EE434
ASIC & Digital Systems

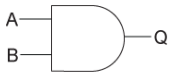
VHDL

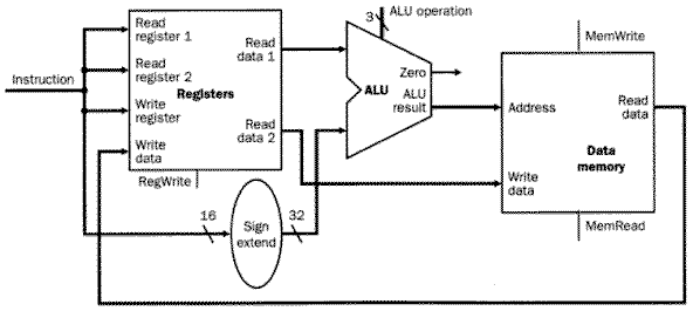
Spring 2016

Dae Hyun Kim
daehyun@eecs.wsu.edu

Terminologies

- Entity
  - Basic building block





- Architecture (belongs to an entity)
  - Description of an entity
    - Behavioral
    - Structural

```
ARCHITECTURE  mux2arch  OF  mux2  IS
  SIGNAL  sel  :  INTEGER;
BEGIN
  sel  <= 0  WHEN  s = '0'
               ELSE  1;
  z  <=   a  AFTER  0.2 NS  WHEN  sel = 0
     ELSE b  AFTER  0.2 NS;
END  mux2arch;
```

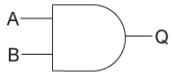
```
ARCHITECTURE  mux2arch  OF  mux2  IS
BEGIN
  u1 : myAND2 PORT MAP (...);
  u2 : myAND2 PORT MAP (...);
  u3 : myINV PORT MAP (...);
  ...
END  mux2arch;
```

Terminologies

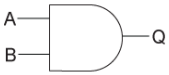
- Package
  - Libraries
- Driver
  - A source on a signal
- Bus
  - A group of signals
- Attribute
  - Properties of an object
- Generic
  - Parameter
- Process
  - A basic unit of execution

Terminologies

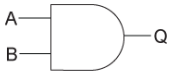
- Instance



Definition



Instance (name: u1)



Instance (name: u2)

- Instantiation
  - Create an instance

VHDL Coding

- For each functional module
  - Create an entity (myModule).
  - Create an architecture (myModule_arch).

- Create a testbench entity to test it.
  - Create an entity (myModule_tb).
  - Create an architecture (myModule_tb_arch).
  - Define test input and output signals.
  - Instantiate the functional module.
  - Connect the test input and output signals to the ports (pins) of the target module.
  - Set the input signals to a specific vector and observe the output values.

Entity

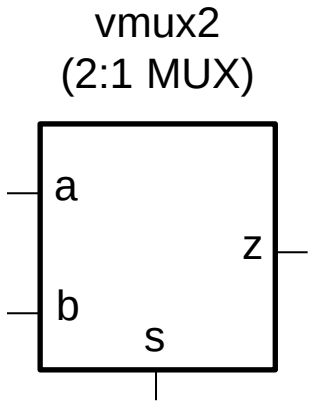
- Format

```
ENTITY  entity_name  IS
   PORT  (  port_list  );
END  entity_name;
```

- Example

```
ENTITY  vmux2  IS
   PORT  (  a, b : IN  BIT;
            s : IN BIT;
            z : OUT  BIT  );
END  vmux2;
```

keyword



Example

- Create an entity for a four-input NAND gate.
- Create an entity for a full adder.
- Create an entity for a D-F/F.

Example

- Create an entity for a four-input NAND gate.

```
ENTITY vnand4 IS
   PORT ( a, b, c, d : IN BIT;
          zn : OUT BIT );
END vnand4;
```

- Create an entity for a full adder.

```
ENTITY vfa IS
   PORT ( a, b, ci : IN BIT;
          s, co : OUT BIT );
END vfa;
```

- Create an entity for a D-F/F.

```
ENTITY vdff IS
   PORT ( D, CK : IN BIT;
          Q, QN : OUT BIT );
END vdff;
```

Architecture

- Details of an entity
- Format

```
ARCHITECTURE architecture_name OF entity_name IS
BEGIN
END architecture_name;
```

- Example (behavioral description)

```
ARCHITECTURE vnand4_arch OF vnand4 IS
BEGIN
   zn <= NOT (a AND b AND c AND d);
END vnand4_arch;
```

"<=" is a signal assignment.

