CPE 487: Digital System Design

Spring 2018

Lecture 9

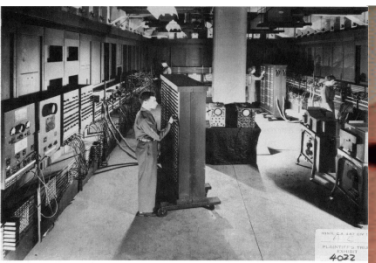
Generics & Configurations

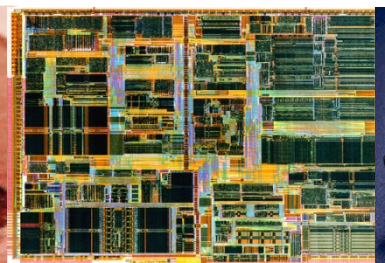
Bryan Ackland

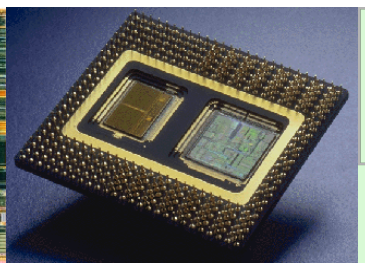
Department of Electrical and Computer Engineering

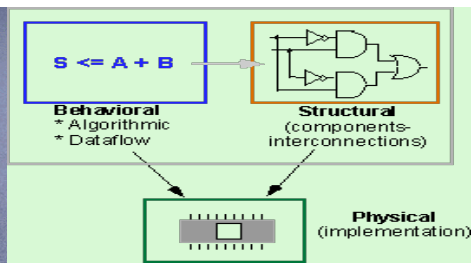
Stevens Institute of Technology

Hoboken, NJ 07030









Generics

- Entity declaration can contain generics as well as ports
- Generic value defined in entity declaration can be used as a parameter in the architecture

```
entity xor2 is
generic (gate_delay: time:= 2 ns);
port(a,b: in std_logic;
z : out std_logic);
end entity xor2;
```

```
architecture x1 of xor2 is
begin
z <= (a xor b) after gate_delay;
end architecture x1;
```

- How is this different to just defining *gate_delay* as a constant?

Parameter Passing

Generics allow parameters to be passed down through hierarchy:

generic map takes precedence over

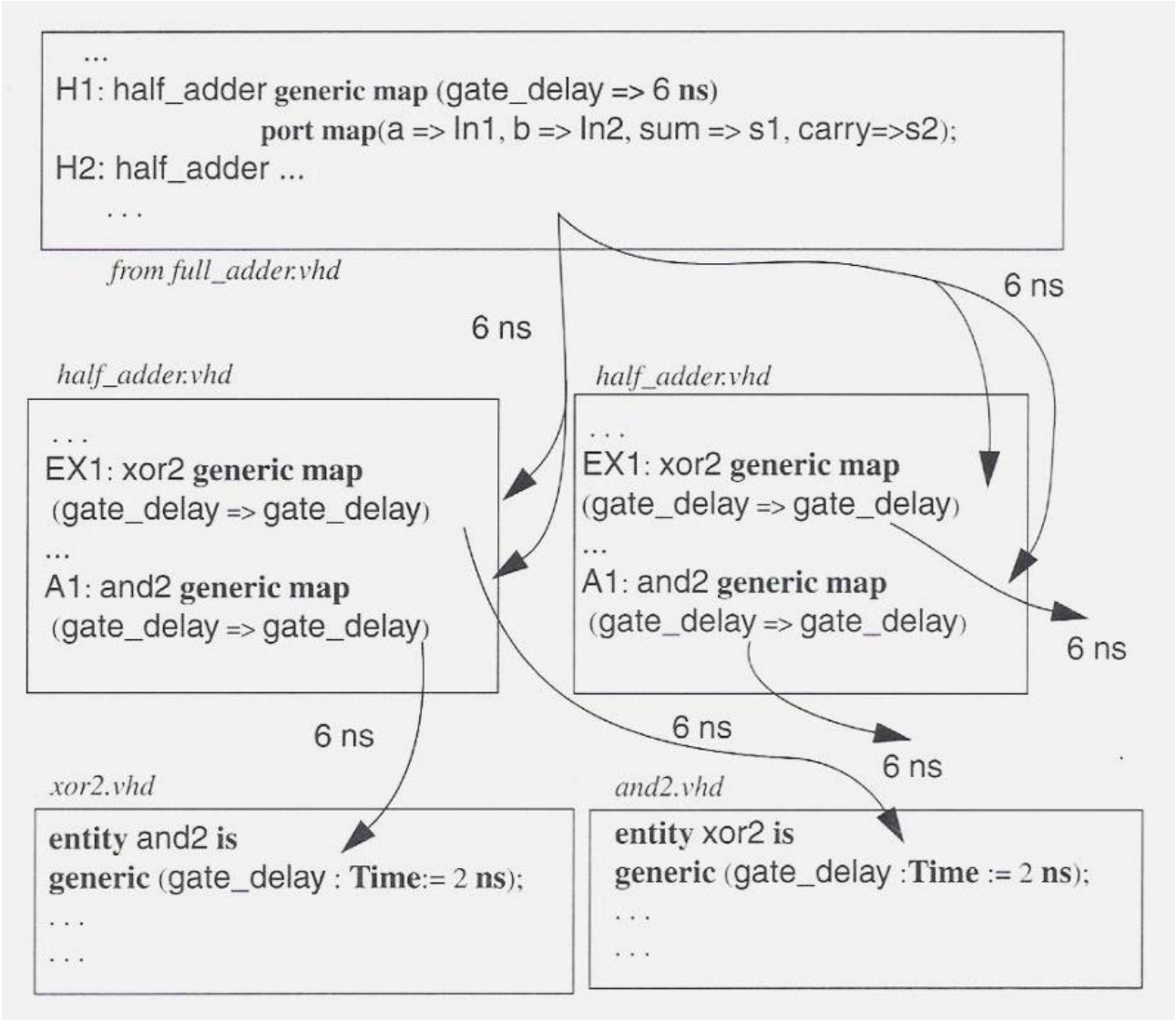
generic declaration which takes precedence over

default value in model

(Note: there is no semi-colon after generic map clause)

```
architecture gen_del of half_adder is
component xor2 is
generic (gate_delay: Time:=4ns);
port (a, b : in std_logic;
      z : out std_logic);
end component;

component and2
generic (gate_delay: Time:=2ns);
port (a, b : in std_logic;
      c : out std_logic);
end component;
begin
EX1: xor2 generic map(gate_delay=>6 ns)
      port map(a => a, b => b, z => sum);
A1: and2 generic map(gate_delay=>3 ns)
      port map(a=> a, b=> b, c=> carry);
end architecture gen_del;
```

Parameter Passing thru Hierarchy

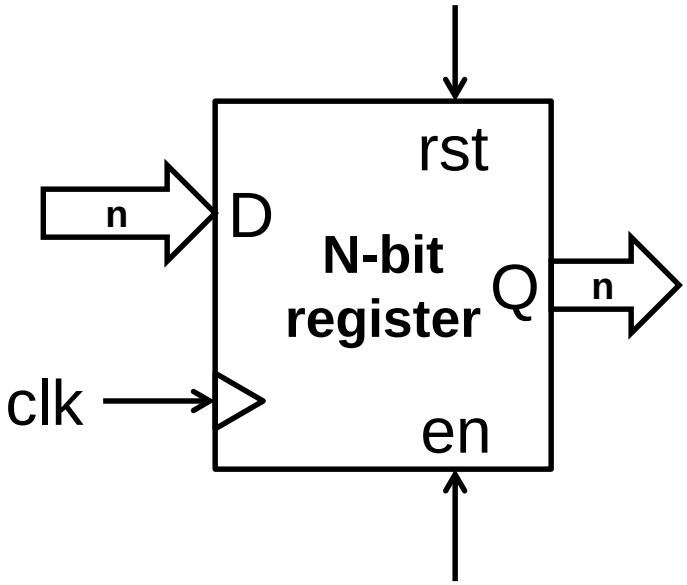


Some Rules about Generics

- Generics are like parameters that “generalize” the model
- Generics do not represent physical signals
- Generics are read-only
  - value cannot be modified by assignment statements
- Generics can be specified using expressions but value must be known at compile time
- Not limited to delays
  - can be used to specify many physical attributes of the structure

