香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

Lecture 08: Use of Signals and Variables

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

Outline

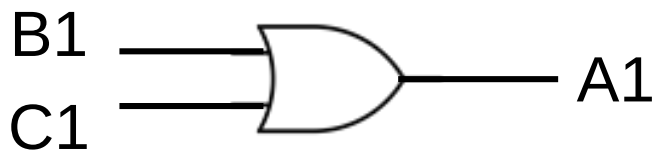
- Revisit: Signal (<=) and Variable (:=) Assignments

- Use of Signals and Variables
  - Outside Process: Concurrent Statement
  - Inside Process: Sequential Statement
    - Combinational Process
    - Sequential Process

Revisit: Signal Assignment (<=)

- **Signal Assignment (<=)**
  - Global to the **`entity`**
  - Concurrent execution
  - Do not be confused with the operator **<=** (equal or smaller)

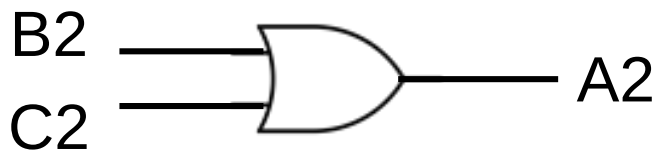
- For example: **`A1 <= B1 or C1`**
  - A1 must be declared outside a **`process`**.
  - A1 represents an internal wire or an input/output pin in port.



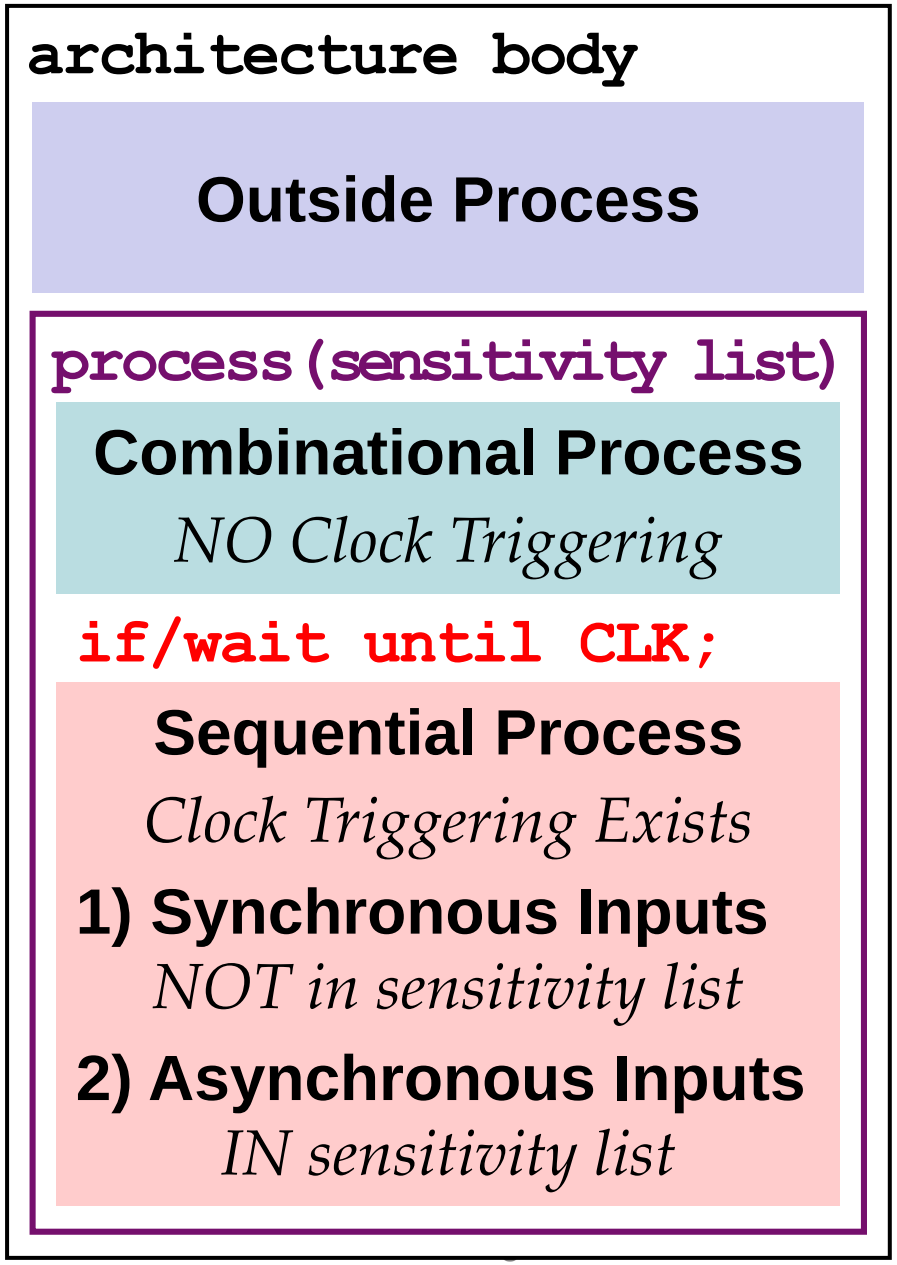
Revisit: Variable Assignment (:=)

