

# Signals & Variables (3A)

---

## Simulation & Synthesis

Copyright (c) 2012 Young W. Lim.

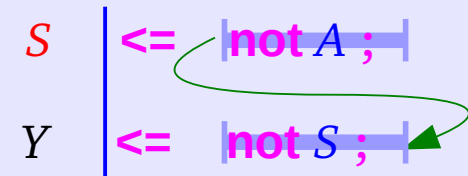
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to [youngwlim@hotmail.com](mailto:youngwlim@hotmail.com).

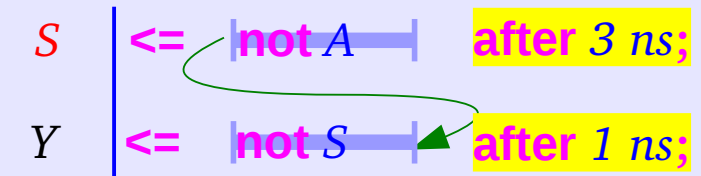
This document was produced by using OpenOffice and Octave.

# Sequential Assignment (1)

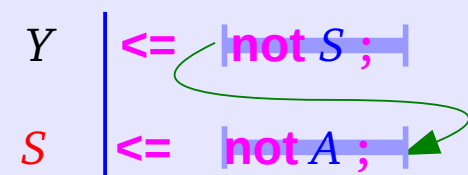
```
process (A)
  signal S: std_logic ;
begin
  S <= not A ;
  Y <= not S ;
end process;
```



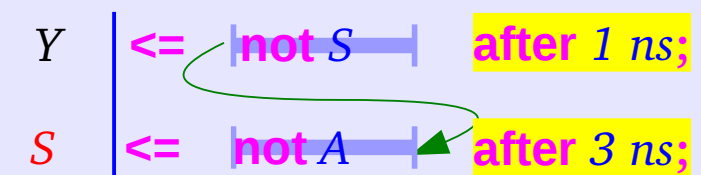
```
process (A)
  signal S: std_logic ;
begin
  S <= not A ; after 3 ns;
  Y <= not S ; after 1 ns;
end process;
```



```
process (A)
  signal S: std_logic ;
begin
  Y <= not S ;
  S <= not A ;
end process;
```

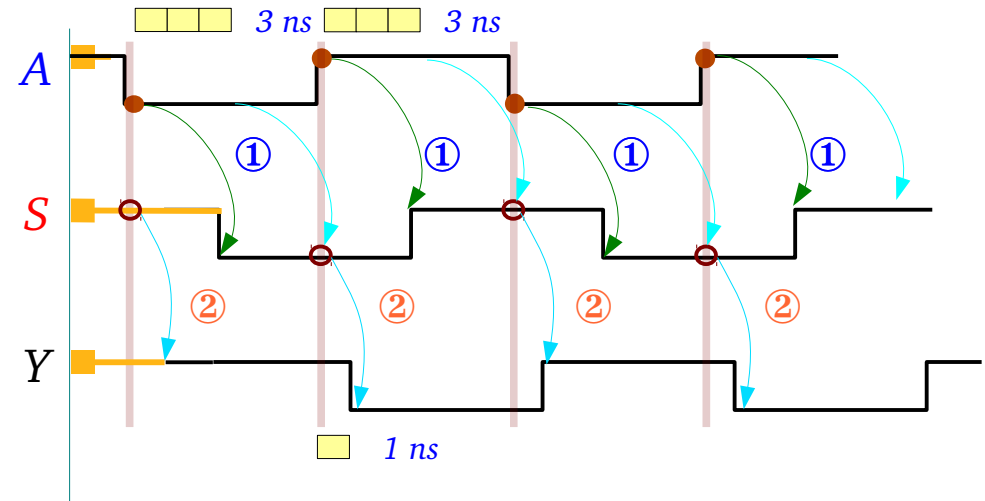


```
process (A)
  signal S: std_logic ;
begin
  Y <= not S ; after 1 ns;
  S <= not A ; after 3 ns;
end process;
```