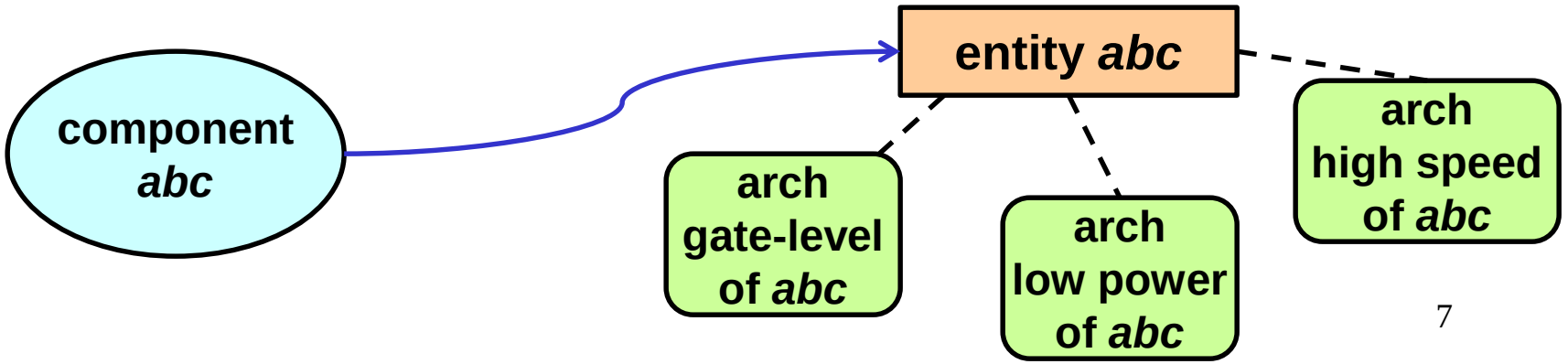
Example: N-bit register

```
entity generic_reg is
generic (N: positive:=2);
port ( clk, reset, enable: in std_logic;
     d: in std_logic_vector (N-1 downto 0);
     q: out std_logic_vector (N-1 downto 0));
end entity generic_reg;
architecture behavioral of generic_reg is
begin
reg_process: process (clk, reset)
     begin
          if reset = '1' then
               q <= (others => '0');
          elsif (rising_edge(clk)) then
               if enable = '1' then q <= d;
               end if;
          end if;
     end process reg_process;
end architecture behavioral;
```



Configurations: Using the Right Architecture

- There may be more than one architecture associated with an entity
  - represent different modeling views of the entity
    - structural vs. behavioral
    - different levels of abstraction
    - competing implementations
    - different speed grades etc.
- When we instantiate a component in a higher level mode, which architecture will be incorporated?



Binding Rules

- Binding is the process of associating an architecture with an instanced entity
- Default binding rules:
  - Search for *entity* with same name as *component*
    - current VHDL file
    - working directory
    - declared libraries
- Bind most recently compiled architecture
- This can be somewhat haphazard
  - Multiple designers working on a project?
  - How do we exercise more control?

