Digital Circuit Model Design

Eri Prasetyo Wibowo
Gunadarma University

Model

- A model of a digital circuit is an abstract expression in some modeling language that captures those aspects that we are interested in for certain design tasks and omits other aspects
- For example, one form of model may express just the function that the circuit is to perform, without including aspects of timing, power consumption or physical construction

Cont’

- Another form of model ay express the logical structure of the circuit, that is, the way in which it is composed of interconnected components such as gates and flip-flops.
- Both of these forms of model may be conveniently expressed in a hardware description language (HDL), which is a form of computer language akin to programming languages, specialized for this purpose.

VHDL

- VHDL was one of the products of that program.
- Since then, the specification of VHDL has been standardized in the United States by the Institute of Electrical and Electronic Engineers (IEEE) and internationally by the International Electrotechnical Commission (IEC), and the language has been widely adopted by designers and tool developers.

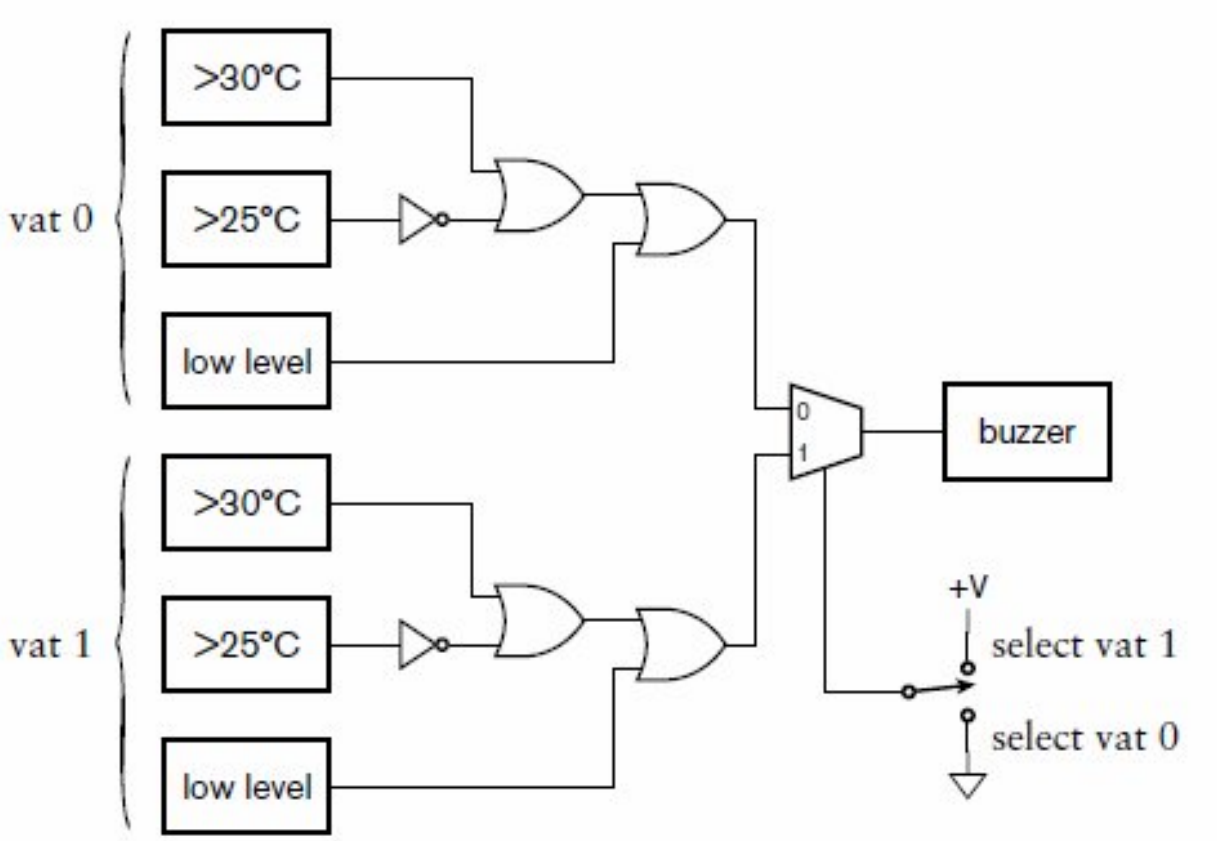
Cont’

- VHDL is not the only HDL used for digital system design. Others include Verilog and its more recent successor, SystemVerilog. Also, SystemC, an extension of the C++ programming language, is gaining increased usage.

Cont’

The VHDL model has two parts, one describing the inputs and output of the circuit, and the other describing the interconnection of gate components that realizes the circuit.

Example



Bagaimana Design di VHDL ?