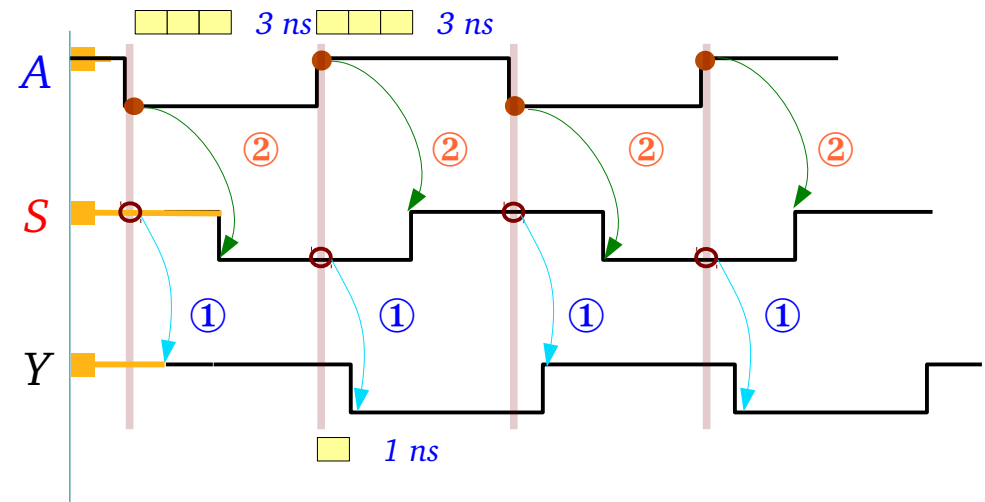


# Sequential Assignment (1) - with non-zero delay

```
process (A)
  signal S: std_logic ;
begin
  S <= A after 3 ns;
  Y <= S after 1 ns;
end process;
```