Configurations

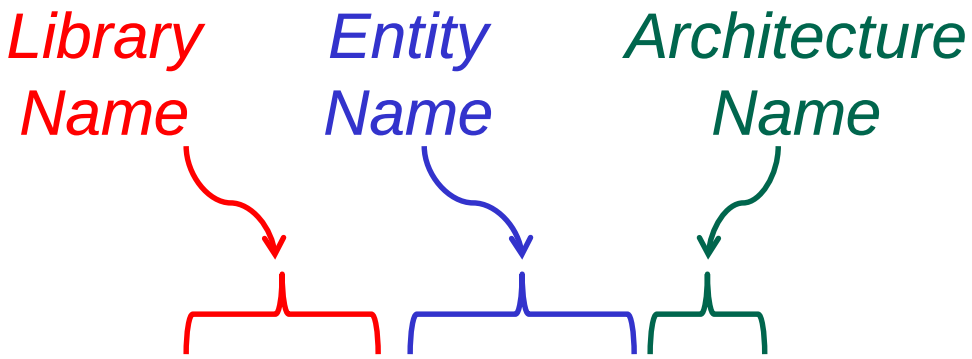
- Configurations allow designer to specify binding:

- Configuration Specification
  - Used to bind specific component instantiations to specific architectures stored in design libraries.
  - The specification appears in the declarations part of the architecture or block in which the components are instantiated
- Configuration Declaration
  - Introduced as a separate design unit – binding can be performed after the architecture body has been written.

Configuration Specification

for *list_of_component_labels:*
component_name **use entity** *entity_name* (*architecture_name);*

for example:



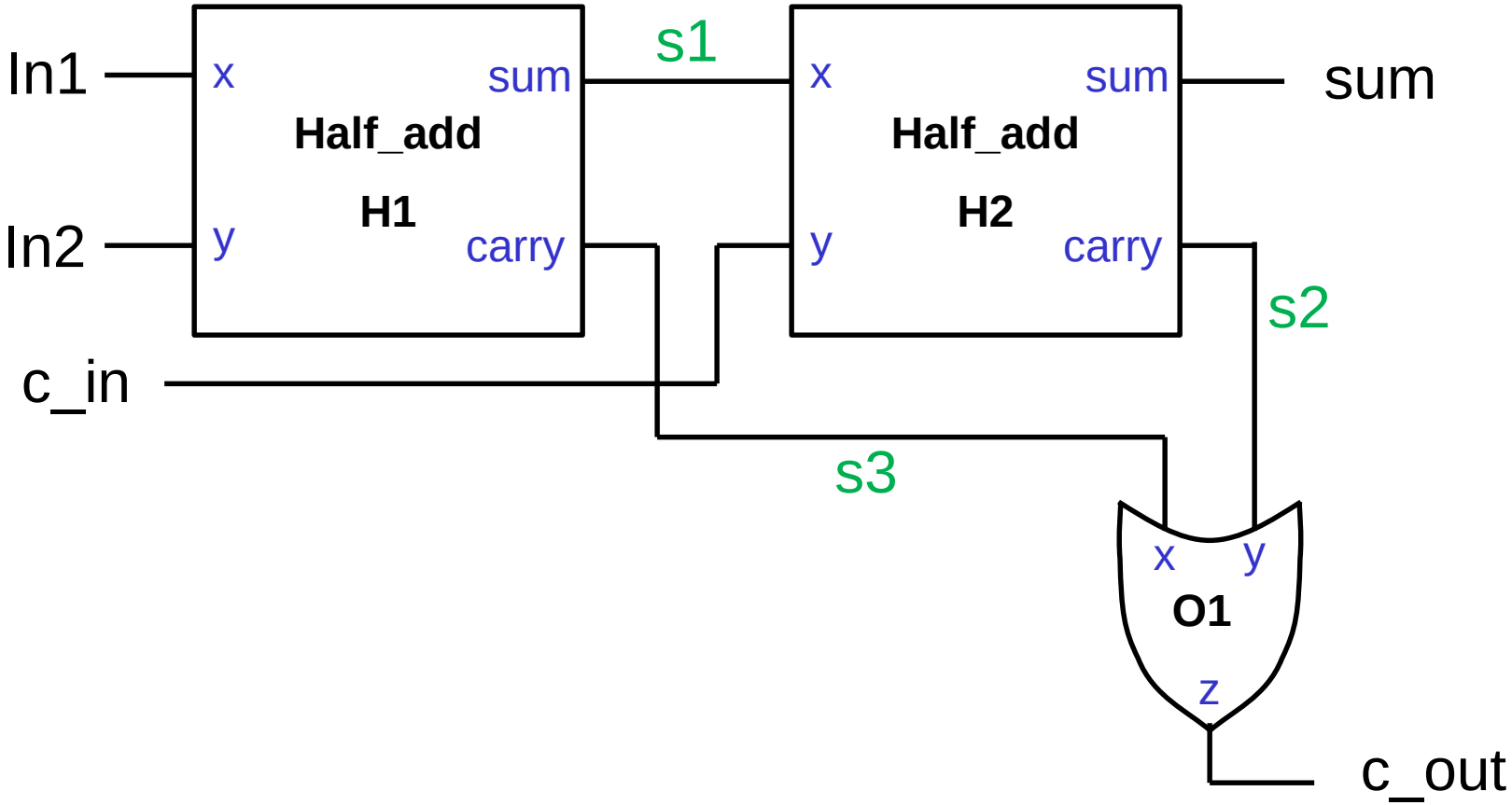
for H1: half_add **use entity** WORK.half_add (gate);