Solution

The first of these takes the form of an entity declaration:

```
library ieee; use ieee.std_logic_1164.all;

entity vat_buzzer is
  port ( buzzer                                  : out std_logic;
         above_25_0, above_30_0, low_level_0 : in  std_logic;
         above_25_1, above_30_1, low_level_1 : in  std_logic;
         select_vat_1                          : in std_logic );
end entity vat_buzzer;
```

The circuit is represented in VHDL as an entity, in this case named vat_buzzer. The first line identifies a standard library, ieee, and a package, std_logic_1164, containing definitions that we want to reference in our model

```
library ieee; use ieee.std_logic_1164.all;

entity vat_buzzer is
  port ( buzzer                                  : out std_logic;
         above_25_0, above_30_0, low_level_0 : in  std_logic;
         above_25_1, above_30_1, low_level_1 : in  std_logic;
         select_vat_1                          : in std_logic );
end entity vat_buzzer;
```

The entity has *ports*, described in the port list of the entity declaration. Each port is given a name and is either an output (out) or an input (in). The port description also specifies the *type* of each port, that is, the set of values that the port can take on. In this example, each port is of type std_logic, defined by the std_logic_1164 package

```
library ieee; use ieee.std_logic_1164.all;

entity vat_buzzer is
  port ( buzzer                                  : out std_logic;
         above_25_0, above_30_0, low_level_0 : in  std_logic;
         above_25_1, above_30_1, low_level_1 : in  std_logic;
         select_vat_1                          : in std_logic );
end entity vat_buzzer;
```

- The std_logic type includes the values '0' and '1', corresponding to the binary values 0 and 1 introduced earlier.
- The difference is that VHDL requires the quotation marks to distinguish the std_logic values from the numbers 0 and 1.

The second part of the VHDL model takes the form of an *architecture*:

```
library dld; use dld.gates.all;

architecture struct of vat_buzzer is

  signal below_25_0, temp_bad_0, wake_up_0 : std_logic;
  signal below_25_1, temp_bad_1, wake_up_1 : std_logic;

begin

  -- components for vat 0
  inv_0 : inv port map (above_25_0, below_25_0);
  or_0a : or2 port map (above_30_0, below_25_0, temp_bad_0);
  or_0b : or2 port map (temp_bad_0, low_level_0, wake_up_0);

  -- components for vat 1
  inv_1 : inv port map (above_25_1, below_25_1);
  or_1a : or2 port map (above_30_1, below_25_1, temp_bad_1);
  or_1b : or2 port map (temp_bad_1, low_level_1, wake_up_1);

  select_mux : mux2 port map (wake_up_0, wake_up_1,
                              select_vat_1, buzzer);

end architecture struct;
```

Cont’

```
library dld; use dld.gates.all;

architecture struct of vat_buzzer is

  signal below_25_0, temp_bad_0, wake_up_0 : std_logic;
  signal below_25_1, temp_bad_1, wake_up_1 : std_logic;

begin

  -- components for vat 0
  inv_0 : inv port map (above_25_0, below_25_0);
  or_0a : or2 port map (above_30_0, below_25_0, temp_bad_0);
  or_0b : or2 port map (temp_bad_0, low_level_0, wake_up_0);

  -- components for vat 1
  inv_1 : inv port map (above_25_1, below_25_1);
  or_1a : or2 port map (above_30_1, below_25_1, temp_bad_1);
  or_1b : or2 port map (temp_bad_1, low_level_1, wake_up_1);

  select_mux : mux2 port map (wake_up_0, wake_up_1,
                              select_vat_1, buzzer);

end architecture struct;
```

The first line of the architecture identifies a design library, dld, and a package, gates, containing definitions of components that we want to include in our model

```
library dld; use dld.gates.all;

architecture struct of vat_buzzer is

  signal below_25_0, temp_bad_0, wake_up_0 : std_logic;
  signal below_25_1, temp_bad_1, wake_up_1 : std_logic;

begin

  -- components for vat 0
  inv_0 : inv port map (above_25_0, below_25_0);
  or_0a : or2 port map (above_30_0, below_25_0, temp_bad_0);
  or_0b : or2 port map (temp_bad_0, low_level_0, wake_up_0);

  -- components for vat 1
  inv_1 : inv port map (above_25_1, below_25_1);
  or_1a : or2 port map (above_30_1, below_25_1, temp_bad_1);
  or_1b : or2 port map (temp_bad_1, low_level_1, wake_up_1);

  select_mux : mux2 port map (wake_up_0, wake_up_1,
                              select_vat_1, buzzer);

end architecture struct;
```

