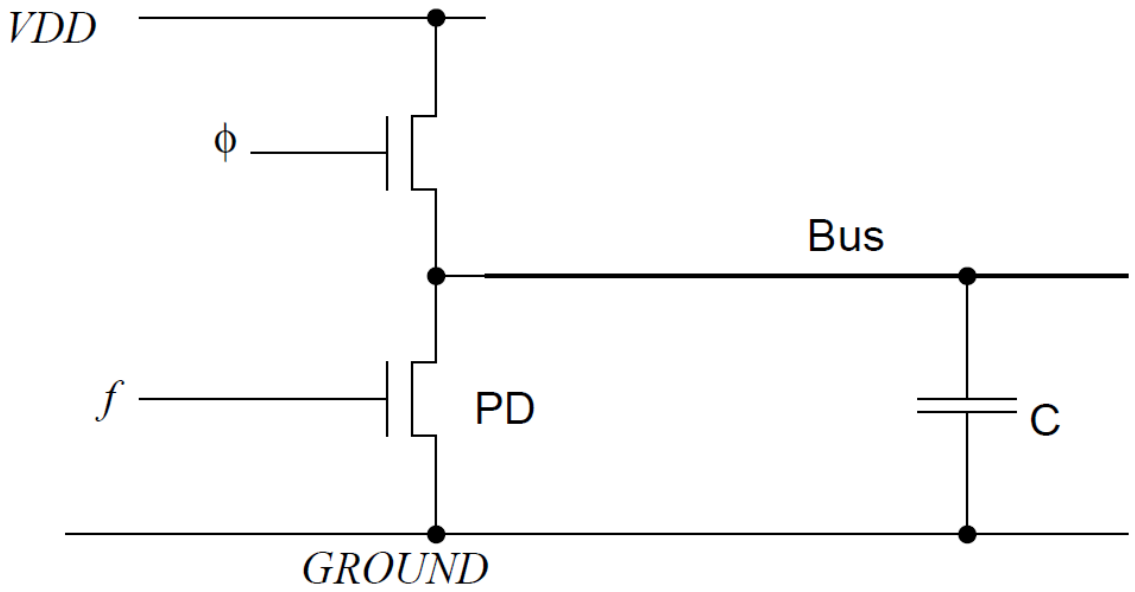
  ⇒ Introduce 'L'
  - A weak signal of level '0'
  - #('L', '1')='1'
  - #('L','Z') = 'L'

  ⇒ Introduce 'W'
  - A weak signal of undefined level 'X'
  - #('L', 'H')='W'
  - #('L','W') = 'W'



Four Signal Strengths

- Current values insufficient for describing pre-charging



- Pre-charged '1'-levels are the weakest so far
  - Except for ‘Z’

  ⇒ Introduction of ‘h’
  - A very weak signal of level ‘1’
  - #('h', '0')='0'; #('h','Z') = 'h'

Four Signal Strengths, Cont'd

- Weaker signals of level '0'

  ⇒ Introduction of 'l'

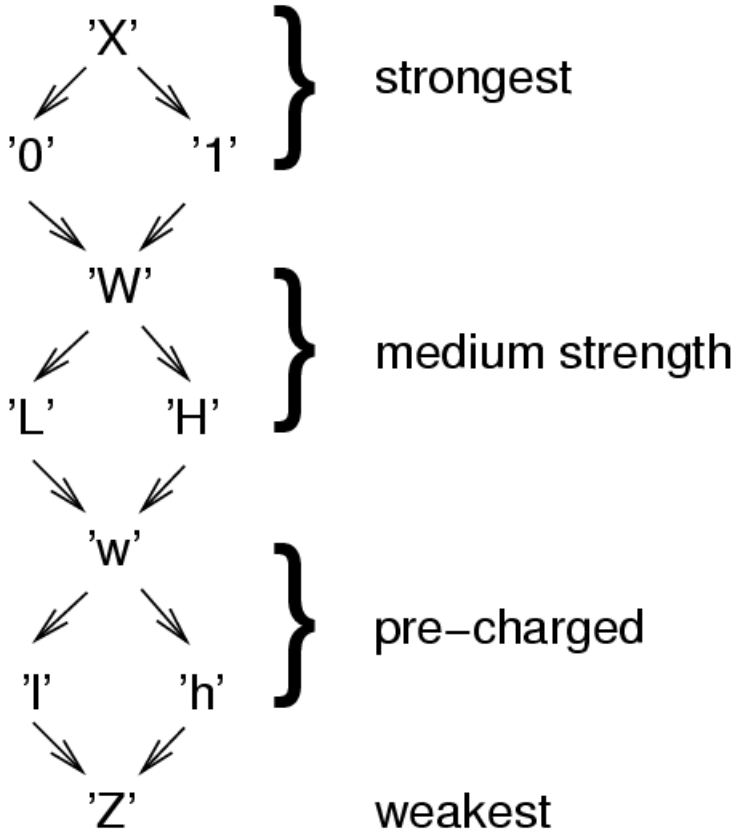
  – Very weak signal, level '0'

  – #('l', '0')='0'; #('l','Z')='l';

  ⇒ Introduction of 'w'

