CPE 487: Digital System Design

Spring 2018

Lecture 4

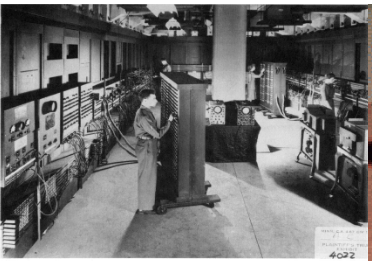
Entities, Architectures & Signals

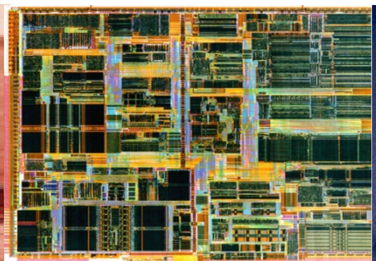
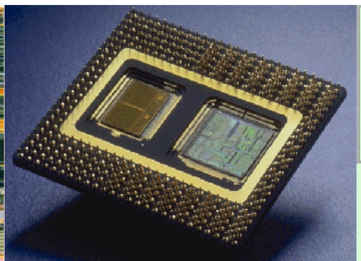
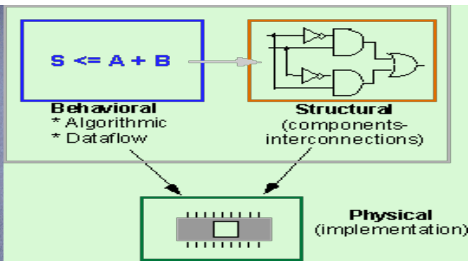
Bryan Ackland

Department of Electrical and Computer Engineering

Stevens Institute of Technology

Hoboken, NJ 07030



Entity Template

- An entity names a system and defines its interface (input/output signals):

```
entity name_of_entity is
    [ generic generic_declarations);]      -- will these cover later
    port ( signal_names: mode type;
           signal_names: mode type;
                :
           signal_names: mode type);
end [entity] [name_of_entity] ;
```

- Entities, signals (and variables, constants, architectures etc.) are named using identifiers

VHDL Basic Identifiers

- an identifier can be any length (but no spaces)
- may contain only alpha-numeric characters (A to Z, a to z, 0-9) and the underscore “_” character
- must start with a letter
- may not end with underscore “_”.
  - no successive underscores“__”.
- VHDL is case insensitive
  - (eg. And2 and AND2 or and2 are the same)
- cannot be a reserved keyword

Identifier Examples

Valid:

Bus, clock, comp1, LED_1, COUNT, cout, c_out, AB2_5C, VHSIC, x1, FFT, decoder, A_B_C, xyZ

Invalid:

1comp, bus_, 2CA, My-name, H$B, _abc, A__B, Decode_, alpha 2, end, AB AC, N#3

Ports

- In an entity description, the port construct defines the input/output signals that make up the system interface
- Each port declaration has the form:

sig_name1, sig_name2, …,sig_name_n : **mode** *type* [:= *init_value*];

- Each port must have a mode and a type
- Can optionally be assigned an initial value
- Mode is one of:
  - **in** signal is an input: read-only port within the architecture
  - **out** signal is an output: write-only port within the architecture
  - **inout** signal can be an input or an output: bidirectional port
  - **buffer** signal is an output whose value can be read inside the entity's architecture

Port Examples

```
entity example is
        port ( a,b: in std_logic:=‘0’;
        c: in integer:= 15;
        x,y: out std_logic;
        z: out std_logic_vector (7 downto 0);
        q: buffer integer);
end entity example;

architecture exarc1 of example is
signal s1, s2: std_logic; signal k: integer;
begin
        s1 <=  a and b;                -- read in ports
        y <= s1 nor a;                 -- assign out port
        q <= c + 3;                    -- assign buffer port
        k <= q – 1;                    -- read buffer port
        s2 <= a xor y;                 -- ERROR – cannot read out port
        b <= s2 and s1;                -- ERROR – cannot assign in port
        z <= c or a;                   -- ERROR – type mismatch
end architecture exarc1;
```

Comments

- A comment line in VHDL is represented by two adjacent hyphens “--”.
- A comment extends from “--” to the end of the line.
- Can appear anywhere within a description;

Examples:

```
--  The following entity is a 32-bit ALU
--  Designer: Fred Bloggs 10/14/08

A <= B and C;   -- A comment explaining this operation

-- X <= Y-15;     -- Commenting out an operation
```

