

# VHDL Process Revisited

Last updated 4/2/26

# VHDL Process Revisited

- The process construct allows portions of the VHDL code to be executed only under certain conditions
  - The code is only executed in **simulations** when a signal in its sensitivity list has changed
    - See example
- The process construct allows the more flexible **if/else** and **case** statements to be used
- The process construct **ONLY** updates signals in **simulations** at the end of the process
  - See example

# VHDL Process Revisited

- Structure

## Optional label

Cannot be a duplicate  
of any other process  
label or signal name

```
label process (sensitivity list)  
begin  
    hdl code  
end process;
```

## Sensitivity list

The process block is not  
evaluated unless a  
signal in the sensitivity  
list has changed

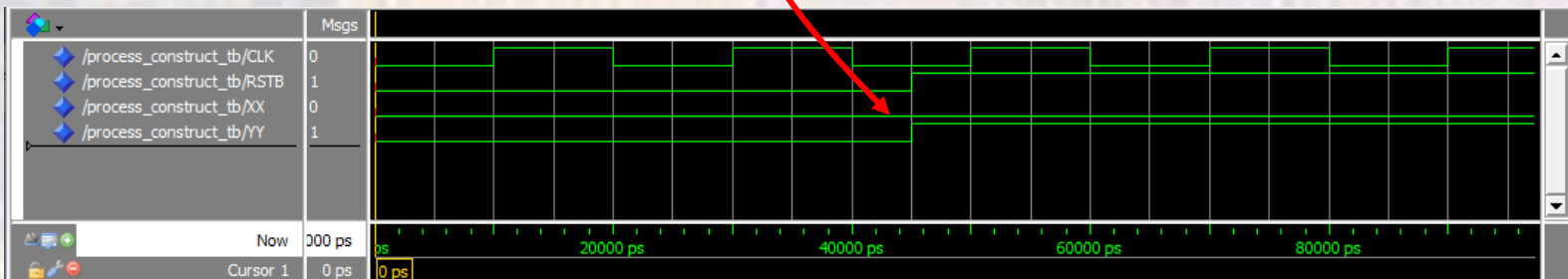
# VHDL Process Revisited

- The code is only executed when a signal in its sensitivity list has changed – **correct** version

```
process(i_clk, i_rstb)
begin
  if(i_rstb = '0') then
    x <= '0';
    y <= '0';
  elsif(i_rstb = '1') then
    y <= '1';
  elsif(rising_edge(i_clk)) then
    x <= '1';
    y <= x;
  end if;
end process;
```

Note: *i\_rstb* IS in sensitivity list

Expect Y → 1 when *i\_rstb* → 1



# VHDL Process Revisited

- The code is only executed when a signal in its sensitivity list has changed – **incorrect** version

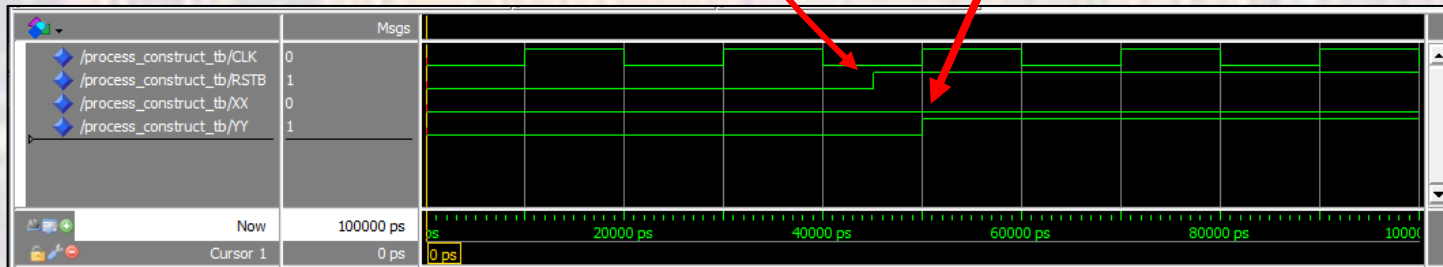
```
process(i_clk)
begin
  if(i_rstb = '0') then
    x <= '0';
    y <= '0';
  elsif(i_rstb = '1') then
    y <= '1';
  elsif(rising_edge(i_clk)) then
    x <= '1';
    y <= x;
  end if;
end process;
```