The architecture also references the definitions in the std_logic_1164 package, but since we identified that package and its library in the entity declaration, we don't need to duplicate the library and use clauses in the architecture.

```
library dld; use dld.gates.all;

architecture struct of vat_buzzer is

  signal below_25_0, temp_bad_0, wake_up_0 : std_logic;
  signal below_25_1, temp_bad_1, wake_up_1 : std_logic;

begin

  -- components for vat 0
  inv_0 : inv port map (above_25_0, below_25_0);
  or_0a : or2 port map (above_30_0, below_25_0, temp_bad_0);
  or_0b : or2 port map (temp_bad_0, low_level_0, wake_up_0);

  -- components for vat 1
  inv_1 : inv port map (above_25_1, below_25_1);
  or_1a : or2 port map (above_30_1, below_25_1, temp_bad_1);
  or_1b : or2 port map (temp_bad_1, low_level_1, wake_up_1);

  select_mux : mux2 port map (wake_up_0, wake_up_1,
                              select_vat_1, buzzer);

end architecture struct;
```

- The architecture is named struct and is associated with the vat_buzzer entity.
- Within the architecture, a number of signals are declared for connecting the components together. Each signal is given a name and has a specified type, in this case, std_logic.

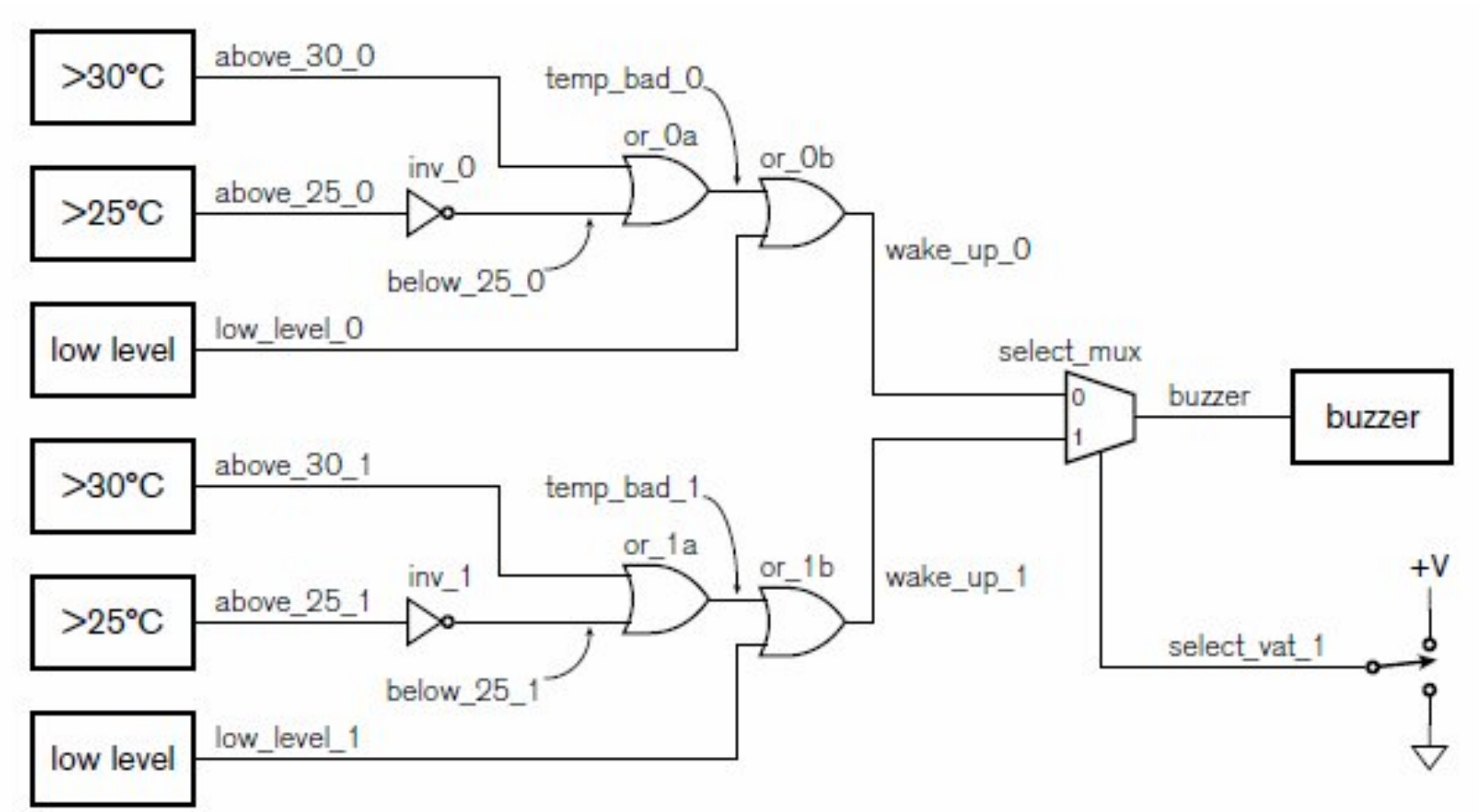
```
library dld; use dld.gates.all;

architecture struct of vat_buzzer is

  signal below_25_0, temp_bad_0, wake_up_0 : std_logic;
  signal below_25_1, temp_bad_1, wake_up_1 : std_logic;

begin

  -- components for vat 0
  inv_0 : inv port map (above_25_0, below_25_0);
  or_0a : or2 port map (above_30_0, below_25_0, temp_bad_0);
  or_0b : or2 port map (temp_bad_0, low_level_0, wake_up_0);

  -- components for vat 1
  inv_1 : inv port map (above_25_1, below_25_1);
  or_1a : or2 port map (above_30_1, below_25_1, temp_bad_1);
  or_1b : or2 port map (temp_bad_1, low_level_1, wake_up_1);

  select_mux : mux2 port map (wake_up_0, wake_up_1,
                              select_vat_1, buzzer);

end architecture struct;
```

