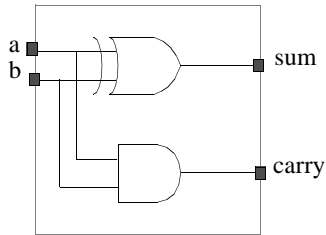
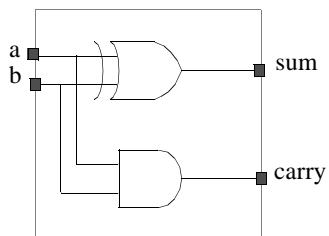


Basic Language Concepts



- Primary programming abstraction is a *design entity*
 - register, logic block, chip, board, or system
- What aspects of a digital system do we want to describe?
 - interface: how do we connect to it
 - function: what does it do?
- VHDL 1993 vs. VHDL 1987

Describing the Interface: The Entity Construct



```
entity half_ADDer is
  port ( a, b : in bit;
         sum, carry :out bit);
end entity half_adder;
```

case insensitive

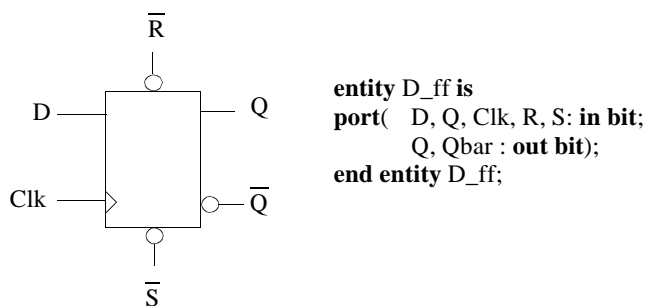
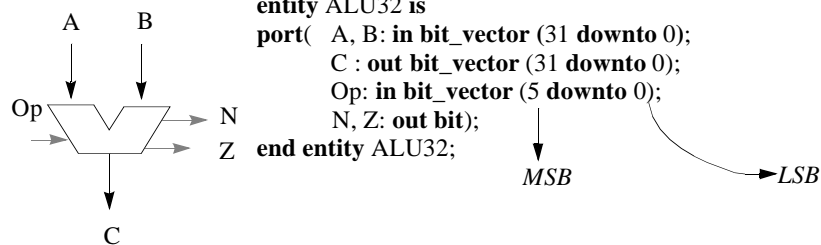
VHDL 1993

- The interface is a collection of *ports*
 - ports are a new programming object: *signal*
 - ports have a type, e.g., **bit**
 - ports have a mode: in, out, inout (bidirectional)

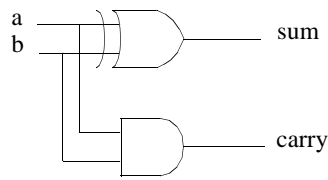
The Signal Object Type

- VHDL supports four basic objects: variables, constants, signals and file types (1993)
- Variable and constant types
 - a single value
- The signal object type is motivated by digital system modeling
 - distinct from variable types in the association of time with values
 - implementation of a signal is really a sequence of time-value pairs!

Example Entity Descriptions



Describing Behavior: The Architecture Construct



```

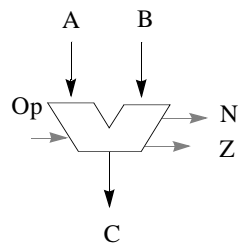
entity half_adder is
port (a, b : in bit;
        sum, carry :out bit);
end entity half_adder;

architecture behavioral of half_adder is
begin
    sum <= (a xor b) after 5 ns;
    carry <= (a and b) after 5 ns;
end architecture behavior;
    
```

VHDL 1993

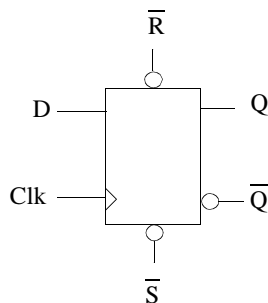
- Description of events on output signals in terms of events on input signals: the *signal assignment statement*
- Specification of propagation delays
- Type **bit** is not powerful enough for realistic simulation: use the IEEE 1164 value system