Note: i\_rstb is **NOT** in sensitivity list

Expect Y → 1 when i\_rstb → 1  
but

The process is only called by i\_clk  
so

the change in i\_rstb is not seen until the  
next clock edge

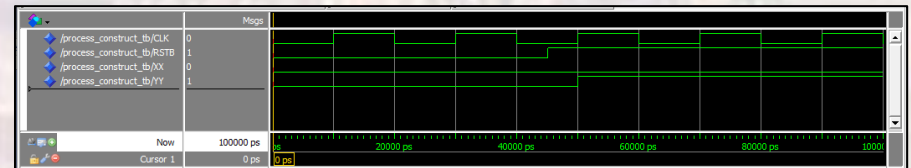
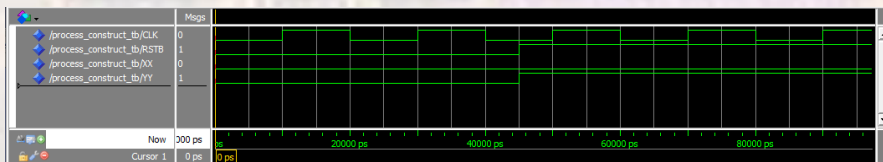


# VHDL Process Revisited

- The code is only executed when a signal in its sensitivity list has changed

```
process(i_clk, i_rstb)
begin
  if(i_rstb = '0') then
    x <= '0';
    y <= '0';
  elsif(i_rstb = '1') then
    y <= '1';
  elsif(rising_edge(i_clk)) then
    x <= '1';
    y <= x;
  end if;
end process;
```

```
process(i_clk)
begin
  if(i_rstb = '0') then
    x <= '0';
    y <= '0';
  elsif(i_rstb = '1') then
    y <= '1';
  elsif(rising_edge(i_clk)) then
    x <= '1';
    y <= x;
  end if;
end process;
```



- These create the **same RTL solution** but **different simulation solutions** – we verify by simulation so the final design may not work as expected

# VHDL Process Revisited

- Processes update sequential signals at the end of the process

```
process(i_clk, i_rstb)
begin
  if(i_rstb = '0') then
    x <= '0';
    y <= '0';
  elsif(rising_edge(i_clk)) then
    x <= '1';
    y <= '1';
  end if;
end process;
```

set y → 1

These appear to do the same thing:  
Set x and y → 1 at the same time

```
process(i_clk, i_rstb)
begin
  if(i_rstb = '0') then
    x <= '0';
    y <= '0';
  elsif(rising_edge(i_clk)) then
    x <= '1';
    y <= x;
  end if;
end process;
```

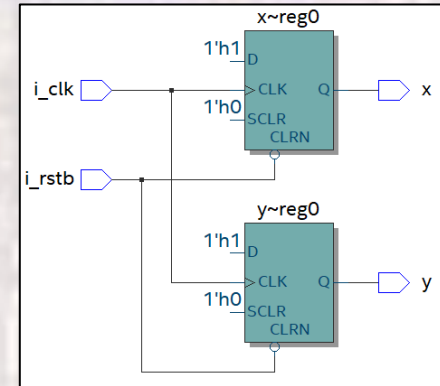
set y → 1 via x already set to 1

# VHDL Process Revisited

- Processes update sequential signals at the end of the process

```
process(i_clk, i_rstb)
begin
  if(i_rstb = '0') then
    x <= '0';
    y <= '0';
  elsif(rising_edge(i_clk)) then
    x <= '1';
    y <= '1';
  end if;
end process;
```