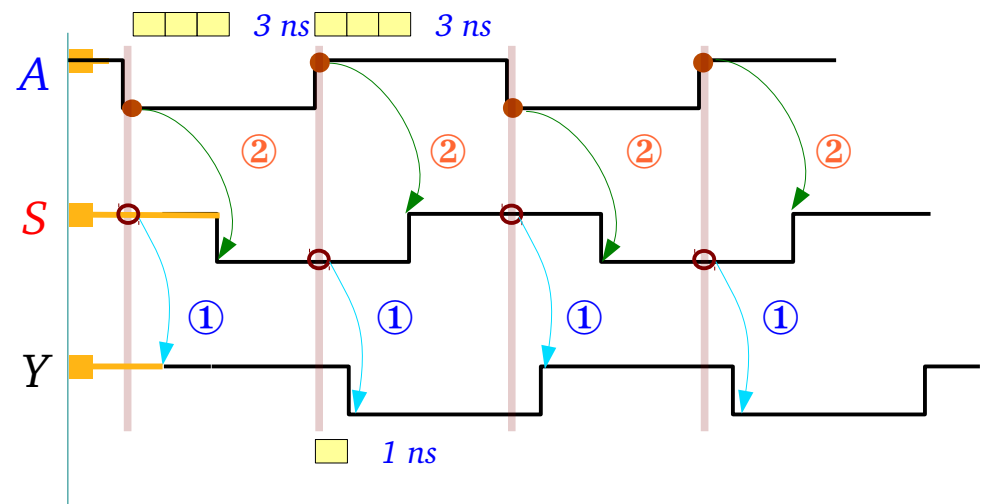
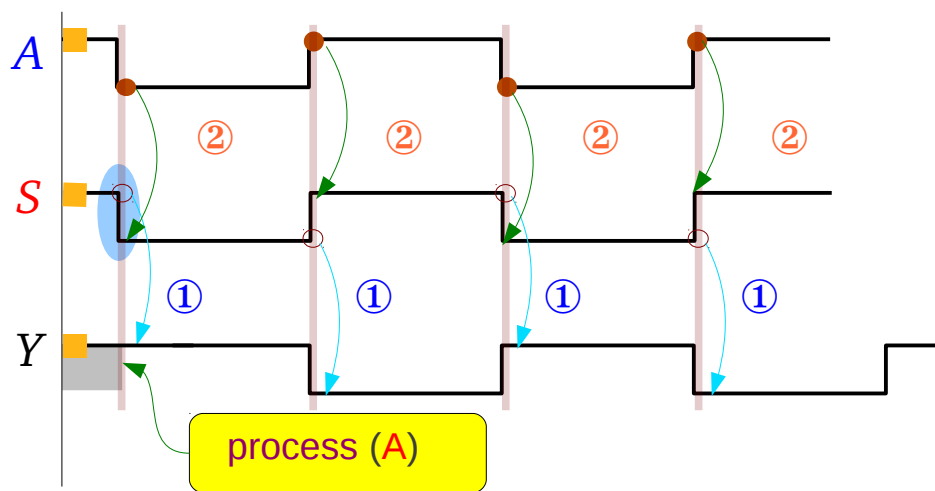
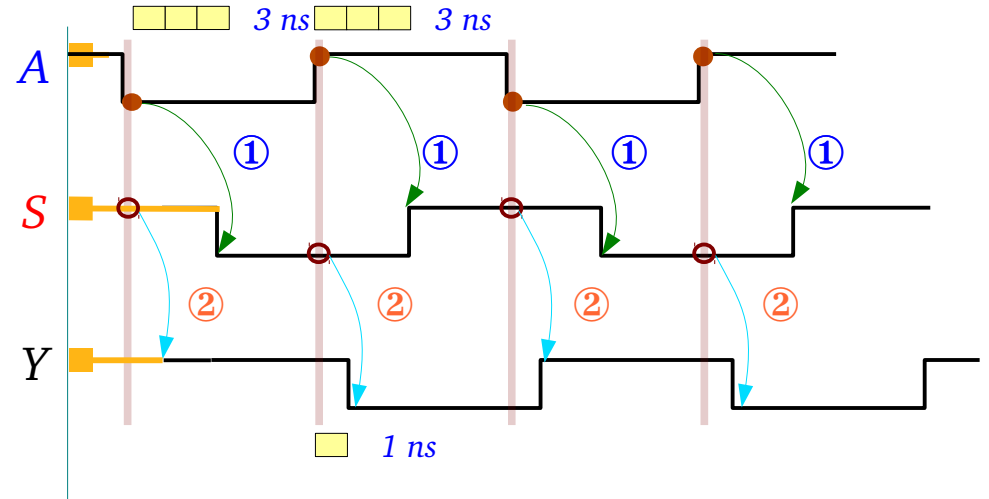
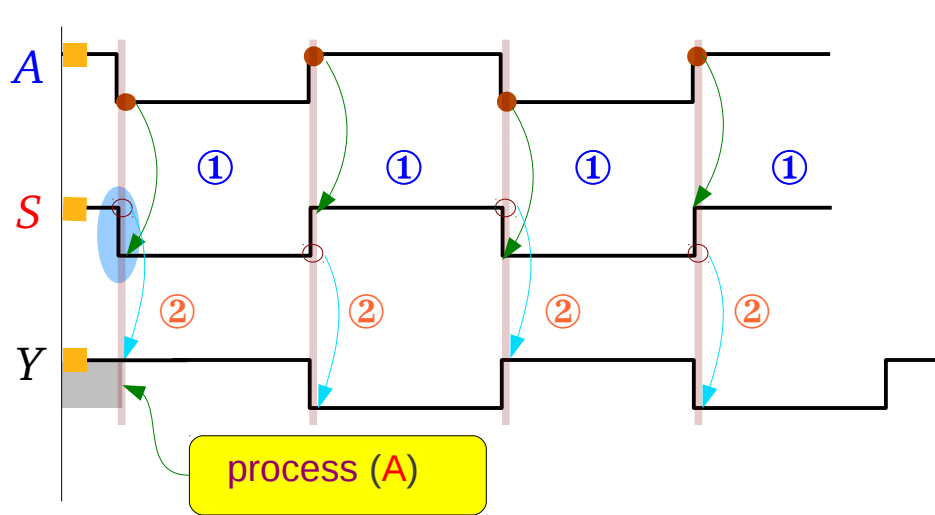


```
process (A)
  signal S: std_logic ;
begin
  Y <= S after 1 ns;
  S <= A after 3 ns;
end process;
```



