Generic Example Process:

**Process** – Collection of sequential statements that execute in parallel with other concurrent statements and other processes

* Always running or suspended
* Creates a “box” that listens for a signals (wires)
* When the signal changes the box is told to run

\*\*\* Processes are scary!

* Add memory
* Make circuit hard to visualized

🡺 If you want a Flip-Flop you need a process

process (*signal name, signal name, …., signal name*)

*Type declarations*

*Variable declarations*

*Constant declarations*

*Functions declarations*

*Procedure definitions*

begin

*sequential statement*

*…*

*sequential statement*

end process;

Specific Example Process: We get:

process(A, B)

begin

Y <= A and B;

end process;

What if we have assignments not in a process: We get:

* It all happens at the same time!
* Not a good situation!

X = Illegal Value

Y <= A **or** C;

Y <= A **and** B;

Y <= C **and** D;

So we make a process: We get:



**process(A, B, C, D)**

**begin**

**Y <= A or C;**

**Y <= A and B;**

**Y <= C and D;**

**end process;**

process ( ) begin

* Things are scheduled
* Rest of stuff is ignored
* Y is never assigned the first two values
* Schedule is overwritten

If C=1 and D becomes 0 🡺 Y=0

If D removed from sensitivity list C=1 and D changes from 0 to 1 🡺 Y=0

\*\*\*Y must remember its old value

end process;

**Rules for processes:**

1. Think before using variables (don’t use them if you can help it)
2. Avoid “innovative” use of language constructs
3. \*Avoid overriding a signal in a process (don’t assign a value twice)
4. \*Only use processes for sequential circuits

\*Indicates a rule for this class only

If statements:

if *boolean expression* then *sequential statement*

elsif *boolean expression* then *sequential statement* elsif executed only if boolean

… expression true and previous

elsif *boolean expression* then *sequential statement* boolean expressions were false

else *sequential statement*

end if;

* Note: Do not use

\*\*\* clk’event and clk = ‘1’

Only use rising\_edge(clk)

🡺 can only use if inside a process

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Flip\_Flop is

Port ( clk, D : in STD\_LOGIC;

Q : out STD\_LOGIC);

end Flip\_Flop;

architecture Behavioral of Flip\_Flop is

begin

process(clk)

begin

if (rising\_edge(clk)) then

Q <= D;

end if;

end process;

end Behavioral;

entity is

port (

);

end;

architecture of is

Build stuff

begin

process ( )

begin

Runs when clk changes

if then

Only when 0🡪1

🡺 Memory

end if;

* How do we make a Latch:

if (clk = '0') then

Q <= D;

end if;

end process;

end ;

process(A, B)

variable tmp : STD\_LOGIC;

begin

tmp := '0';

tmp := tmp or A;

tmp := tmp or B;

Y <= tmp

end process;

process ( ) Result:

 variable ;

begin

end process;

process(A, B)

variable tmp : STD\_LOGIC;

begin

tmp <= '0';

tmp <= tmp or A;

tmp <= tmp or B;

Y <= tmp;

end process;

process ( ) Result:

begin

end process;