Lesson 13

Sequential Logic

Mohd Saufy Rohmad
Technical Trainer and Consultant

Sequantial Logic

- The sequential logic in VHDL requires a clocked process.
- We have already seen how process can be used in lesson 12 for combinational logic.
- In case of sequential logic, the process will be used to generate flip-flops, latches, registers and other types of memories.
- Then what is the difference between a process of combinational logic and a process of sequential logic.
- The major difference is the sequential logic process is a clocked process; i.e. the clock is included in the sensitivity list.

D Flip-flop

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity flop is
port(
clk: in std_logic;
d_in: in std_logic;
d_out: out std_logic);
end flop;

architecture flop_arch of flop is
signal d_R: std_logic;
begin
process(clk)
begin
if clk'event and clk = '1' then
d_R <= d_in;
end if;
end process;
d_out <= d_R;
end flop_arch;
```

D Flip-flop

- This is a VHDL code of a simple D-flip flop.
- The process includes clk instead of including all the signals.
- This is an example of a clocked process.
- The if will be executed only when the clk changes, indicated by clk'event.
- The change is on the rising edge (not on falling edge) is indicated by clk = '1'. An alternative for the same flip-flop is,

  ```
  process(clk)
  begin
  if RISING_EDGE (clk) then
  d_R <= d_in;
  end if;
  end process;
  ```

- Here, the RISING_EDGE(clk) is the same as clk'event and clk = '1'.

D Latch

- A D-latch has almost similar VHDL code except clk'event and d_in in the input.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity flop is
port(
clk: in std_logic;
d_in: in std_logic;
d_out: out std_logic);
end flop;

architecture flop_arch of flop is
signal d_R: std_logic;
begin
process(clk,d_in)
begin
if clk = '1' then
d_R <= d_in;
end if;
end process;
d_out <= d_R;
end flop_arch;
```

D Latch

- Here, we do not say what to do when clk is not 1, thats why the compiler creates an inferred latch.
- We recommend to avoid using latch unless absolutely necessary

Registers

- A register is nothing but a combination of flip-flops.
- Therefore, it has a very similar structure to D flip flops, but the inputs, outputs and signals only works on vectors.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity regis is
port(
clk: in std_logic;
d_in: in std_logic_vector(7 downto 0);
d_out: out std_logic_vector(7 downto 0));
end regis;

architecture regis_arch of flop is
signal d_R: std_logic_vector(7 downto 0);
begin
process(clk)
begin
if clk'event and clk = '1' then
d_R <= d_in;
end if;
end process;
d_out <= d_R;
end regis_arch;
```

Registers with Resets

- There can be two types of resets. Synchronous and asynchronous.
- Asynchronous Reset – next slide
- Synchronous Reset – next 2 slide

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity regis is
port(
clk: in std_logic;
rst: in std_logic;
d_in: in std_logic_vector(7 downto 0);
d_out: out std_logic_vector(7 downto 0));
end regis;
architecture arch1 of regis is
signal d_R: std_logic_vector(7 downto 0); -- output register
Begin
process(clk, rst)
begin
if rst = '0' then
d_R <= (others => '0');
elsif clk'event and clk = '1' then
d_R <= d_in;
end if;
end process;
d_out <= d_R;
end arch1;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity regis is
port(
clk: in std_logic;
rst: in std_logic;
d_in: in std_logic_vector(7 downto 0);
d_out: out std_logic_vector(7 downto 0));
end regis;

architecture arch1 of regis is
signal d_R: std_logic_vector(7 downto 0); -- output register
begin
process(clk)
begin

if clk'event and clk = '1' then
    if rst = '1' then
        d_R <= (others => '0');
    else
        d_R <= d_in;
    end if;
end if;
end process;
d_out <= d_R;
end arch1;
```

Thank You