Example

- Create an architecture for a three-input OR gate.
  - Entity name: vor2
  - Inputs: a, b, c
  - Output: z
  - Use “OR”.

Example

- Create an architecture for a three-input OR gate.
  - Entity name: vor2
  - Inputs: a, b, c
  - Output: z
  - Use “OR”.

```
ARCHITECTURE vor2_arch OF vor2 IS
BEGIN
   z <= a OR b OR c;
END vor2_arch;
```

Example

- Create an architecture for a two-input MUX.
  - Entity name: vmux2
  - Inputs: a, b, s (selection)
  - Output: z
  - Function
    - z = a when s is 0.
    - z = b when s is 1.

Example

- Create an architecture for a two-input MUX.
  - Entity name: vmux2
  - Inputs: a, b, s (selection)
  - Output: z
  - Function
    - z = a when s is 0.
    - z = b when s is 1.

```
ARCHITECTURE vmux2_arch OF vmux2 IS
BEGIN
   z <= (a AND (NOT s)) OR (b AND s);
END vmux2_arch;
```

Example

- Create an architecture for a full adder.
  - Entity name: vfa
  - Inputs: a, b, ci (carry-in)
  - Outputs: s (sum), co (carry-out)
  - Function
    - s = a^b^ci (^ is XOR)
    - co = a*b + b*ci + a*ci
  - Use “XOR” for XOR.

Example

- Create an architecture for a full adder.
  - Entity name: vfa
  - Inputs: a, b, ci (carry-in)
  - Outputs: s (sum), co (carry-out)
  - Function
    - s = a^b^ci (^ is XOR)
    - co = a*b + b*ci + a*ci
  - Use “XOR” for XOR.

```
ARCHITECTURE vfa_arch OF vfa IS
BEGIN
   s <= a XOR b XOR ci;
   co <= (a AND b) OR (b AND ci) OR (a AND ci);
END vfa_arch;
```

Architecture

- How does it work?

```
ARCHITECTURE vnand4_arch OF vnand4 IS
BEGIN
   zn <= NOT (a AND b AND c AND d);
END vnand4_arch;
```

- Execution of a statement is triggered by events.

- An event on a signal
  - A change in the value of the signal
  - The signal is *sensitive* to the change.

Architecture

- How does it work?

```
ARCHITECTURE vnand4_arch OF vnand4 IS
BEGIN
   zn <= NOT (a AND b AND c AND d);
END vnand4_arch;
```

- If any of the input signals (a, b, c, d) changes, the above statement will be executed.

Architecture

- How does it work?

```
ARCHITECTURE vfa_arch OF vfa IS
BEGIN
   s <= a XOR b XOR ci;
   co <= (a AND b) OR (b AND ci) OR (a AND ci);
END vfa_arch;
```

- The order of the statements is not important.

- All the statements in an architecture are executed in parallel.

Signal

- Signal
  - Local net
- Format

```
ARCHITECTURE architecture_name OF entity_name IS
   SIGNAL signal_name : signal_type;
BEGIN
END architecture_name;
```

- Example

```
ARCHITECTURE vnand4_arch OF vnand4 IS
   SIGNAL n1, n2 : BIT;
BEGIN
   n1 <= a AND b;
   n2 <= c AND d;
   zn <= NOT (n1 AND n2);
END vnand4_arch;
```

Architecture

- Example (structural description) – Netlist

```
ARCHITECTURE vnand4_arch OF vnand4 IS
   COMPONENT vnand2
      PORT ( a, b : IN BIT; zn : OUT BIT );
   END COMPONENT;
   COMPONENT vor2
      PORT ( a, b : IN BIT; z : OUT BIT );
   END COMPONENT;
   SIGNAL n1, n2 : BIT;
BEGIN
   u1 : vnand2 PORT MAP (a, b, n1);
   u2 : vnand2 PORT MAP (c, d, n2);
   u3 : vor2 PORT MAP (a => n1, b => n2, z => zn);
END vnand4_arch;
```