Architecture Template

- An architecture of an entity is one representation of the internal behavior and/or structure of the entity

```
architecture arch_name of entity_name is
    -- Declarations
            -- components declarations
            -- signal declarations
            -- constant declarations
            -- function declarations
            -- procedure declarations
            -- type declarations
begin
    -- Architecture body:
        --Statements that describe behavior and/or structure
end [architecture] arch_name;
```

Architecture Body

- The body of an architecture specifies behavior and/or structure using any of the following modeling styles:

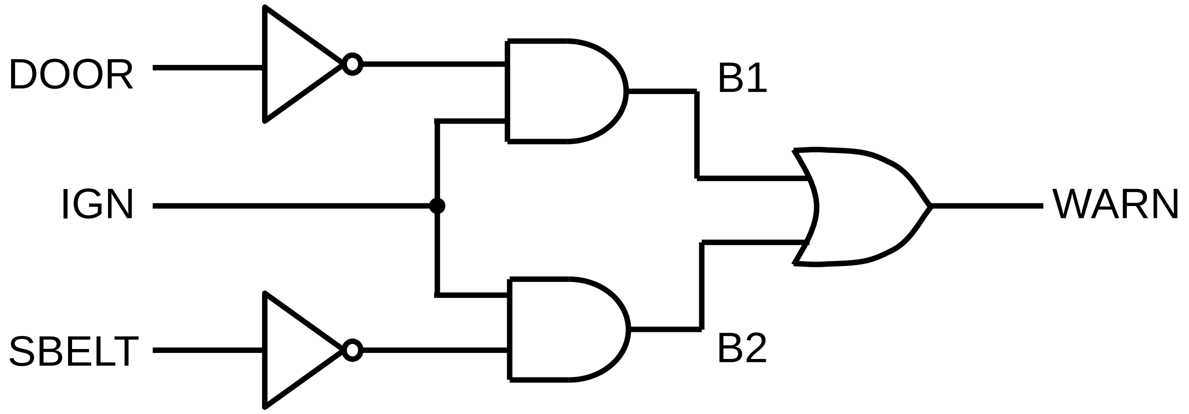
1. A set of concurrent assignment statements (to represent dataflow)

2. A set of sequential assignment statements (to represent behavior)

3. A set of interconnected components (to represent structure

4. Any combination of the above three

Putting it All Together…



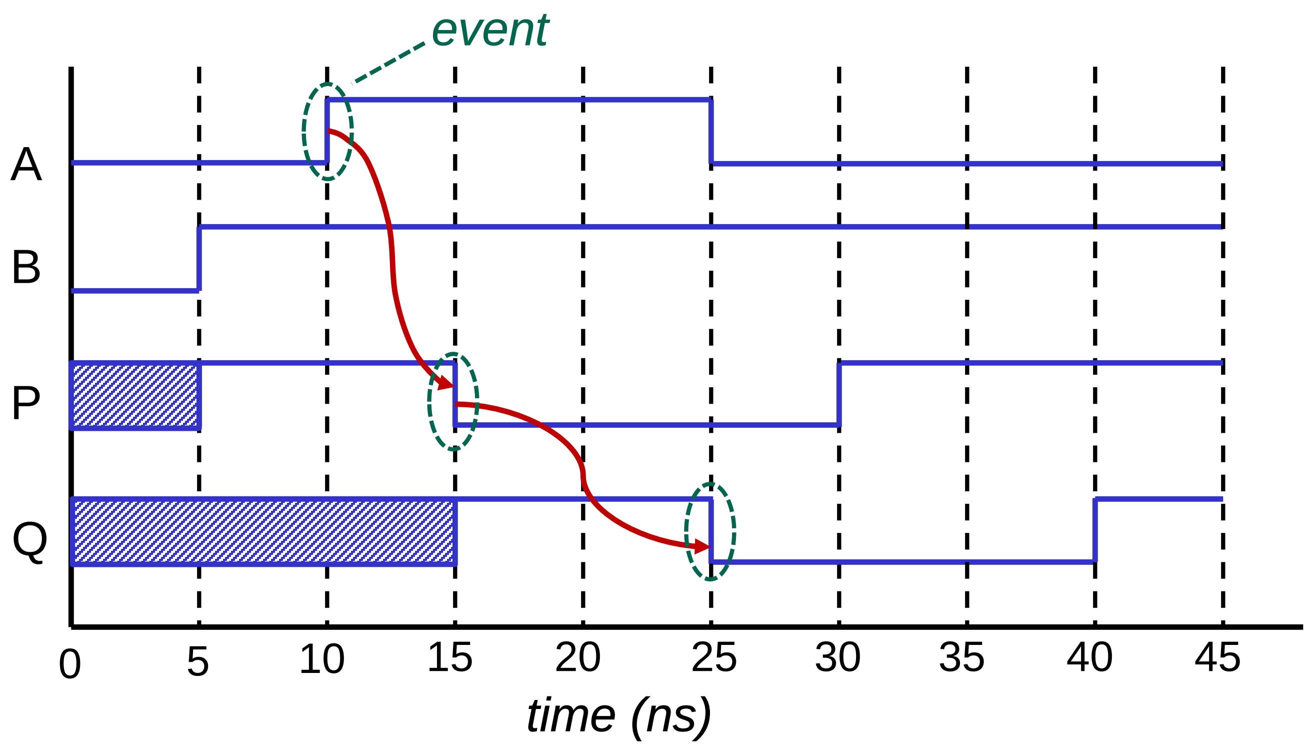
```
library IEEE;
use IEEE.std_logic_1164.all;

entity sb_alarm is
        port ( door, ign, sbelt:in std_logic;
        warn: out std_logic);
end entity sb_alarm;
```