- **Variable Assignment (:=)**
  - Local to a **`process`**
  - Sequential execution
  - Constant/signal/variable initialization also uses "`:=`"

- For example: **`A2 := B2 or C2`**
  - A2 must be declared inside a **`process`**.
  - A2 must be a variable.

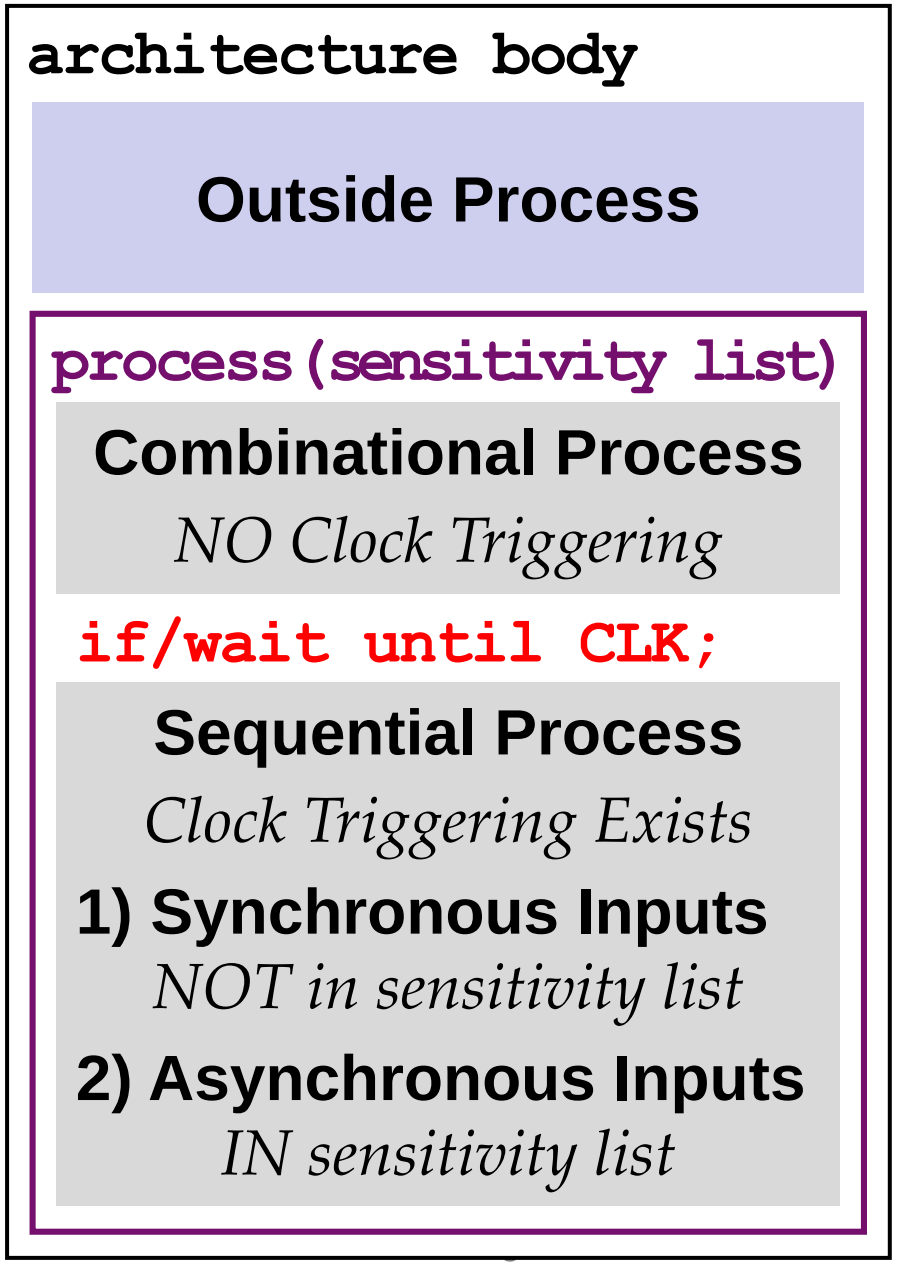


Overview: Use of Signals and Variables



- **Outside Process**
  - Concurrent Statements
- **Inside Process**
  - Sequential Statements
  1) **Combinational Process**: NO `CLK` triggering
     - "<=" is a combinational logic
     - All involved inputs should be in the sensitivity list
  2) **Sequential Process**: Has `CLK` triggering
     - "<=" is a flip-flop
     - **Synchronous Inputs**: should NOT be in the sensitivity list
     - **Asynchronous Inputs**: should be in sensitivity list

Overview: Use of Signals and Variables



- **Outside Process**
  - Concurrent Statements
- **Inside Process**
  - Sequential Statements
  1) **Combinational Process**: NO `CLK` triggering
     - "<=" is a combinational logic
     - All involved inputs should be in the sensitivity list
  2) **Sequential Process**: Has `CLK` triggering
     - "<=" is a flip-flop
     - **Synchronous Inputs**: should NOT be in the sensitivity list
     - **Asynchronous Inputs**: should be in sensitivity list