Testing

- Entity

```
ENTITY  FA_X1  IS
   PORT  (  a, b, ci : IN BIT;   s, co : OUT BIT  );
END  FA_X1;
```

- Architecture

```
ARCHITECTURE  FA_X1_arch  OF  FA_X1  IS
BEGIN
   s <= a XOR b XOR ci;
   co <= (a AND b) OR (b AND ci) OR (a AND ci);
END  FA_X1_arch;
```

Testing

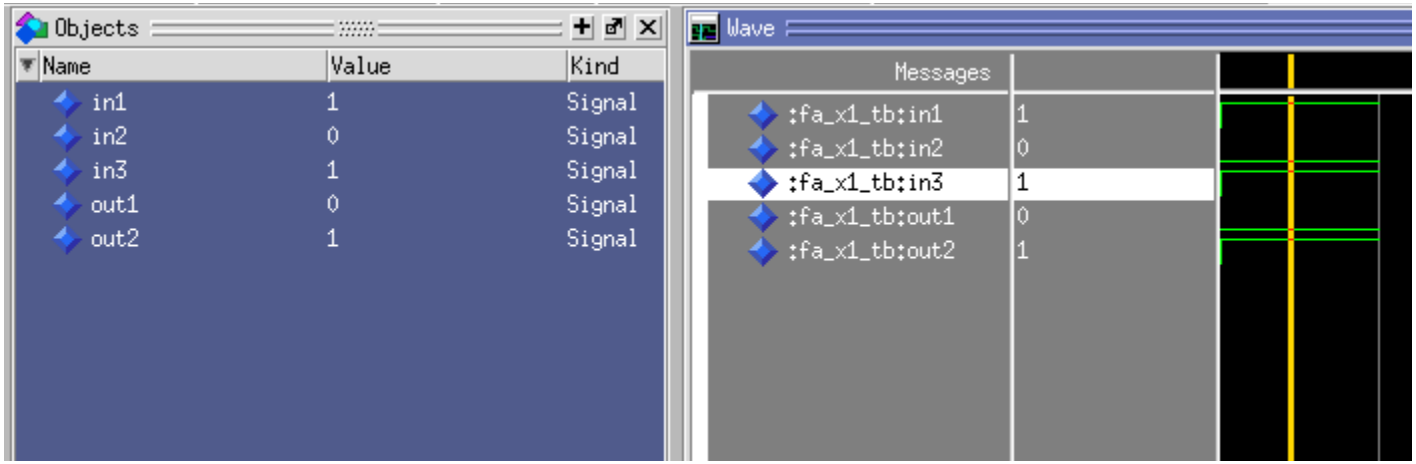
- Test entity

```
ENTITY FA_X1_tb IS
END FA_X1_tb;
```

- Test architecture

```
ARCHITECTURE FA_X1_tb_arch OF FA_X1_tb IS
   COMPONENT FA_X1 PORT ( a, b, ci : IN BIT; s, co : OUT BIT );
   END COMPONENT;
   SIGNAL in1, in2, in3, out1, out2 : BIT;
BEGIN
   u1 : FA_X1 PORT MAP ( a => in1, b => in2, ci => in3, s => out1, co => out2 );
   in1 <= '1';
   in2 <= '0';
   in3 <= '1';
END FA_X1_tb_arch;
```

Testing


Objects
Name
Value
Kind
in1
1
Signal
in2
0
Signal
in3
1
Signal
out1
0
Signal
out2
1
Signal
Wave
Messages
:fa_x1_tb:in1
1
:fa_x1_tb:in2
0
:fa_x1_tb:in3
1
:fa_x1_tb:out1
0
:fa_x1_tb:out2
1

Architecture

- Sequential

```
ARCHITECTURE vnand4_arch OF vnand4 IS
BEGIN
   PROCESS (a, b, c, d)
      VARIABLE v1 : INTEGER;
   BEGIN
      IF (a = '0') THEN v1 := 1;
      ELSIF (b = '0') THEN v1 := 1;
      ELSIF (c = '0') THEN v1 := 1;
      ELSIF (d = '0') THEN v1 := 1;
      ELSE v1 := 0;
      END IF;
      CASE v1 IS
         WHEN 1 =>
            zn <= '1';
         WHEN OTHERS =>
            zn <= '0';
      END CASE;
   END PROCESS;
END vnand4_arch;
```