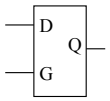
A D-Latch in VHDL

```
entity mydlatch is port (
      signal d, g:   in std_logic;
      signal     q:   out std_logic
);
end mydlatch;
architecture behavior of mydlatch is
-- rising edge triggered DFF
 state:  process (g, d)
  if (g = '1') then
      q < = d;
  end if;
end process;
end behavior;
```

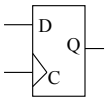


Both d, g on sensitivity list (when g=1, want changes on D to affect output)

No default assignment for 'q'; only assigned when gate is high.

A Dff in VHDL

```
entity mydff is port (
      signal d, clk:   in std_logic;
      signal     q:   out std_logic
);
end mydff;
architecture behavior of mydff is
-- rising edge triggered DFF
 state:  process (clk)
  if (clk'event and clk = '1') then
      q < = d;
  end if;
 end process;
end behavior;
```



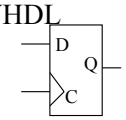
Input d is not on sensitivity list because 'q' only changes when 'clk' changes.

Assignment 'protected' by clock edge. So DFF is synthesized.

Rising edge

Another way to do a Dff in VHDL

```
entity mydff is port (
      signal d, clk:   in std_logic;
      signal     q:   out std_logic
);
end mydff;
architecture behavior of mydff is
-- rising edge triggered DFF
 state:  process
  wait until (clk'event and clk = '1');
  q < = d;
 end process;
end behavior;
```



No sensitivity list

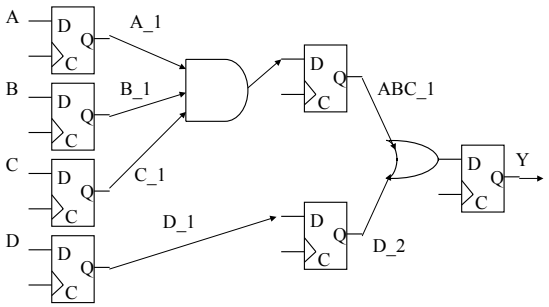
Wait for Rising edge

Assignment after rising edge clock so DFF is synthesized.

Comments on Examples

- Process with a clock in sensitivity list or a 'wait' on a clock edge is called a 'clocked process'.
- ALL assignments that are protected by a clock edge will have a DFFs placed on the logic outputs.
- Can very easily insert DFFs between blocks of logic in VHDL.

An example



Entity Declaration

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

entity plogic is
  port (  signal a,b,c,d:  in std_logic;
          signal clk:  in std_logic;
          signal y: out std_logic
       );
 end plogic;
```

Architecture

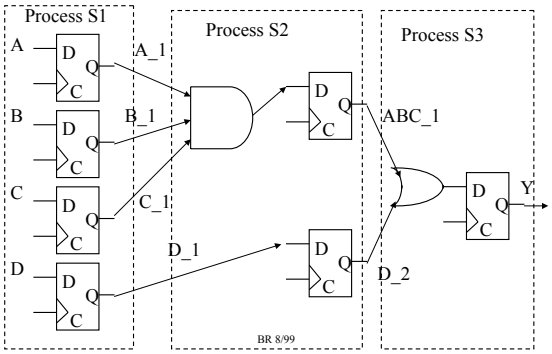
```
architecture a of plogic is
  signal a_1, b_1, c_1 :std_logic;
  signal d_1,d_2, abc_1: std_logic;
begin
  s1: process
    begin
    wait until clk'event and clk='1';
     a_1 <= a; b_1 <= b; c_1 <= c;  d_1 <= d;
  end process s1;

  s2: process
  begin
    wait until (clk'event and clk = '1');
      abc_1 <= a_1 and b_1 and c_1;
      d_2 <= d_1;
  end process s2;

  s3: process
  begin
    wait until (clk'event and clk = '1');
     y <= abc_1 or d_2;
  end process s3;
end a;
```

Each process defines a block of logic plus DFFs.

Could have used 'if' statements with clk in sensitivity list as well.

Logic in process can be as complex as you wish.

Processes S1, S2, S3



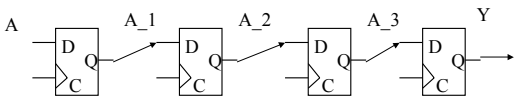
A Problem with VHDL Semantics vs Maxplus Synthesis……