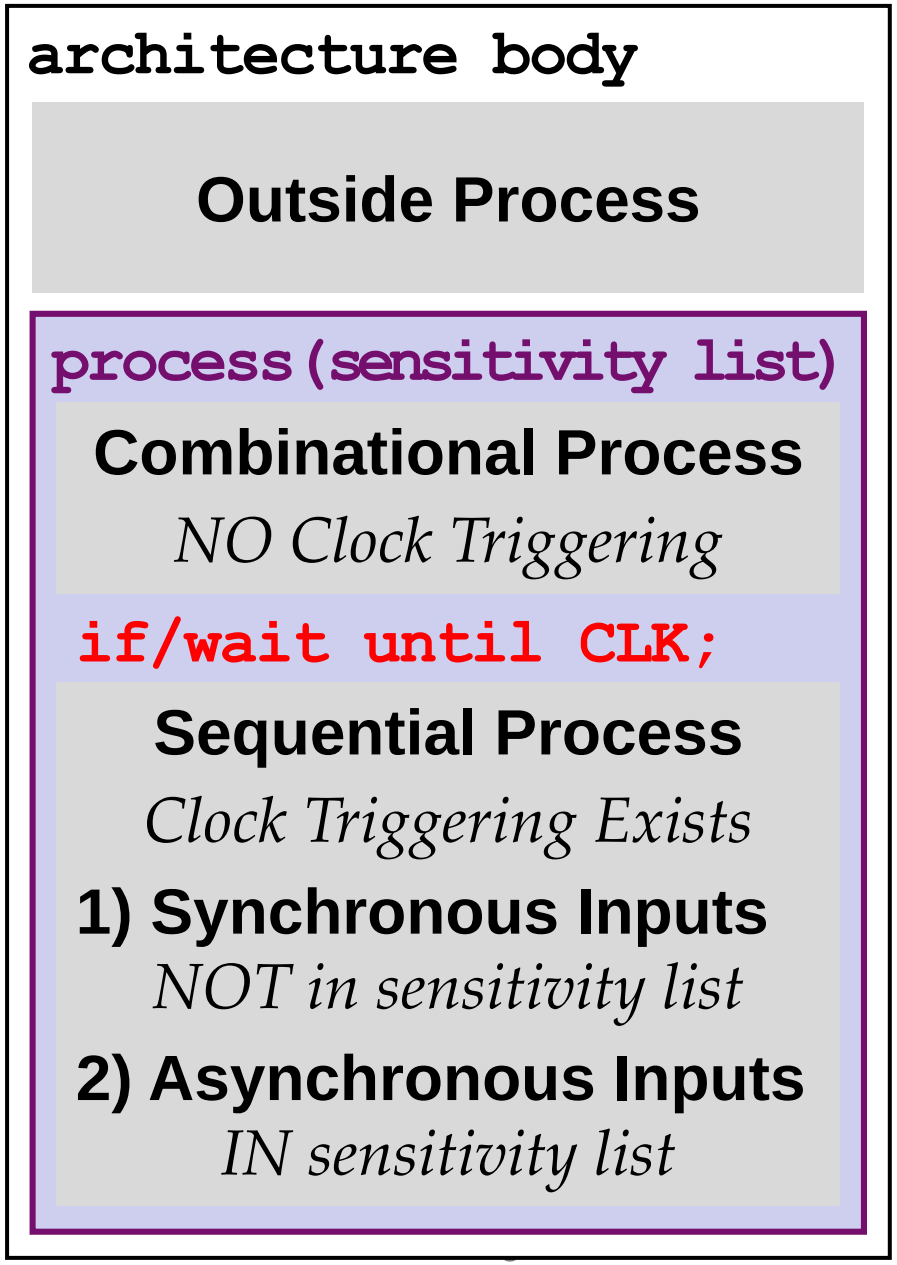
Outside Process: Concurrent Statement

- **Signal Assignments outside a Process**
  - All the statements outside processes are "concurrent".
    - All concurrent statements can be interchanged freely.
    - Each statement will be executed once when any signal in it changes.
  - Signals can be assigned with multiple values if "resolved logic" (i.e., `std_logic` rather than `std_ulogic`) is allowed.

Ex:
```
architecture test_arch of test is
    out1 <= in1 and in2; -- concurrent statement
    out2 <= in1 or in2;  -- concurrent statement
    out2 <= in2;         -- multi-value assignment
end test_arch;
```

- **Variable Assignments outside a Process**
  - Variables can only live *inside* processes!

Overview: Use of Signals and Variables



- **Outside Process**
  - Concurrent Statements
- **Inside Process**
  - Sequential Statements
  1) **Combinational Process**: NO `CLK` triggering
     - "<=" is a combinational logic
     - All involved inputs should be in the sensitivity list
  2) **Sequential Process**: Has `CLK` triggering
     - "<=" is a flip-flop
     - **Synchronous Inputs**: should NOT be in the sensitivity list
     - **Asynchronous Inputs**: should be in sensitivity list