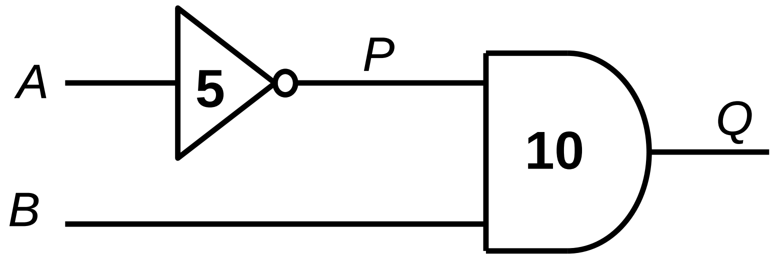
```
architecture gates of  sb_alarm is
signal b1, b2: std_logic;
begin
        b1<= (not door) and ign after 7 ns;
        b2<=(not sbelt) and ign after 7 ns;
        warn <= b1 or b2 after 9 ns;
end architecture gates;
```

Concurrent Operation

- Conventional programming languages operate on variables with no intrinsic concept of time
  - Execution order is determined by order of programming statements (+ branch, subroutine calls etc.)
  - Only one operation is happening at a time
- Hardware is concurrent
  - All components execute in parallel
  - In digital circuits, output signals change in response to changes in input signals
  - When a signal changes, we say that an event occurs on that signal

Events in Digital Circuits





```
P <= not A after 5 ns;
Q <= P and B after 10 ns;
```

Concurrent Signal Assignment Statements

- Signal assignment statements in the body of the architecture are called concurrent signal assignment statements (CSA's), e.g.:

```
architecture abc of xyz is
…
begin
    …
    a <= b and c after 10 ns;
    …
end architecture abc;
```

- This CSA will be executed whenever an event occurs on *b* or *c*
  - Suppose b changes at time 100ns.
  - The value (b **and** c) will be calculated
  - This new value will be assigned to *a* at (100ns + 10ns) = 110ns

CSA Execution triggered by Events

a <= b **and** c **after** 10 ns;
x <= a **or** c **after** 20 ns;

is the same as:

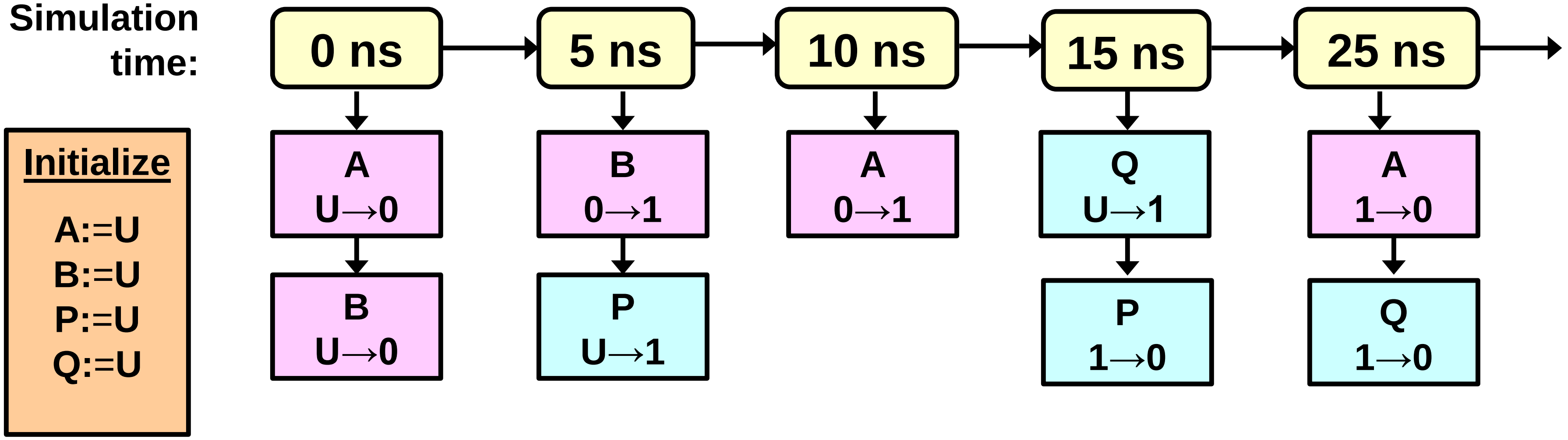
x <= a **or** c **after** 20 ns;
a <= b **and** c **after** 10 ns;