Binding to Different Entity

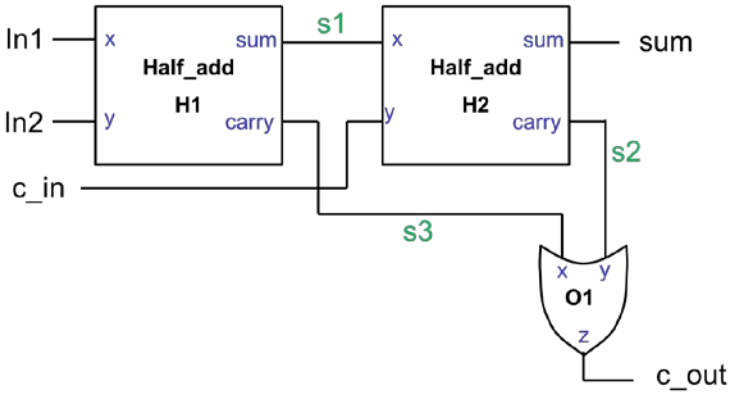
- Can also use a configuration specification to bind to a different entity
- New entity will likely have different port names and generic names
  - Need to specify generic and port maps

for *list_of_component_labels:*
component_name **use entity** *entity_name* (*architecture_name)*
[**generic map** *(generic_association list)*]
[**port map** *(port_association_list)*];

Example: Full Adder



Example: Architecture Declarations




```
architecture structural of full_adder is

component half_add is
port(x, y: in std_logic;
     sum, carry: out std_logic);
end component half_add;

component or_2 is
generic (gate_delay: Time:=2ns);
port(x, y: in std_logic;
     z: out std_logic);
end component or_2;

signal s1, s2, s3: std_logic;

for H1: half_add use entity WORK.half_add (gate);
for H2: half_add use entity WORK.half_add (RTL);
for O1: or_2 use entity POWER.Ipo2 (behavioral)
     generic map(unit_delay => gate_delay)
     port map(Ain =>x, Bin=>y, Zout=>z);
```