Inside Process: Sequential Statement

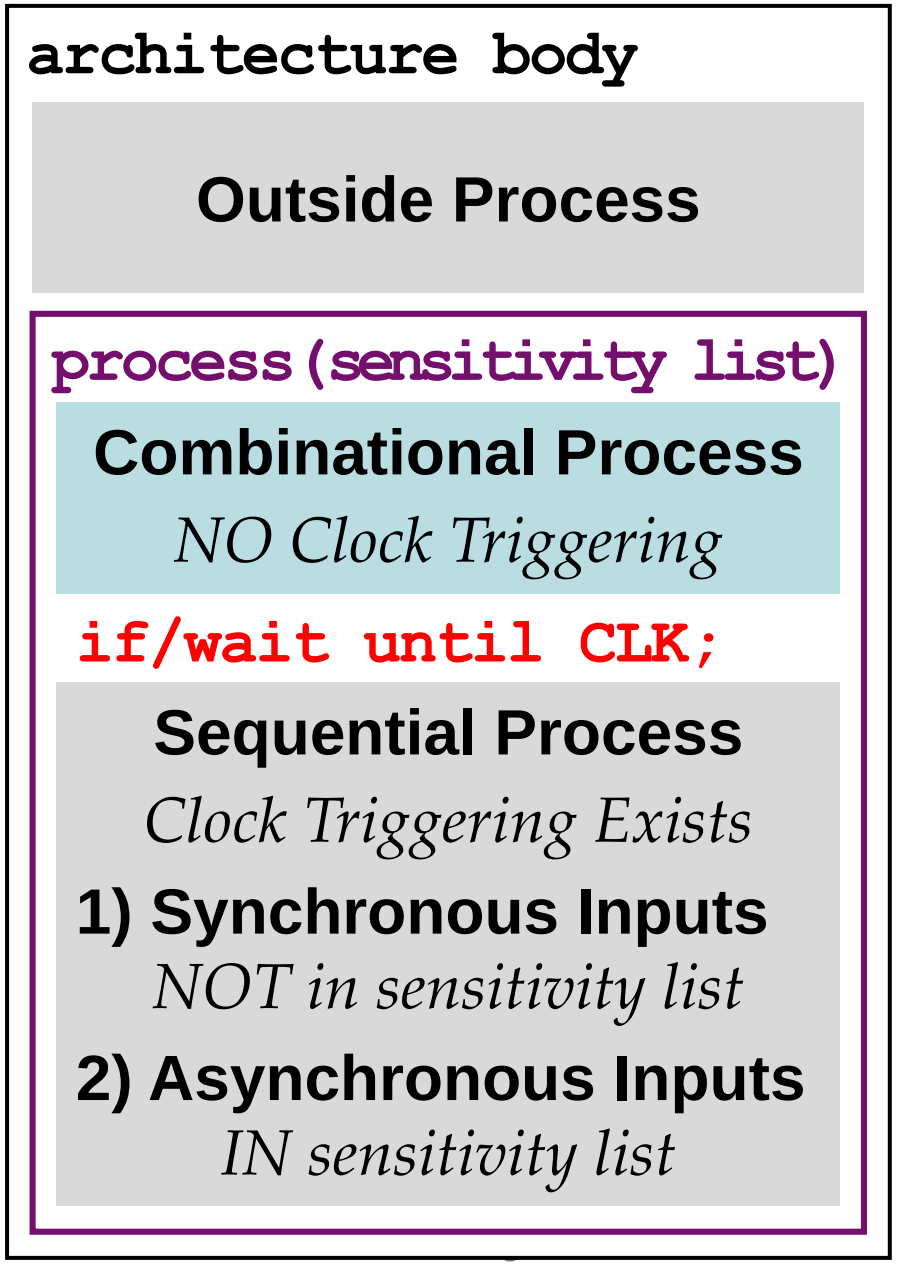
- Statements inside **`process`** are executed sequentially.
  - The process will be executed once when one or more signals in the sensitivity list changes.

Ex:
```
process(in1, in2) -- sensitivity list
variable v1, v2: std_logic;
begin
   s1 <= in1 and in2;
   s1 <= in1 or in2;
   v1 := in1 and in2;
   v1 := in1 or in2;
end process
```

  - **Signals Assignments (<=) inside a Process:**
    Only *the last* assignment for a particular signal takes effect.
  - **Variables Assignments (:=) inside a Process:**
    All assignments take effect immediately and sequentially.

- A process can be: "combinational" or "sequential".