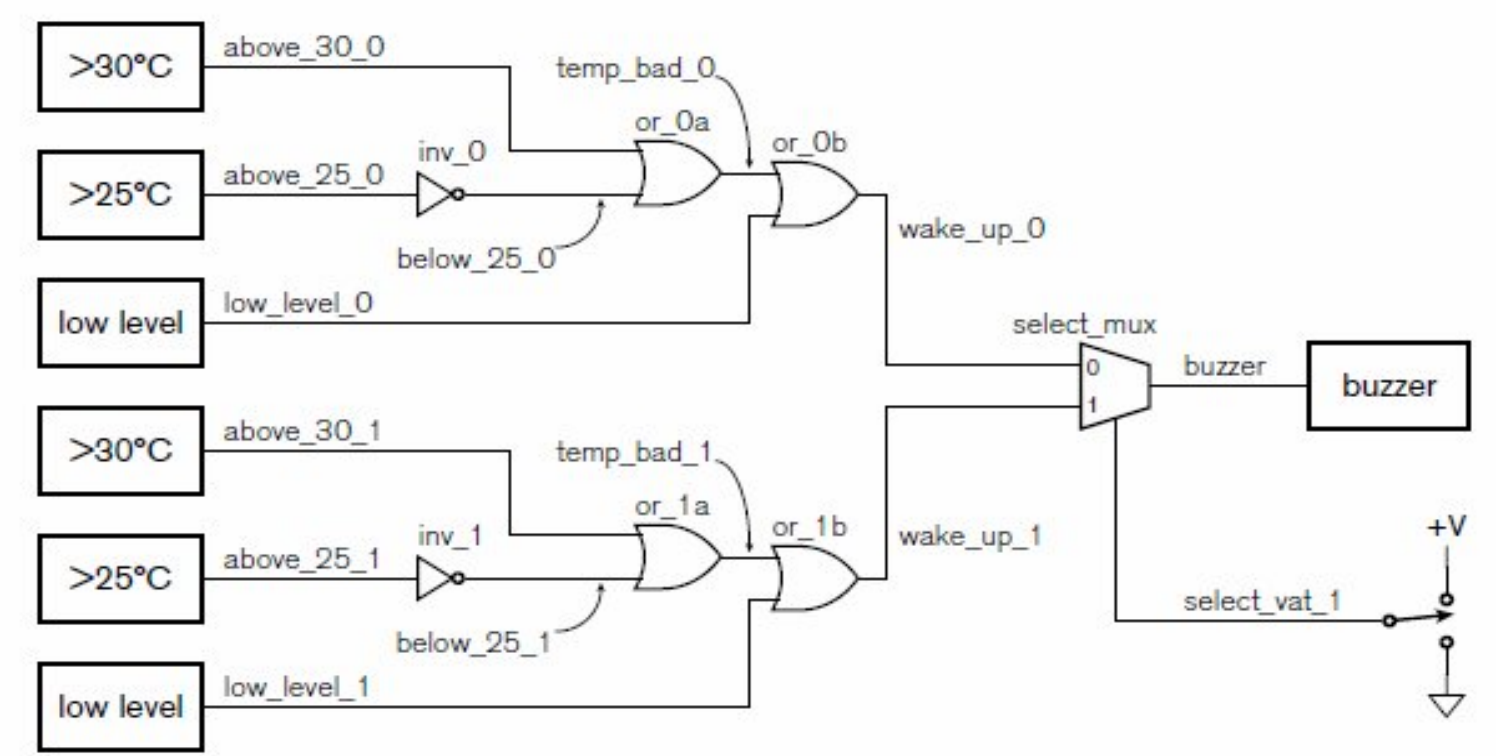
- The architecture is named struct and is associated with the vat_buzzer entity.
- Within the architecture, a number of signals are declared for connecting the components together. Each signal is given a name and has a specified type, in this case, std_logic.

```
library dld; use dld.gates.all;

architecture struct of vat_buzzer is

  signal below_25_0, temp_bad_0, wake_up_0 : std_logic;
  signal below_25_1, temp_bad_1, wake_up_1 : std_logic;

begin

  -- components for vat 0
  inv_0 : inv port map (above_25_0, below_25_0);
  or_0a : or2 port map (above_30_0, below_25_0, temp_bad_0);
  or_0b : or2 port map (temp_bad_0, low_level_0, wake_up_0);

  -- components for vat 1
  inv_1 : inv port map (above_25_1, below_25_1);
  or_1a : or2 port map (above_30_1, below_25_1, temp_bad_1);
  or_1b : or2 port map (temp_bad_1, low_level_1, wake_up_1);

  select_mux : mux2 port map (wake_up_0, wake_up_1,
                              select_vat_1, buzzer);

end architecture struct;
```