component declarations

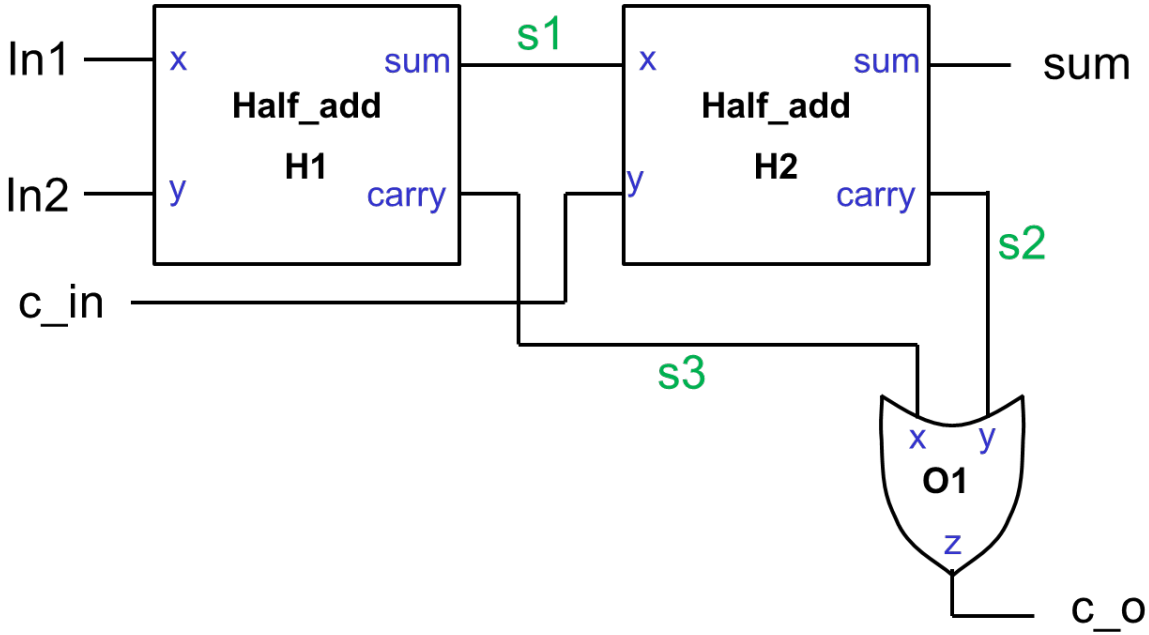
signal declarations

configuration specification

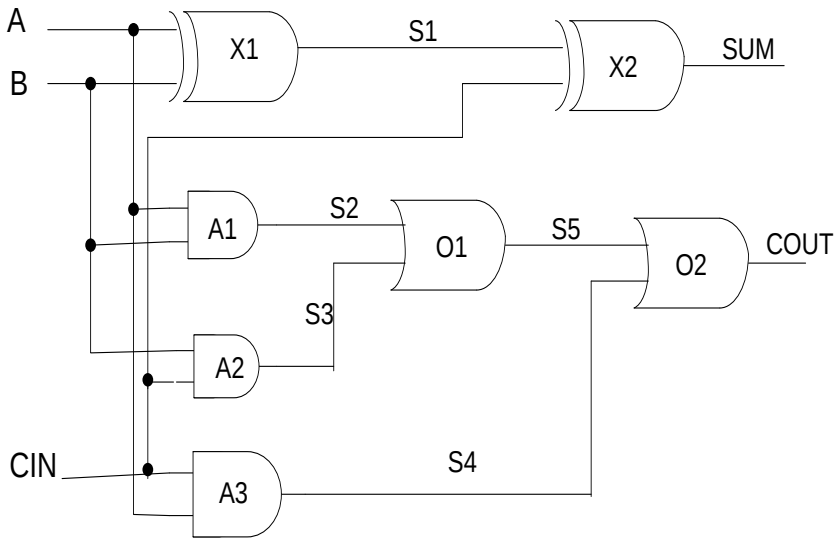
Example: Component Instantiations

begin -- *component instantiation statements*
H1: half_adder **port map** (x =>In1, y => In2, sum => s1, carry=> s3);
H2: half_adder **port map** (x => s1, y => c_in, sum => sum, carry => s2);
O1: or_2 **port map**(x => s3, y => s2, z => c_out);

end structural;