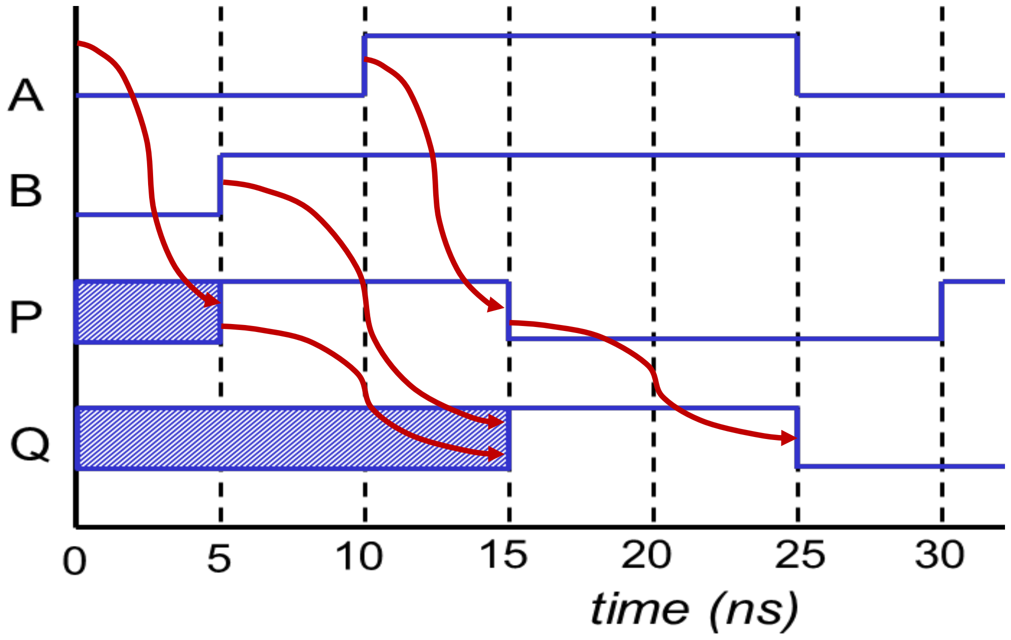
- order of concurrent signal assignments is not important
  - An event on c at 100ns may lead to an event on a at 110ns and an event on x at 120ns
  - An event on a at 110 ns may, in turn, lead to a second event on x at 130 ns
  - Note that a signal assignment only generates a new event if the value of the signal changes

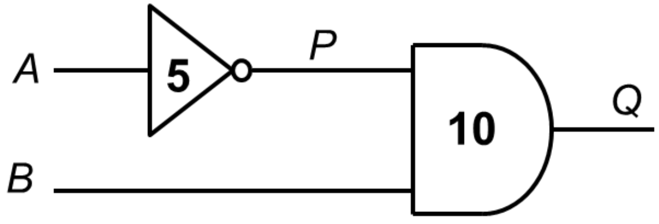
Event Driven Simulation Model

- VHDL is simulated using event driven simulation
- Simulator maintains variable current time
- Simulator also maintains event queue
  – Time ordered list of all future events
  – Each event consists of:
    - name of signal
    - change in value
    - time at which the change takes place
- Operation of simulator:
  1. Advance *current time* to earliest event on event queue
  2. Adopt all events at *current time* (i.e. update signals)
  3. Execute simulation models affected by these events
  4. Schedule future events generated by these models
  5. Repeat until queue is empty or time-limit reached