```
architecture behavior of pipetest is
    signal a_1, a_2, a_3: std_logic;

begin
  process
    begin
    wait until clk'event and clk='1';
     a_1 <= a;
     a_2 <= a_1;   -- what happens??
     a_3 <= a_2;
     y <= a_3;
  end process;
end behavior;
```

What logic should be synthesized for this architecture?

VHDL Definition of Signal Update

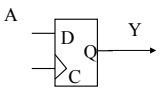
VHDL defines that SIGNAL UPDATE within a PROCESS takes place after the process is EXITED. This means that that the logic synthesized should act like the following:



On the clock edge, signal A_1 is updated with A. However, according to VHDL semantics, the signal does not change its value until process EXIT. This means that A_2 will get the old value of A_1, as is shown above.

A Problem In Maxplus Synthesis

The Synopsys and Synplicity synthesis tools do synthesize to 4 Dffs; this is correct. Unfortunately, Altera Maxplus synthesizes to just:



This problem has been corrected in Versions 9.3 and later!!!!!

This is incorrect, but at the same time, the signal updating rules that VHDL uses can be confusing. The code *looks like it* should produce the above logic!!!

What to do????

- If you really want 4 FFs in a chain, then use four separate processes, with *wait* statements in each process. This way, your intentions will be clear and Maxplus will produce the correct logic
- If you need temporary placeholders for intermediate results, then use VARIABLES.
  - Variables can only be declared within processes
  - Variable update semantics act like variable update in normal programming languages. Variables are updated IMMEDIATELY.
  - Variable assignment uses the ' := ' operator. Signal assignment uses ' <= ' .

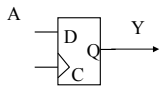
VHDL Variables

```
architecture behavior of pipetest is
 begin
 process
   variable a_1, a_2, a_3: std_logic;
  begin
   wait until clk'event and clk='1';
    a_1 := a;
    a_2 := a_1;   -- what happens??
    a_3 := a_2;
    y <= a_3;
  end process;

end behavior;
```

Variable declaration

Variable assignment

Synthesizes correctly to :

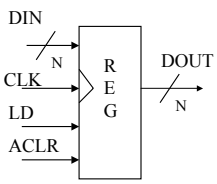


Variables vs. Signals

- ALWAYS use variables for temporary values within processes
  - However, for the RTL done in this class I doubt if you will ever need to use variables.
- Use SIGNALS for passing information between processes
  - Variables cannot be used outside of processes
  - A variable '*x*' in a process cannot be accessed by other processes. Can only be used within the process it is declared.

Registers

The most common sequential building block is the register. A register is N bits wide, and has a load line for loading in a new value into the register.



Register contents do not change unless LD = 1 on active edge of clock.

A DFF is NOT a register! DFF contents change every clock edge.

ACLR used to asynchronously clear the register

VHDL for 8-bit Register (Entity)

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

-- 8 bit register
entity reg8 is
 port (  clk:  in std_logic;
         reset: in std_logic; -- async reset
         ld: in std_logic;    -- synchronous load
         din: in std_logic_vector(7 downto 0);
        -- outputs
         dout: out std_logic_vector(7 downto 0)
    );
end reg8;
```


VHDL for 8-bit Register (Architecture)

```
architecture a of reg8 is
 begin
  main:process(clk, reset)
   begin
    if (reset = '1') then
          dout <= "00000000";
    elsif (clk'event and clk='1') then
        -- rising edge of clock
          if (ld = '1') then
           dout <= din;
          end if;
     end if;
   end process main;
end a;
```

Asynchronous Reset

Change register state on rising edge and assertion of load line.