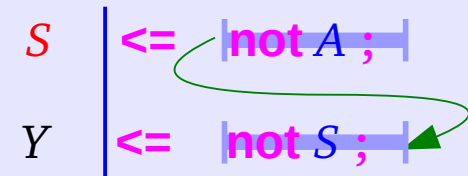
# Sequential Assignment (1) - with zero delay

with zero delay

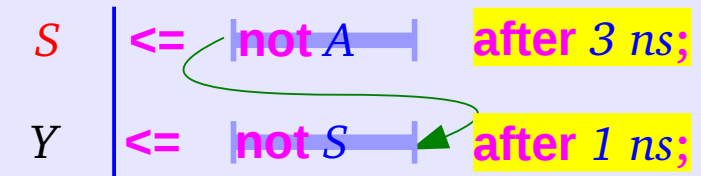


# Sequential Assignment (2)

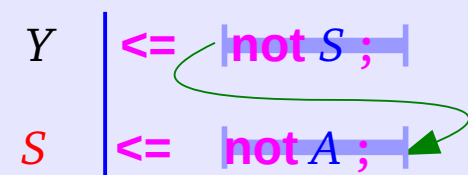
```
process (A, S)
  signal S: std_logic ;
begin
  S <= not A ;
  Y <= not S ;
end process;
```



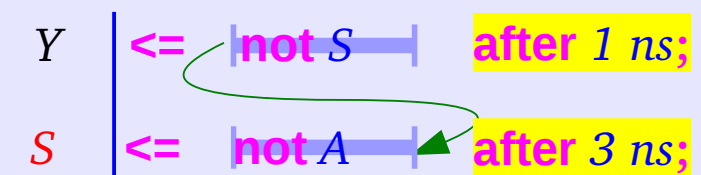
```
process (A, S)
  signal S: std_logic ;
begin
  S <= not A ; after 3 ns;
  Y <= not S ; after 1 ns;
end process;
```



```
process (A, S)
  signal S: std_logic ;
begin
  Y <= not S ;
  S <= not A ;
end process;
```

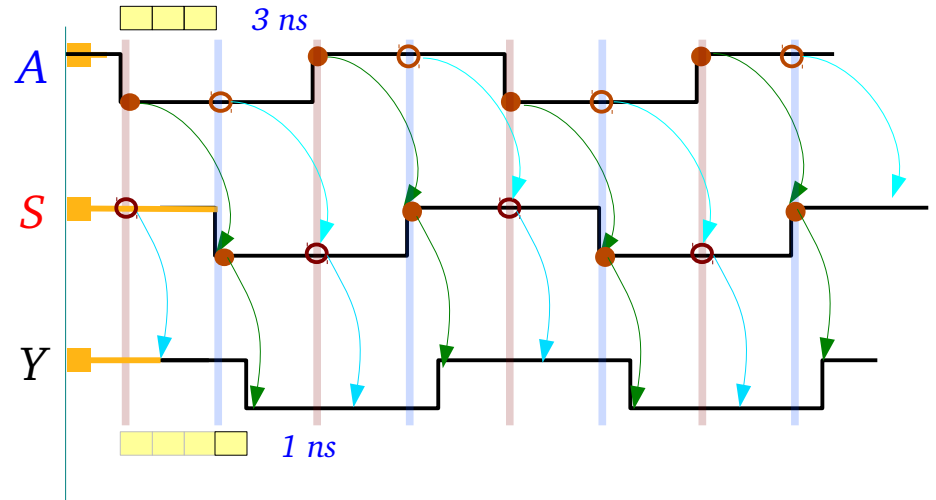


```
process (A, S)
  signal S: std_logic ;
begin
  Y <= not S ; after 1 ns;
  S <= not A ; after 3 ns;
end process;
```