- Between the begin and end lines, the architecture contains a number of component instances
- Each has a label to distinguish it, and specifies which kind of component is instantiated.
- For example, inv_0 is an instance of the inv component, representing the inverter for vat 0 in the circuit

The vat buzzer Circuit, Showing signal names and component label





- For example, the inverter inv_0 has the input above_25_0 connected to its first port and the signal below_25_0 connected to its second port. We have also included some comments in this architecture to provide documentation.

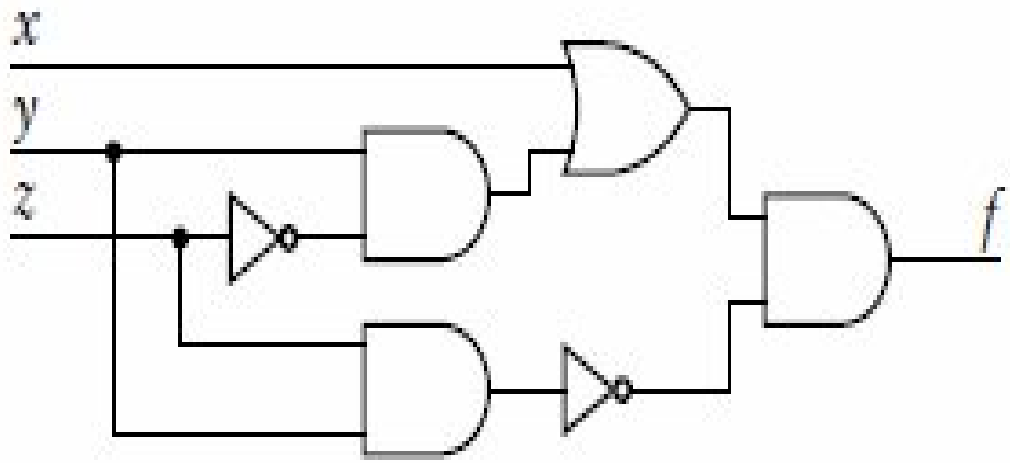
an alternative architecture:

```
architecture behavior of vat_buzzer is
begin
  buzzer <= low_level_1 or (above_30_1 or not above_25_1)
              when select_vat_1 = '1' else
            low_level_0 or (above_30_0 or not above_25_0);
end architecture behavior;
```

Combinational basics

Suppose we are to implement the following Boolean function using AND and OR gates and inverters:

$$f = (x + y \cdot \bar{z}) \cdot \overline{(y \cdot z)}$$



Bagaimana dalam VHDL ?

Solusi

EXAMPLE 2.6 Develop a VHDL model for a circuit that implements the Boolean equation of Example 2.5.

SOLUTION The equation refers to three inputs, x, y and z, and one output, f. We represent them as input and output ports in the entity declaration, as follows:

```
library ieee; use ieee.std_logic_1164.all;

entity circuit is
  port ( x, y, z : in  std_logic;
         f       : out std_logic );
end entity circuit;
```

The architecture contains an assignment statement that represents the Boolean equation, as follows:

```
architecture boolean_eqn of circuit is
begin
  f <= (x or (y and not z)) and not (y and z);
end architecture boolean_eqn;
```

Example

Develop a VHDL model for a combinational circuit that implements the following three Boolean equations, representing part of the control logic for an air conditioner:

Heater_on = temp_low · auto_temp + manual_heat

cooler_on =temp_high · auto_temp + manual_cool

fan_on = heater_on + cooler_on + manual_fan

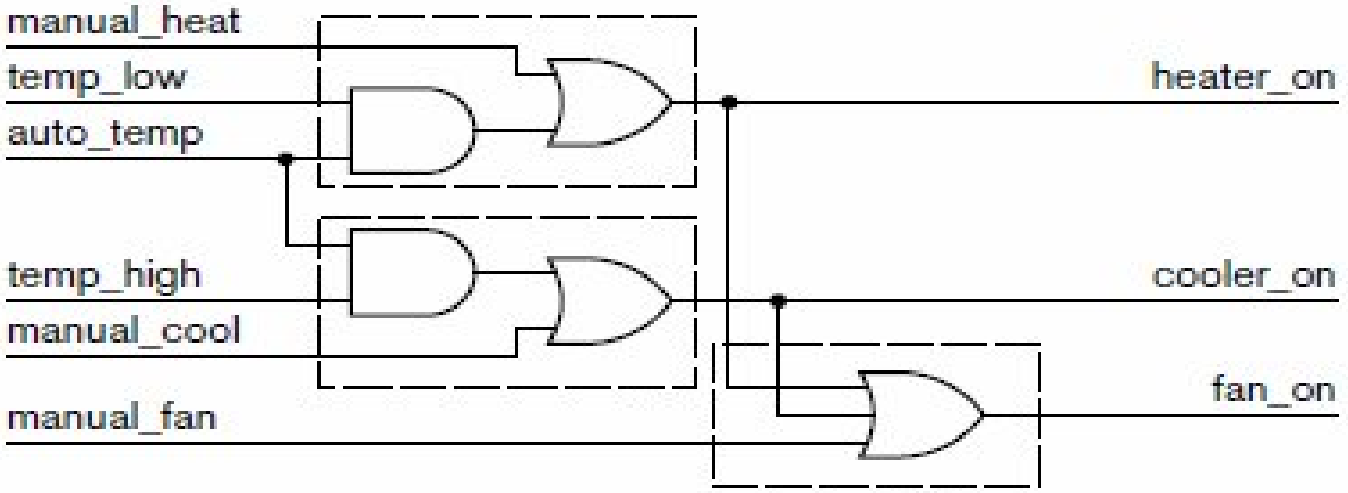
SOLUTION The entity declaration of the VHDL model defines the input and output ports as follows:

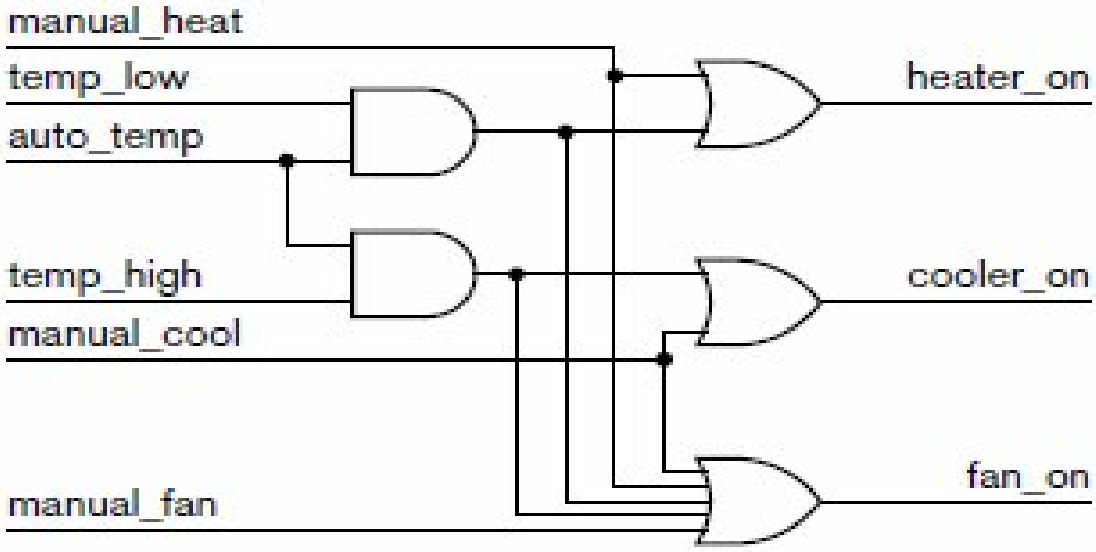
```
library ieee; use ieee.std_logic_1164.all;

entity aircon is
  port ( temp_low, temp_high, auto_temp       : in  std_logic;
         manual_heat, manual_cool, manual_fan : in  std_logic;
        heater_on, cooler_on, fan_on          : out std_logic );
end entity aircon;
```

The architecture body contains assignment statements for the Boolean equations:

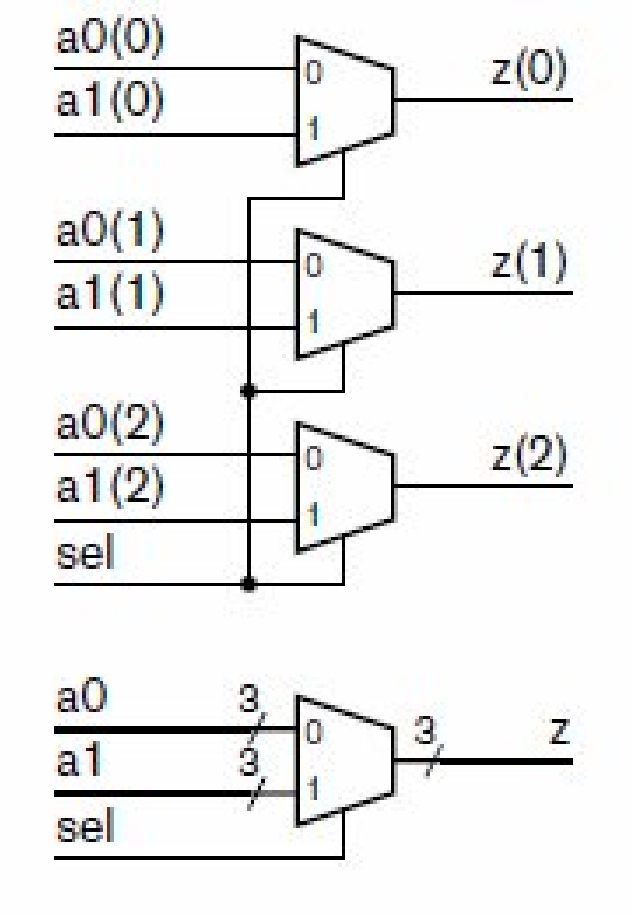
```
architecture eqns of aircon is
  signal heater_on_tmp, cooler_on_tmp : std_logic;
begin
  heater_on_tmp <= (temp_low  and auto_temp) or manual_heat;
  cooler_on_tmp <= (temp_high and auto_temp) or manual_cool;
  fan_on      <= heater_on_tmp or cooler_on_tmp or manual_fan;
  heater_on     <= heater_on_tmp;
  cooler_on     <= cooler_on_tmp;
end architecture eqns;
```

Circuits corresponding to the assignment statements for the air conditioner control logic.





Develop a VHDL model for the 3-bit 2-to-1 multiplexer.



Circuit Multiplexer

Simbol

SOLUTION The entity declaration and architecture are

```
library ieee; use ieee.std_logic_1164.all;

entity multiplexer_3bit_2_to_1 is
  port ( a0, a1 : in  std_logic_vector(2 downto 0);
         sel    : in  std_logic;
         z      : out std_logic_vector(2 downto 0) );
end entity multiplexer_3bit_2_to_1;

architecture eqn of multiplexer_3bit_2_to_1 is
begin
  z <= a0 when sel = '0' else
       a1;
end architecture eqn;
```

Numeric Basic

Develop a VHDL model of a 4-to-1 multiplexer that selects among four unsigned 6-bit integers.

SOLUTION The entity declaration is

```
library ieee;

use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity multiplexer_6bit_4_to_1 is
  port ( a0, a1, a2, a3 : in  unsigned(5 downto 0);
         sel            : in  std_logic_vector(1 downto 0);
         z              : out unsigned(5 downto 0) );
end entity multiplexer_6bit_4_to_1;
```

- The first line identifies the ieee library and the two packages std_logic_1164 and numeric_std that we want to use in our model. The input ports a0 through a3 and the output port z are all 6-bit unsigned vectors, indexed from 5 down to 0.
- We choose this index range so that the index of each bit in a vector corresponds to the power of its binary weight. The input port sel, used to select among the inputs, is of type std_logic_vector, since we are not interpreting it as representing a number. The corresponding architecture is:

```
architecture eqn of multiplexer_6bit_4_to_1 is
begin
  with sel select
    z <= a0 when "00",
         a1 when "01",
         a2 when "10",
         a3 when others;
end architecture eqn;
```

Kerjakan di rumah

Develop a VHDL behavioral model of an adder/ subtracter for 12-bit unsigned binary numbers. The circuit has data inputs x and y, a data output s, a control input mode that is 0 for addition and 1 for subtraction, and an output ovf_unf that is 1 when an addition overflow or a subtraction underflow occurs.

Sequential Basic

Develop a VHDL model for a pipelined circuit that computes the average of corresponding values in three streams of input values, a, b and c. The pipeline consists of three stages: the first stage sums values of a and b and saves the value of c; the second stage adds on the saved value of c; and the third stage divides by three. The inputs and output are all signed fixed-point numbers indexed from 5 down to -8.