Example Entity Descriptions: IEEE 1164



```

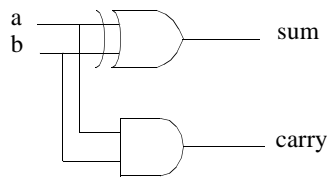
entity ALU32 is
port( A, B: in std_ulogic_vector (31 downto 0);
        C : out std_ulogic_vector (31 downto 0);
        Op: in std_ulogic_vector (5 downto 0);
        N, Z: out std_logic);
end entity ALU32;
    
```



```

entity D_ff is
port( D, Q, Clk, R, S: in std_ulogic;
        Q, Qbar : out std_ulogic);
end entity D_ff;
    
```

Describing Behavior: The Architecture Construct



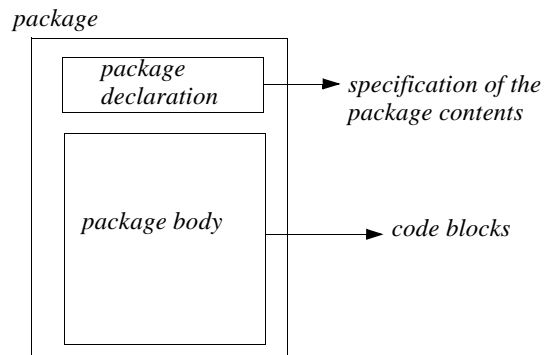
```
library IEEE;  
use IEEE.std_logic_1164.all;  
  
entity half_adder is  
port (a, b : in std_ulogic;  
      sum, carry :out std_ulogic);  
end half_adder;
```

```
architecture behavioral of half_adder is  
begin  
sum <= (a xor b) after 5 ns;  
carry <= (a and b) after 5 ns;  
end architecture behavior;
```

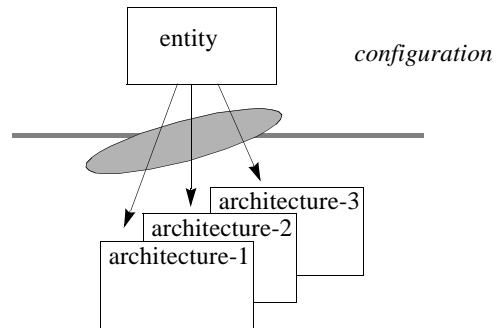
- Use of the IEEE 1164 value system simply requires inclusion of the library and package declaration statements

Libraries and Packages

- Libraries are logical units that are mapped to physical directories
- Packages are repositories for type definitions, procedures, and functions



Configurations



- An entity may have multiple architectures
- Separation of interface from implementation
- Binding rules: default and explicit

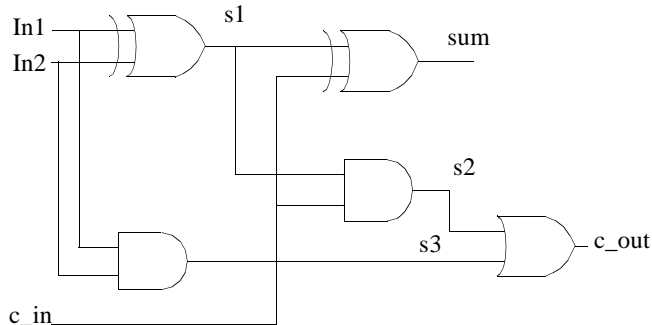
Design Units

- Primary design units
 - Entity
 - Configuration
 - Package Declaration
- Secondary design units
 - Package body
 - Architecture
- Design units are created in design files
- Now you know the layout of a VHDL program!