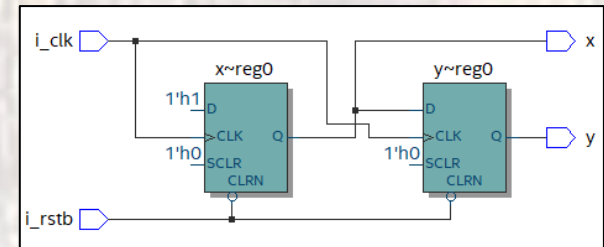
Expected result:  
 $x, y \rightarrow 1$  at the same time



These appear to do the same thing:  
Set  $x$  and  $y \rightarrow 1$  at the same time

```
process(i_clk, i_rstb)
begin
  if(i_rstb = '0') then
    x <= '0';
    y <= '0';
  elsif(rising_edge(i_clk)) then
    x <= '1';
    y <= x;
  end if;
end process;
```

Unexpected (but correct) result:  
 $y$  is not updated to the value of  $x$  until the next clk



$X$  has not been changed to 1 at this point  
It only becomes 1 at the end of the process

# VHDL Process Revisited

- **Warning – Warning – Warning**

- If you do not complete an **if-else** with an **else**, a latch will be created
- If you do not cover all cases in a **case** statement, a latch will be created
- All paths/cases must be covered
- The compiler will always warn you it created a latch

**We do not want latches - EVER**

I can see a latch in an RTL diagram from a mile away

The FF construct is one of very few exceptions

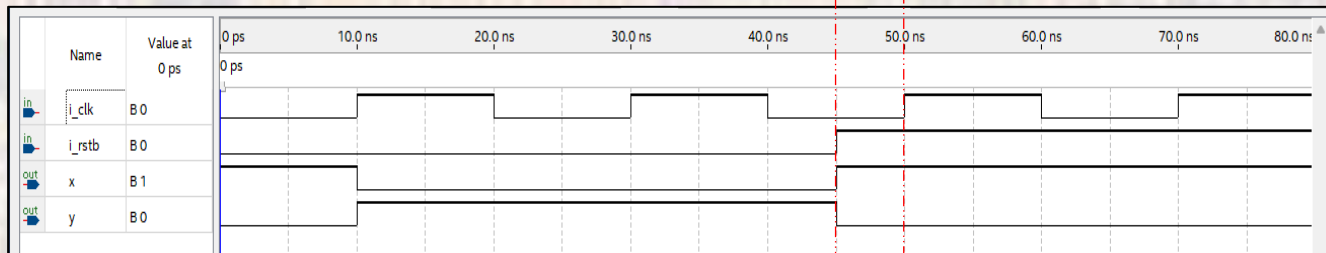
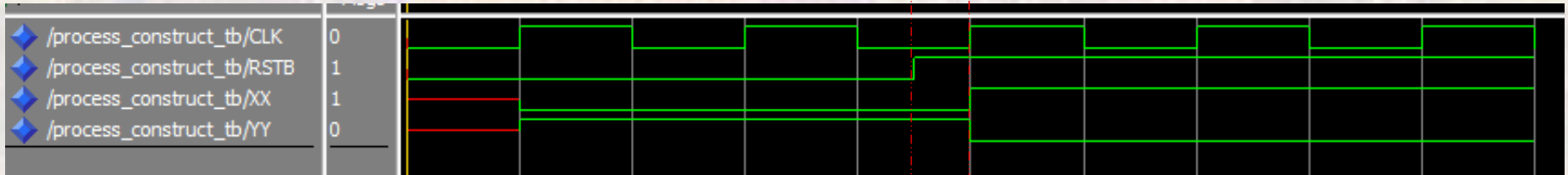
## i\_rstb not in the sensitivity list

```
process(i_clk)
begin
  if(i_rstb = '1') then
    x <= '1';
    y <= '0';
  elsif(rising_edge(i_clk)) then
    x <= '0';
    y <= '1';
  end if;
end process;
```

## Instructor Note

The University Waveform Viewer does not properly simulate this VHDL sensitivity list restriction

- testbench properly shows
- 1) unknown at beginning
  - 2) x,y flipping on clk edge



- vwf improperly shows
- 1) 1, 0 at beginning
  - 2) x,y flipping on rstb change