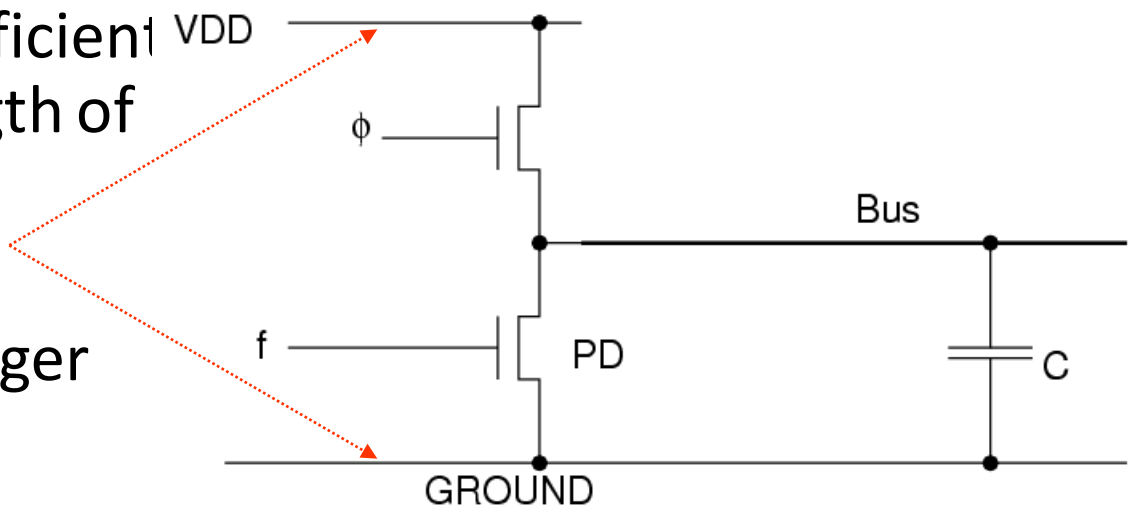
  – Very weak signal, level 'W'

  – #('l', 'h')='w'; #('h','w')='w';



Five Signal Strengths

- Current values insufficient for describing strength of
  - Voltage supply VDD
  - Ground VSS
- Supply voltage stronger than any voltage considered so far



⇒Introduction of 'F0' and 'F1'

- Denoting a very strong signal same of '0 ' and '1'

⇒46-valued logic

- Models uncertainty (Coelho)
- Initially popular, now hardly used

IEEE 1164

- VHDL allows user-defined value sets
  - Each model could use different value sets (unpractical)
- IEEE 1164 formalized a standard set of values
  - {'0', '1', 'Z', 'X', 'H', 'L', 'W', 'U', '-'}
  - First seven values as discussed previously
- Pre-charging and depletion transistors can't be described under IEEE 1164
- 'U': un-initialized signal; used by simulator to initialize all not explicitly initialized signals

IEEE 1164 and 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"   -- 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 always fails!
  - Simulator does not know that '-' means either '1' or '0';
  - It does not include any special handling for '-'!
  - (at least not for pre-VHDL'2006)

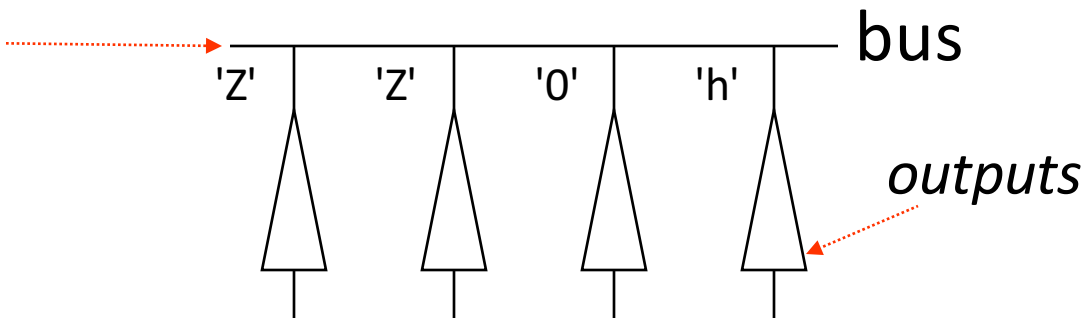
Function std_match

- Special meaning of '-' can be used in special function std_match
- **if** std_match(a&b&c,"10-”)
  - 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

Tying Outputs 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
```

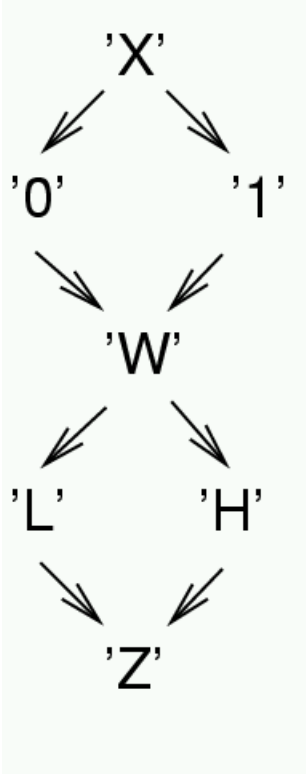
Resolution Function for IEEE 1164

```
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
  - Processes model computation (w/ concurrency)
  - Signals or channels model communication (w/ delay)
  - **wait**, sensitivity lists
  - The VHDL simulation cycle
    - δ-cycles lead to determinate simulation
- Multiple-valued Logic
  - Important abstraction for quickly simulating sophisticated circuits

Next Time

- More DES
  - SystemC
  - Verilog, SystemVerilog
  - SpecC
- Imperative (von Neumann) Models
- Comparing Models of Computation
- Mixing Models of Computation
- Levels of Modeling Abstraction
- Chapters 2.8-10