Another full adder example: Declarations

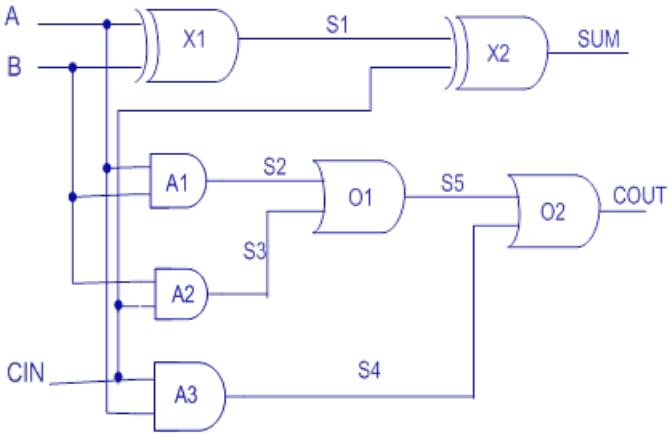



```
Library HS_LIB, CMOS_LIB;
Entity FULL_ADDER is
    port (A, B, CIN : in bit;
    SUM, COUT: out bit);
End FULL_ADDER;
```

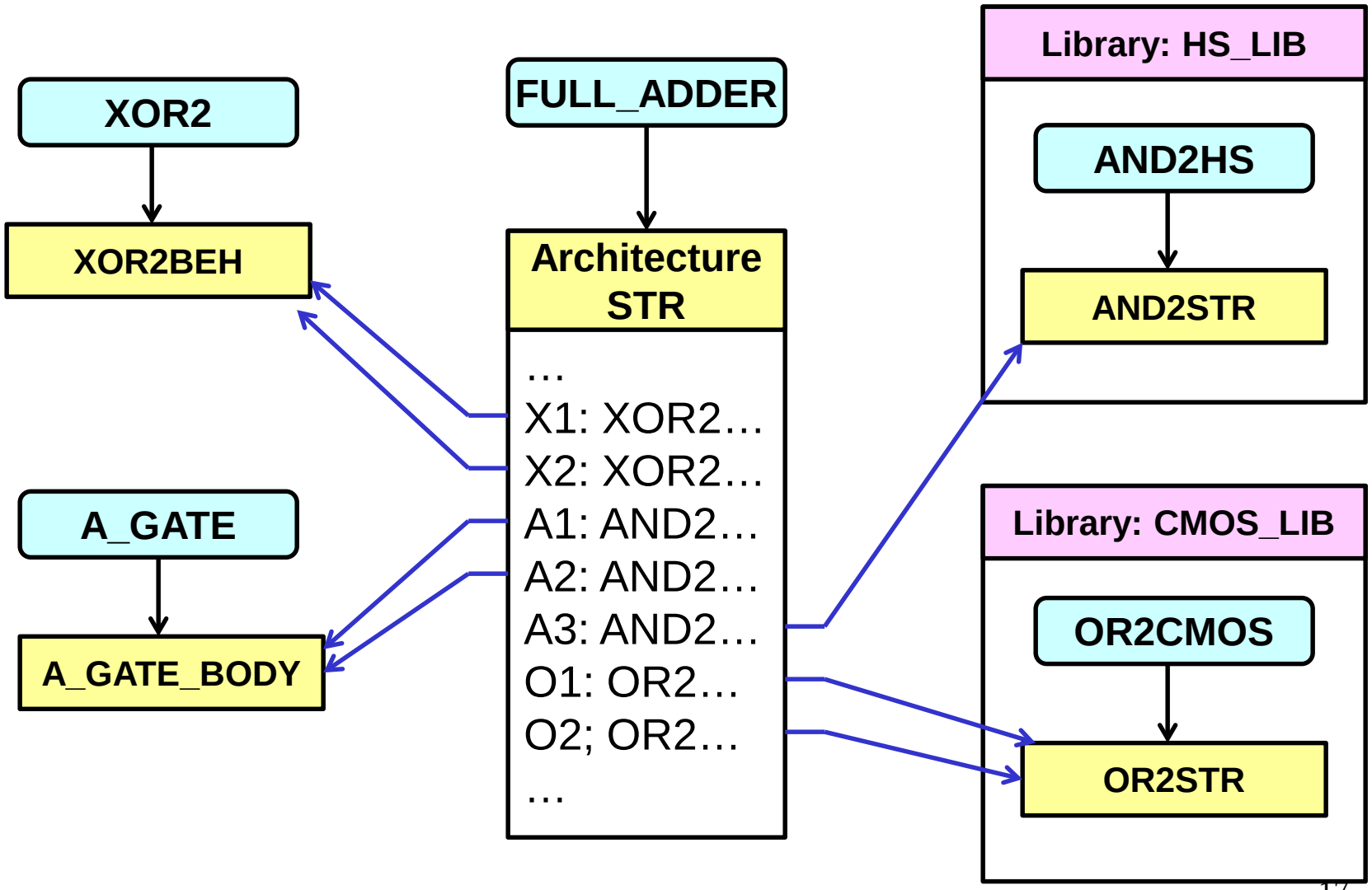
```
Architecture STR of FULL_ADDER is
    component XOR2
        port(D1, D2: in bit; DZ: out bit);
    end component;
    component AND2
        port(Z: out bit; A0, A1: in bit);
    end component;
    component OR2
        port(N1, N2: in bit; Z: out bit);
    end component;

Signal S1, S2, S3, S4, S5: bit;
```