Event Driven Simulation of Simple Circuit







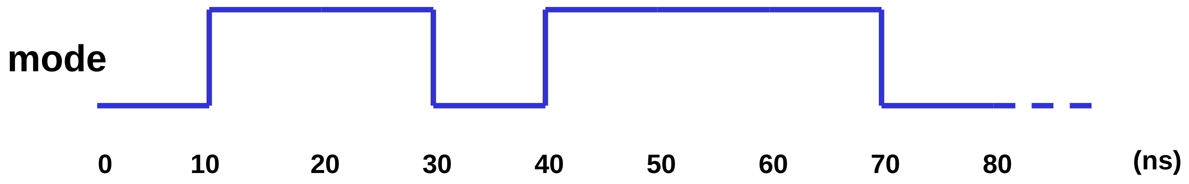
P <= **not** A **after** 5 ns;

Q <= P **and** B **after** 10 ns;

Multiple Event Generation

- Can a signal assignment statement generate multiple events? YES

signal mode**: bit;**
mode<=‘0’,‘1’ **after** 10ns,‘0’ **after** 30ns,‘1’ **after** 40ns,‘0’ **after** 70ns;

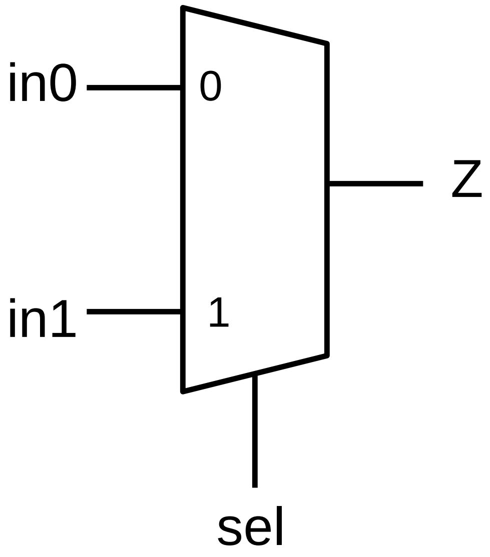


- can be useful for generating input waveforms
- another example:

s1 <= **(**x **xor** y) **after** 5ns, (x **or** y) **after** 10ns, **(not** x) **after** 15ns;

Example – 2-way multiplexer

Write a gate level VHDL model of a 1-bit 2-way multiplexer using concurrent signal assignment statements. Assume the only gates you have available to you are 2-input NAND gates and NOT gates. Use std_logic type for each input and output. Assume all gate delays are 5ns.



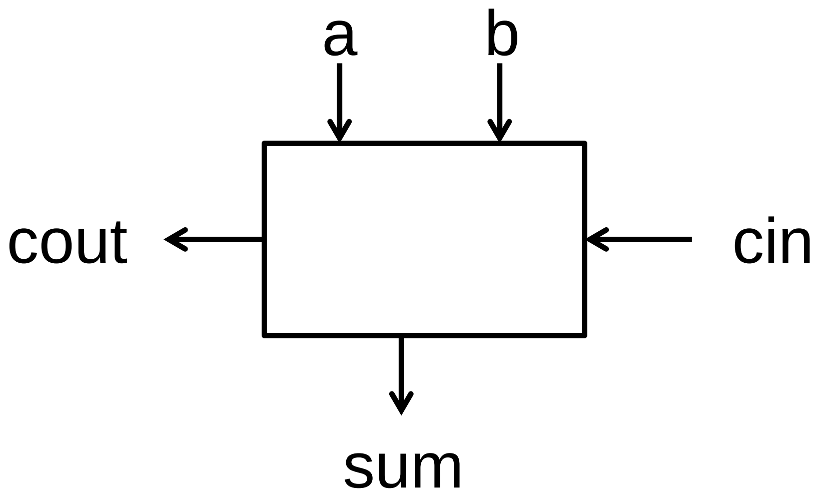
in0	in1	sel	Z
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

Constants

- Only data object we have considered so far is signal
- VHDL statements operate on 4 basic classes of objects
  - Signals
  - Constants
  - Variables (later)
  - Files (later)
- Constants are objects that are assigned a value once, when declared, and do not change their value during simulation.
- Constants are useful for creating more readable design descriptions, and they make it easier to change the design at a later time.
- Examples:
  - **constant** delay: **time** := 10 ns;
  - **constant** bus_width : **integer** := 8;

Example – full adder

Write a gate level VHDL model of a full adder using concurrent signal assignment statements. Use std_logic type for each signal. Use constant to define all gate delays as 2.5ns



a	b	cin	sum	cout
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1