Simple Signal Assignment

```
library IEEE;
use IEEE.std_logic_1164.all;
entity full_adder is
port (in1, in2, c_in: in std_logic;
      sum, c_out: out std_logic);
end entity full_adder;

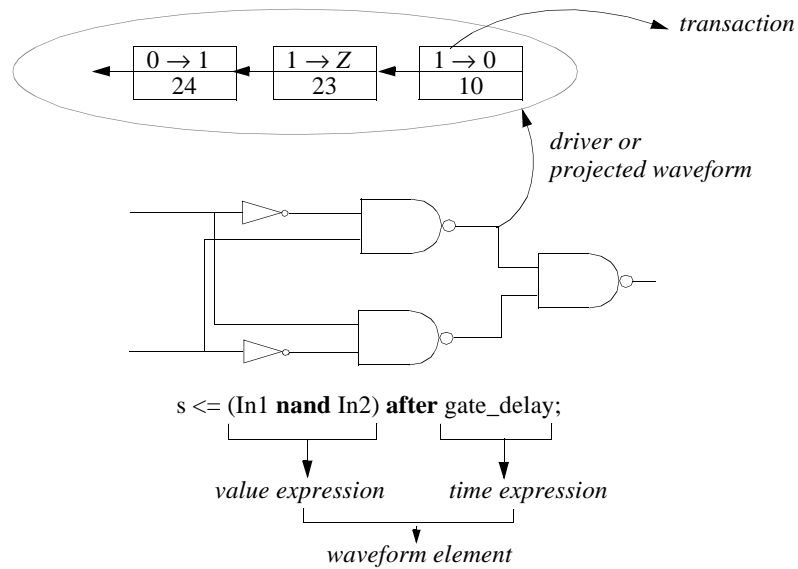
architecture dataflow of full_adder is
signal s1, s2, s3 : std_logic;
constant gate_delay: Time:= 5 ns;
begin
L1: s1 <= (In1 xor In2) after gate_delay;
L2: s2 <= (c_in and s1) after gate_delay;
L3: s3 <= (In1 and In2) after gate_delay;
L4: sum <= (s1 xor c_in) after gate_delay;
L5: c_out <= (s2 or s3) after gate_delay;
end architecture dataflow;
```



Simple Signal Assignment Statement

- The *constant* programming object
 - values cannot be changed
- Use of signals in the architecture
 - internal signals used to connect components
- A statement is executed when a signal in the RHS has value assigned to it
 - 1-1 correspondence between signal assignment statements and signals in the circuit
 - order of statement execution follows propagation of events in the circuit
 - textual order *does not* imply execution order

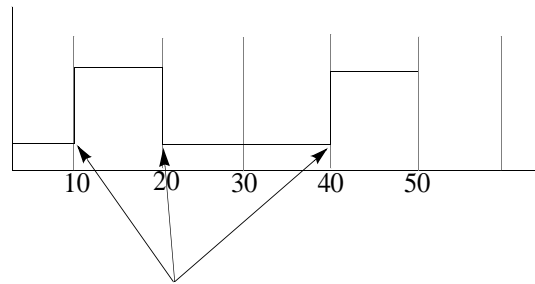
Implementation of Signals



Implementation of Signals (cont.)

- In the absence of initialization default values determined by signal type
- Waveform elements describe time-value pairs
- Transactions are internal representations of signal value assignments
 - events correspond to new signal values
 - a transaction may lead to the same signal value
- Driver is set of future signal values: current signal value provided by the transaction at the head of the list
- We can specify multiple waveform elements in a single assignment statement
- Rules for maintaining the driver
 - conflicting transactions

Example: Waveform Generation

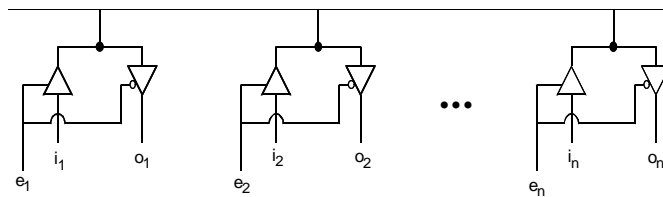


signal transitions for each waveform element

signal <= '0', '1' after 10 ns, '0' after 20 ns, '1' after 40 ns;

- Multiple waveform elements can be specified in a single signal assignment statement

