GENERATE

VHDL provides the GENERATE statement to create well-patterned structures easily.

Any VHDL concurrent statement can be included in a GENERATE statement, including another GENERATE statement.

Two ways to apply

- **FOR scheme**
- **IF scheme**

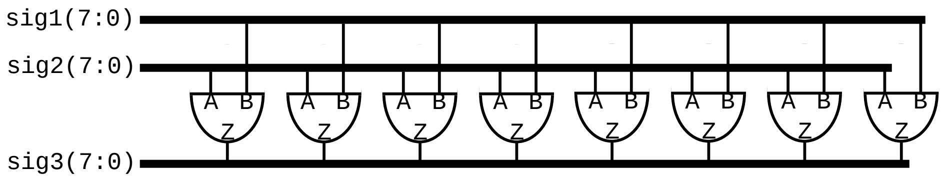
FOR Scheme Format:

```
label : FOR identifier IN range GENERATE
  concurrent_statements;
END GENERATE [label];
```

Generate Statement - FOR scheme

```
ARCHITECTURE test OF test IS
COMPONENT and02
  PORT( a0 : IN  std_logic;
        a1 : IN  std_logic;
        y  : OUT std_logic);
 END COMPONENT and02;

BEGIN
 G1 : FOR n IN (length-1) DOWNTO 0 GENERATE
   and_gate:and02
   PORT MAP( a0 => sig1(n),
             a1 => sig2(n),
             y  => z(n));
  END GENERATE G1;
END test;
```



With the FOR scheme

- **All objects created are similar.**
- **The GENERATE parameter must be discrete and is undefined outside the GENERATE statement.**
- **Loop cannot be terminated early**

Note: This structure could have been created by:

```
 sig3 <= sig1 AND sig2;
```

provided the AND operator was overloaded for vector operations.

Generate Statement - IF scheme

Allows for conditional creation of components.

Can’t use ELSE or ELSIF clauses.

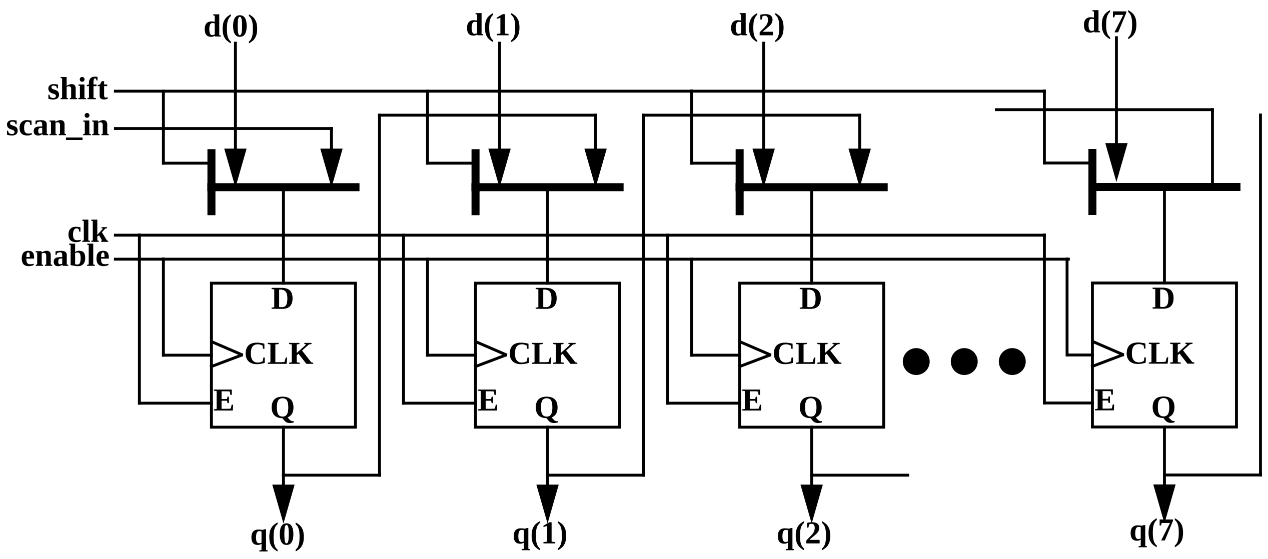
IF Scheme Format:

```
label : IF (boolean_expression) GENERATE
  concurrent_statements;
END GENERATE [label];
```

The next slide will show how we can use both FOR and IF schemes.

Use of GENERATE - An example

Suppose we want to build an 8-bit shift register.



Suppose furthermore that we had previously defined the following components:

```
ENTITY dff IS
  PORT(d, clk, en   : IN   std_logic;
       q, qn        : OUT  std_logic);
  END ENTITY dff;

ENTITY mux21 IS
  PORT(a, b, sel    : IN   std_logic;
       z            : OUT  std_logic);
  END ENTITY mux21;
```

GENERATE

From the block diagram we know what the entity should look like.

```
ENTITY sr8 IS
 PORT(
  din     : IN std_logic_vector(7 DOWNTO 0);
  sel     : IN std_logic;
  shift   : IN std_logic;
  scan_in : IN std_logic;
  clk     : IN sed_logic;
  enable  : IN std_logic;
  dout   : OUT std_logic_vector(7 DOWNTO 0));
```

Within the architecture statement we have to declare the components within the declaration region before using them. This is done as follows:

```
ARCHITECTURE example OF sr8 IS
--declare components in declaration area
COMPONENT dff IS
  PORT(d, clk, en  : IN   std_logic;
       q, qn       : OUT  std_logic);
  END COMPONENT;
COMPONENT mux21 IS
  PORT(a, b, sel : IN   std_logic;
       z         : OUT  std_logic);
  END COMPONENT;
```

Component declarations look just like entity clauses, except COMPONENT replaces ENTITY. Use cut and paste to prevent mistakes!

Generate

After the component declarations, we declare the internal signal.

```
SIGNAL mux_out : std_logic_vector(7 DOWNTO 0);
```

With loop and generate statements, instantiate muxes and dff’s.

```
BEGIN
  OUTERLOOP: FOR i IN 0 TO 7 GENERATE
      INNERLOOP1: IF (i = 0) GENERATE
        MUX: mux21 PORT MAP(a  => d(i),
                            b  => scan_in,
                            z  => mux_out(i));
        FLOP: dff PORT MAP(d   => mux_out(i),
                           clk => clk,
                           en  => enable,
                           q   => dout(i)); --qn not listed
      END GENERATE INNERLOOP1;
      INNERLOOP2: IF (i > 0) GENERATE
        MUX: mux21 PORT MAP(a  => d(i),
                            b  => dout(i-1),
                            z  => mux_out(i));
        FLOP: dff PORT MAP(d   => mux_out(i),
                           clk => clk,
                           en  => enable,
                           q   => dout(i),
                           qn  => OPEN); --qn listed as OPEN
       END GENERATE INNERLOOP2;
  END GENERATE OUTERLOOP;
END example;
```