Overview: Use of Signals and Variables



- **Outside Process**
  - Concurrent Statements
- **Inside Process**
  - Sequential Statements
  1) **Combinational Process**: NO `CLK` triggering
     - "<=" is a combinational logic
     - All involved inputs should be in the sensitivity list
  2) **Sequential Process**: Has `CLK` triggering
     - "<=" is a flip-flop
     - **Synchronous Inputs**: should NOT be in the sensitivity list
     - **Asynchronous Inputs**: should be in sensitivity list

1) Combinational Process

- **Combinational Process**
  - *NO* clock triggering condition can be found inside.
    - **Clock Triggering Condition**: `if (clk='1' and clk'event)`, `(wait until clk='1')`, etc.
  - Each "<=" is a combinational logic.
  - *All* involved inputs should be in the sensitivity list.
    - Otherwise the results will be unpredictable.

Ex:
```
combinational_process: process(in1, in2)
  begin
    out3 <= in1 xor in2;
    out3 <= '1';
  end process;
```

Class Exercise 8.1

Student ID: _____ Date:
Name: _____ _____

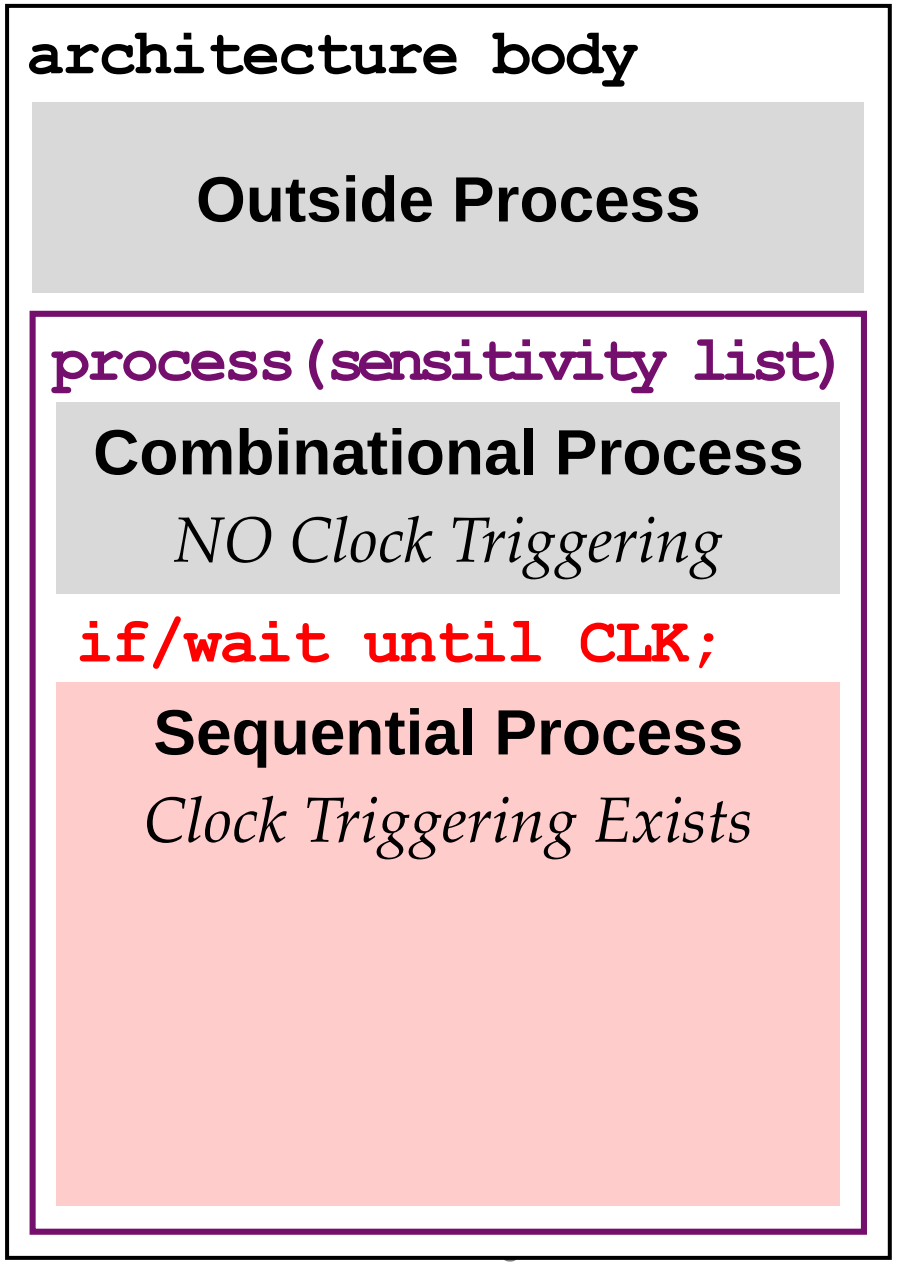
```
 1 signal S1, S2: bit;
 2 signal S_OUT: bit_vector(1 to 8);
 3 process (S1, S2)
 4 variable V1, V2: bit;
 5 begin
 6   V1 := '1';
 7   V2 := '1';
 8   S1 <= '1';
 9   S2 <= '1';
10   S_OUT(1) <= V1;
11   S_OUT(2) <= V2;
12   S_OUT(3) <= S1;
13   S_OUT(4) <= S2;
14   V1 := '0';
15   V2 := '0';
16   S2 <= '0';
17   S_OUT(5) <= V1;
18   S_OUT(6) <= V2;
19   S_OUT(7) <= S1;
20   S_OUT(8) <= S2;
21 end process;
```