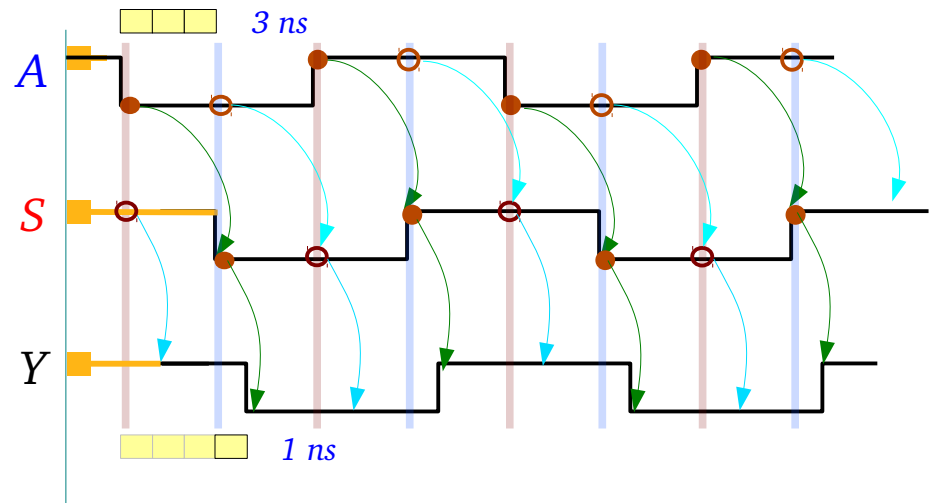


# Sequential Assignment (2) - with non-zero delay

```
process (A, S)
  signal S: std_logic ;
begin
  S <= not A after 3 ns;
  Y <= not S after 1 ns;
end process;
```

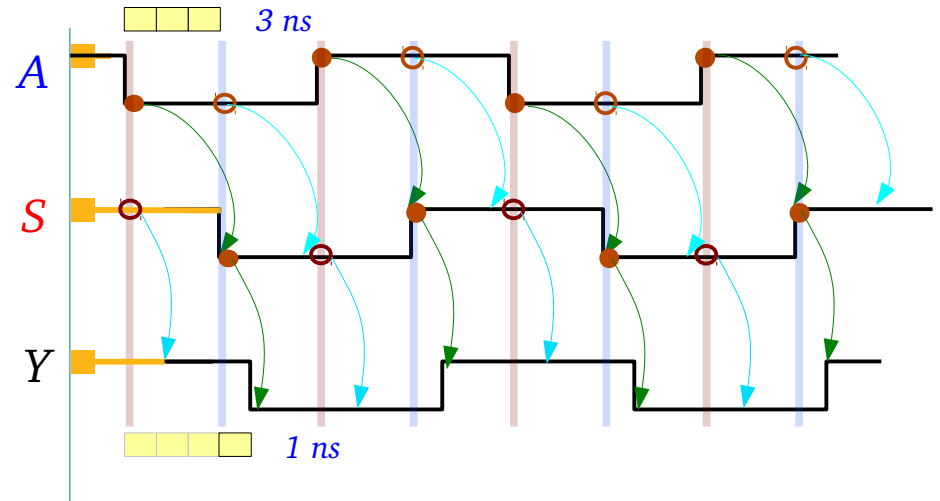