Full adder: Specifications & Architecture Body

```
    for X1, X2 : XOR2      --binding to a specific architecture
        use entity WORK.XOR2(XOR2BEH);

    for A3: AND2           --binding to a different entity
        use entity HS_LIB.AND2HS(AND2STR)
          port map (HS_B => A1, HS_Z=>Z, HS_A=> A0);

    for all: OR2           --binding all instantiations of OR2 component
        use entity CMOS_LIB.OR2CMOS(OR2STR);

    for others: AND2       --binding all unbound instantiations of AND2
        use entity WORK.A_GATE(A_GATE_BODY) port map(A0, A1, Z);
begin
     X1: XOR2 port map (A, B, S1);
     X2 : XOR2 port map (S1, CIN, SUM);
     A1 : AND2 port map (S2, A, B);
     A2 : AND2 port map (S3, B, CIN);
     A3 : AND2 port map (S4, A, CIN);
     O1: OR2 port map (S2, S3, S5);
     O2: OR2 port map (S4, S5, COUT);
end STR;
```



Instances, Entities & Architectures



Configuration Declaration

- Configuration Specification requires editing of the architecture body
- Configuration Declaration takes the configuration information and places it in a separate design unit
  - which can be in a separate file
- Allows for late binding of components
  - after the architecture has been written

configuration *configuration-name* **of** *entity-name* **is**

…

…

end [**configuration**] [*configuration-name*];

Full adder(1): Configuration Declaration

```
configuration Config_A of full_adder is   -- name of entity being configured
    for structural                        -- name of architecture being configured

        for H1: half_add
            use entity WORK.half_add(gate);
        end for;

        for H2: half_add
            use entity WORK.half_add(RTL);
        end for;

        for O1: or_2
            use entity POWER.Ipo2(behavioral)
                generic map(unit_delay => gate_delay)
                port map(A=>x, B=>y, Zout=>z);
        end for;

    end for;
end configuration Config_A;
```

Full adder (2): Configuration Declaration

```
library HS_LIB, CMOS_LIB;
configuration FA_CON of FULL_ADDER is   -- name of entity being configured
    for STR                             -- name of architecture being configured
        for X1, X2 : XOR2
            use entity WORK.XOR2(XOR2BEH);
        end for;
        for A3: AND2
            use entity HS_LIB.AND2HS(AND2STR)
                port map (HS_B => A1, HS_Z=>Z, HS_A=> A0);
        end for;
        for all: OR2
            use entity CMOS_LIB.OR2CMOS(OR2STR);
        end for;
        for others: AND2
            use entity WORK.A_GATE(A_GATE_BODY)
                port map(A0, A1, Z);
        end for;
    end for;
end configuration FA_CON;
```