Resolved Signal Types



- At any point in time what is the value of the bus signal?
- We need to “resolve” the value
 - take the values at the head of all drivers
 - pick one according to a resolution function
- Predefined IEEE 1164 type is `std_logic` and `std_logic_vector`

Conditional Signal Assignment

```
library IEEE;
use IEEE.std_logic_1164.all;
entity mux4 is
port ( In0, In1, In2, In3 : in std_logic_vector (7 downto 0);
      Sel: in std_logic_vector(1 downto 0);
      Z : out std_logic_vector (7 downto 0));
end entity mux4;

architecture behavioral of mux4 is
begin
Z <= In0 after 5 ns when Sel = "00" else
     In1 after 5 ns when Sel = "01" else
     In2 after 5 ns when Sel = "10" else
     In3 after 5 ns when Sel = "11" else
     "00000000" after 5 ns;
end architecture behavioral;
```

note type!

← Evaluation order is important!

- First true conditional expression determines the output value

Unaffected Signals

```
library IEEE;
use IEEE.std_logic_1164.all;
entity pr_encoder is
port (S0, S1, S2, S3: in std_logic;
      Z : out std_logic_vector (1 downto 0));
end entity pr_encoder;

architecture behavioral of pr_encoder is
begin
Z <= "00" after 5 ns when S0 = '1' else
     "01" after 5 ns when S1 = '1' else
     unaffected when S2 = '1' else
     "11" after 5 ns when S3 = '1' else
     "00" after 5 ns;
end architecture behavioral;
```

- Value of the signal is not changed
- VHDL 1993 only!

Selected Signal Assignment Statement

```
library IEEE;
use IEEE.std_logic_1164.all;
entity mux4 is
port (   In0, In1, In2, In3 : in std_logic_vector (7 downto 0);
        Sel: in std_logic_vector(1 downto 0);
        Z : out std_logic_vector (7 downto 0));
end entity mux4;

architecture behavioral-2 of mux4 is
begin
with Sel select
Z <= (In0 after 5 ns) when "00",
     (In1 after 5 ns) when "01",
     (In2 after 5 ns) when "10",
     (In3 after 5 ns) when "11"
     (In3 after 5 ns) when others;
end architecture behavioral;
```

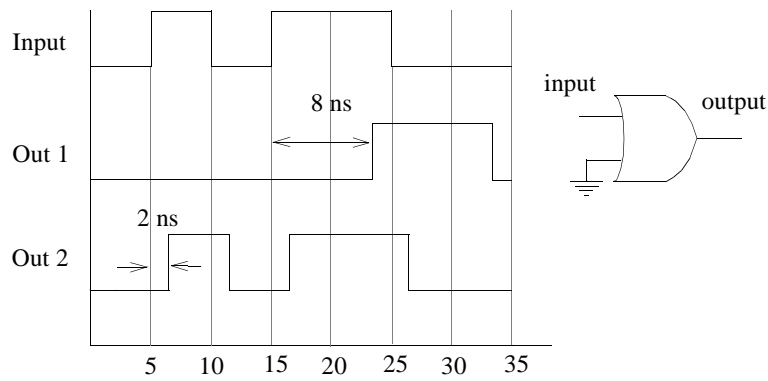
← All options must be covered and only one must be true!

- The “when others” clause can be used to ensure that all options are covered
- The “unaffected” clause may also be used here

Delay Models in VHDL

- Inertial delay
 - default delay model
 - suitable for modeling delays through devices such as gates
- Transport Delay
 - model delays through devices with very small inertia, e.g, wires
 - all input events are propagated to output signals
- Delta delay
 - what about models where no propagation delays are specified?
 - infinitesimally small delay is automatically inserted by the simulator to preserve correct ordering of events