Solution

SOLUTION The entity declaration is

```
library ieee;
use ieee.std_logic_1164.all, ieee.fixed_pkg.all;

entity average_pipeline is
  port ( clk     : in  std_logic;
         a, b, c : in  sfixed(5 downto -8);
         avg     : out sfixed(5 downto -8) );
end entity average_pipeline;
```

The architecture body is

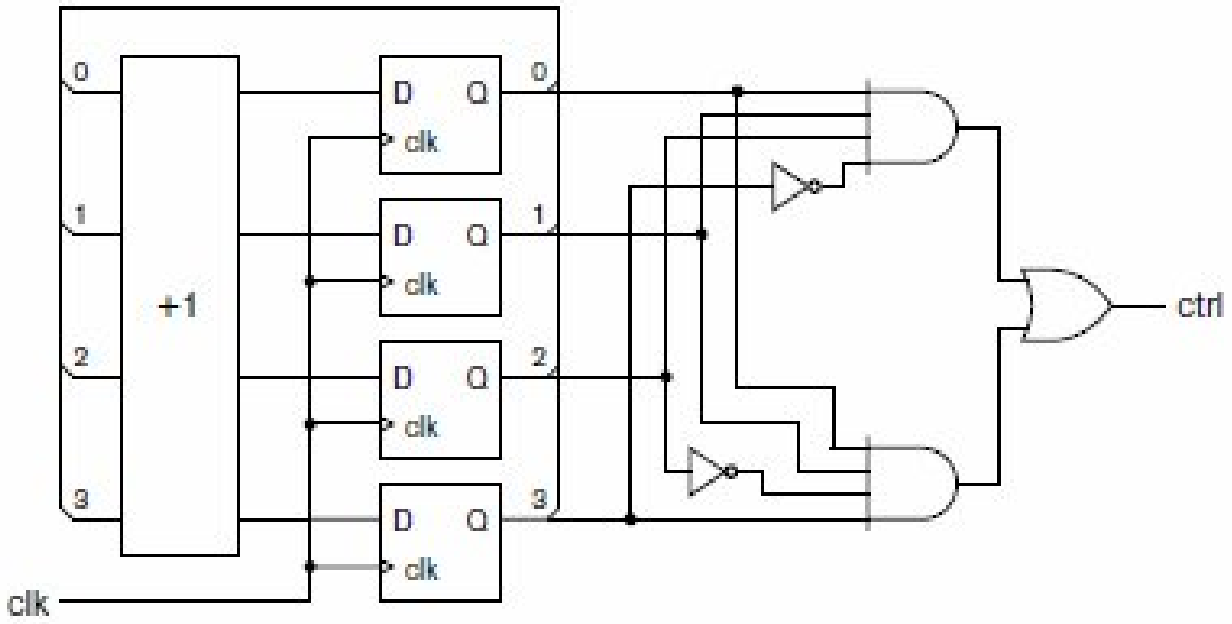
```
architecture rtl of average_pipeline is

  signal a_plus_b, sum, sum_div_3 : sfixed(5 downto -8);
  signal saved_a_plus_b,
         saved_c, saved_sum       : sfixed(5 downto -8);
begin

  a_plus_b <= a + b;

  reg1 : process (clk) is
  begin
    if rising_edge(clk) then
      saved_a_plus_b <= a_plus_b;
      saved_c  <= c;
    end if;
  end process reg1;

  sum <= saved_a_plus_b 1 saved c;
```

Cont’

```
  reg2 : process (clk) is
  begin
    if rising_edge(clk) then
      saved_sum <= sum;
    end if;
  end process reg2;

  sum_div_3 <= saved_sum *
               to_sfixed(1.0/3.0,
                         sum_div_3'left, sum_div_3'right);

  reg3 : process (clk) is
  begin
    if rising_edge(clk) then
      avg <= sum_div_3;
    end if;
  end process reg3;

end architecture rtl;
```

Design a circuit that counts 16 clock cycles and produces a control signal, ctrl, that is 1 during every eighth and twelfth cycle.

- solution We need a 4-bit counter, since 16=2^4. The counter counts from 0 to 15 and then wraps back to 0. During the eighth cycle, the counter value is 7 (01112), and during the twelfth cycle, the counter value is 11 (10112). We can generate the control signal by decoding the two required counter values and forming the logical OR of the decoded signal



The architecture contains a process that represents the counter. It is similar in form to a process for an edge-triggered register. The difference is that the value assigned to the count_value output on a rising clock edge is the incremented count value.

The assignment to count_value represents the update of the value stored in the register, and the addition of 1 represents the incrementer. The final assignment statement in the architecture represents the decoder.

The entity and architecture are

```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;

entity decoded_counter is
  port ( clk   : in std_logic;
         ctrl  : out std_logic );
end entity decoded_counter;

architecture rtl of decoded_counter is
  signal count_value : unsigned(3 downto 0);
```

```
begin
  counter : process (clk) is
  begin
    if rising_edge(clk) then
      count_value <= count_value + 1;
    end if;
  end process counter;

  ctrl <= '1' when count_value = "0111" or
                  count_value = "1011" else '0';
end architecture rtl;
```