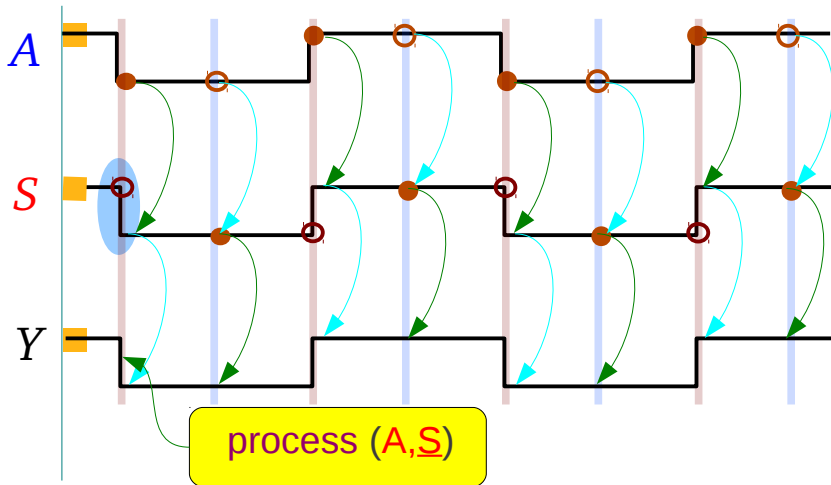
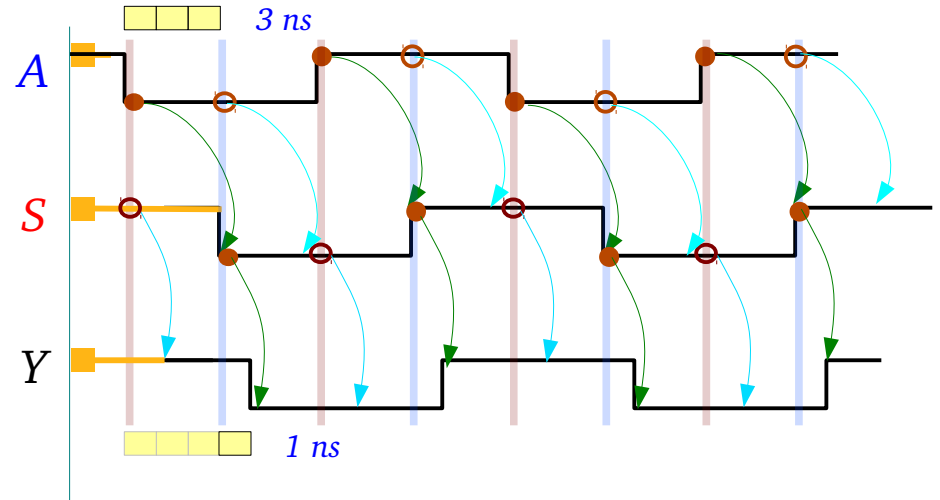
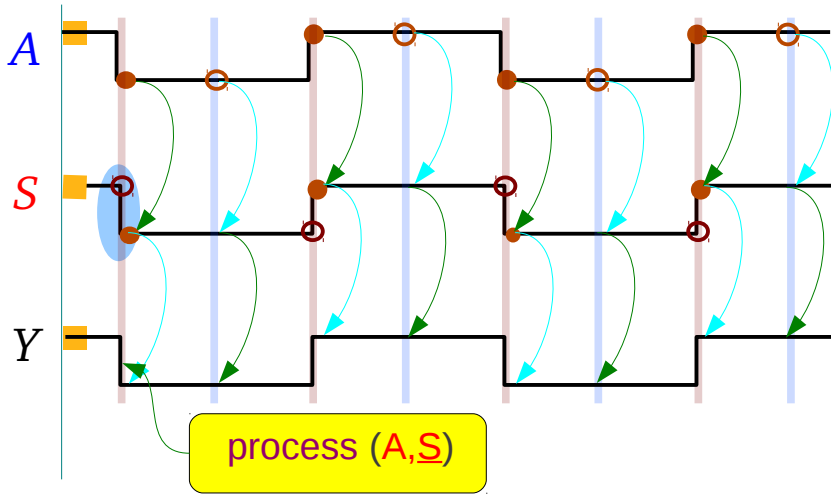


```
process (A, S)
  signal S: std_logic ;
begin
  Y <= not S after 1 ns;
  S <= not A after 3 ns;
end process;
```