Inertial Delays: Example

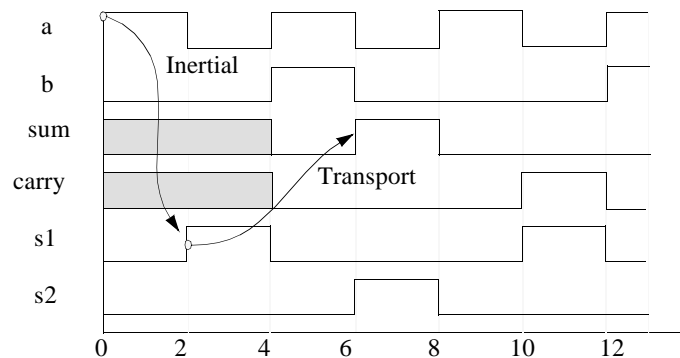
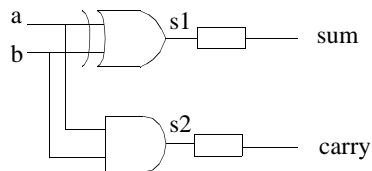


- `signal <= reject time-expression inertial value-expression after time-expression;`
- Most general form of a waveform element

Transport Delays: Example

```

architecture transport_delay of half_adder
is
signal s1, s2: std_logic:= '0';
begin
s1 <= (a xor b) after 2 ns;
s2 <= (a and b) after 2 ns;
sum <= transport s1 after 4 ns;
carry <= transport s2 after 4 ns;
end architecture transport_delay;
    
```

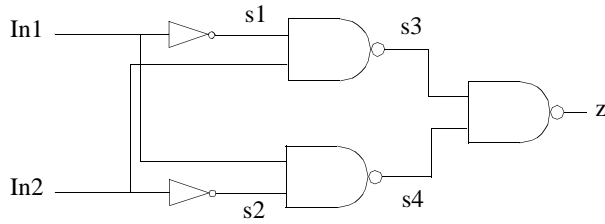


Delta Delays: Example

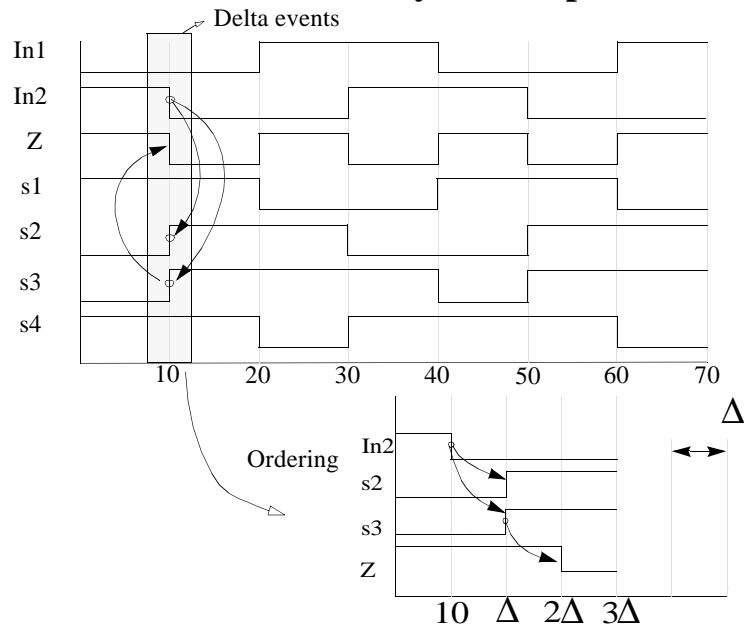
```

library IEEE;
use IEEE.std_logic_1164.all;
entity combinational is
  port (In1, In2: in std_logic;
        z : out std_logic);
end entity combinational;

architecture behavior of combinational
  signal s1, s2, s3, s4: std_logic:= '0';
  begin
    s1 <= not In1;
    s2 <= not In2;
    s3 <= not (s1 and In2);
    s4 <= not (s2 and In1);
    z <= not (s3 and s4);
  end architecture behavior;
  
```



Delta Delays: Example



Chapter Summary

- Primary unit of abstraction is a design entity
- Design units include
 - primary design units
 - entity, configuration, package declaration
 - secondary design units
 - architecture, package body
- Concurrent signal assignment statements
 - simple, selected, conditional
- Delay models
 - inertial
 - transport
 - delta