# Sequential Assignment (2) - with zero delay

with zero delay





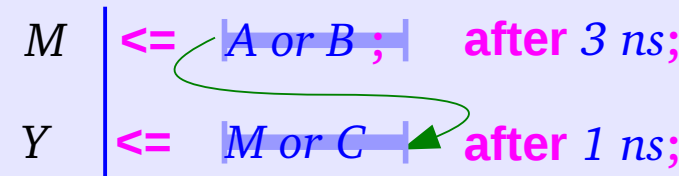
# Ex 1

```
process (A, B, C)
  variable M: std_logic ;
begin
  M := A or B ;
  Y <= M or C after 1 ns;
end process;
```

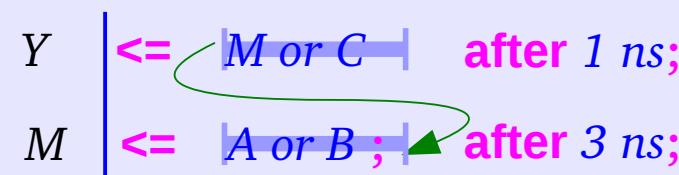
```
process (A, B, C)
  variable M: std_logic ;
begin
  Y <= M or C after 1 ns;
  M := A or B ;
end process;
```

# Ex 2

```
process (A, B, C, M)
  signal M: std_logic ;
begin
  M <= A or B ; after 3 ns;
  Y <= M or C ; after 1 ns;
end process;
```

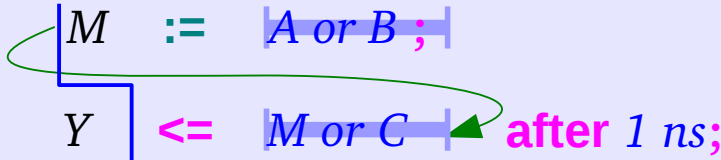


```
process (A, B, C, M)
  signal M: std_logic ;
begin
  Y <= M or C ; after 1 ns;
  M <= A or B ; after 3 ns;
end process;
```



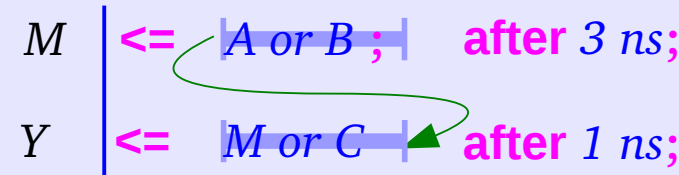
# Ex 3

```
process (Clock)
  variable M: std_logic ;
begin
  if rising_edge(Clock) then
    M := A or B ;
    Y <= M or C after 1 ns;
  end if;
end process;
```



# Ex 4

```
process (Clock)
  signal M: std_logic ;
begin
  if rising_edge(Clock) then
    M <= A or B ; after 3 ns;
    Y <= M or C ; after 1 ns;
  end if;
end process;
```



# Ex 4

```
process (Clock)
    signal M: std_logic ;
begin
    if rising_edge(Clock) then
        M <= A or B ; after 3 ns;
        Y <= M or C ; after 1 ns;
    end if;
end process;
```

# Variable & FlipFlop (1)

```
process (Clock)
  variable M, N: std_logic
begin
  if rising_edge(Clock) then
    Y <= N;
    N := M;
    M := X;
  end if;
end process;
```

```
process (Clock)
  variable M, N: std_logic
begin
  if rising_edge(Clock) then
    M := X;
    N := M;
    Y <= N;
  end if;
end process;
```

# Variable & FlipFlop (2)

```
process (Clock)
  variable A : std_logic (3 downto 0) ;
begin
  if rising_edge(Clock) then
    for i in 3 downto 0 loop
      A(i) := A(i-1) ;
    end loop
    A(0) := Data ;
    YA := A ;
  end if;
end process;
```

```
process (Clock)
  variable B : std_logic (0 to 3) ;
begin
  if rising_edge(Clock) then
    for i in 0 to 3 loop
      B(i) := B(i-1) ;
    end loop
    B(0) := Data ;
    YB := B ;
  end if;
end process;
```

# Variable & FlipFlop (3)

```
process (Clock)
  signal A : std_logic (3 downto 0) ;
begin
  if rising_edge(Clock) then
    for i in 3 downto 0 loop
      A(i)    <=  A(i-1) ;
    end loop
    A(0)    <=  Data ;
    YA     <=  A ;
  end if;
end process;
```

```
process (Clock)
  signal B : std_logic (0 to 3) ;
begin
  if rising_edge(Clock) then
    for i in 0 to 3 loop
      B(i)    <=  B(i-1) ;
    end loop
    B(0)    <=  Data ;
    YB     <=  B ;
  end if;
end process;
```



## References

- [1] <http://en.wikipedia.org/>
- [2] J. V. Spiegel, VHDL Tutorial,  
[http://www.seas.upenn.edu/~ese171/vhdl/vhdl\\_primer.html](http://www.seas.upenn.edu/~ese171/vhdl/vhdl_primer.html)
- [3] J. R. Armstrong, F. G. Gray, Structured Logic Design with VHDL
- [4] Z. Navabi, VHDL Analysis and Modeling of Digital Systems
- [5] D. Smith, HDL Chip Design
- [6] <http://www.csee.umbc.edu/portal/help/VHDL/stdpkg.html>
- [7] VHDL Tutorial - VHDL online [www.vhdl-online.de/tutorial/](http://www.vhdl-online.de/tutorial/)