- Which line(s) will NOT take effect?
  Answer: _____
- When will the process be executed?
  Answer: _____

  _____
- What are the values of **`S_OUT`** after execution?
  Answer:

```
S_OUT(1):      S_OUT(5):
S_OUT(2):      S_OUT(6):
S_OUT(3):      S_OUT(7):
S_OUT(4):      S_OUT(8):
```

Overview: Use of Signals and Variables



- **Outside Process**
  - Concurrent Statements
- **Inside Process**
  - Sequential Statements
  1) **Combinational Process**: NO `CLK` triggering
     - "<=" is a combinational logic
     - All involved inputs should be in the sensitivity list
  2) **Sequential Process**: Has `CLK` triggering
     - "<=" is a flip-flop
     - **Synchronous Inputs**: should NOT be in the sensitivity list
     - **Asynchronous Inputs**: should be in sensitivity list

2) Sequential Process

- **Sequential Process (a.k.a. Clocked Process)**
  - A clock edge expression can be found inside:
    - "**`if`**" statement:

```
clocked_process: process(sensitivity list)
begin
   … -- same as combinational process
   if (clk='1' and clk'event) then
     out1 <= in1 and in2;
   end if;
   … -- same as combinational process
end process;
```

1) Each "**<=**" is a flip-flop.

2) The assignment takes effect on next clock edge.

    - "**`wait until`**" statement:

```
clocked_process: process -- no sensitivity list
begin
   wait until clk='1';
   out1 <= in1 and in2;
end process;
```

1) Each "**<=**" is a flip-flop.

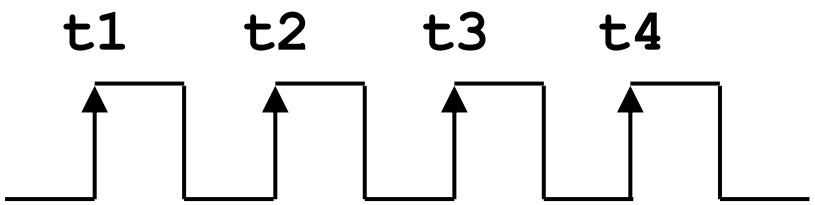
2) The assignment takes effect on next clock edge.

Class Exercise 8.2

Student ID: _____ Date:
Name: _____ _____

- Find the signal results after clock edges **t1 ~ t4**:

```
process
signal s1: integer:=1;
signal s2: integer:=2;
signal s3: integer:=3;
begin
wait until rising_edge(clk);
   s1 <= s2 + s3;
   s2 <= s1;
   s3 <= s2;
   sum <= s1 + s2 + s3;
end process
end
```



	t1	t2	t3	t4
s1				
s2				
s3				
sum				

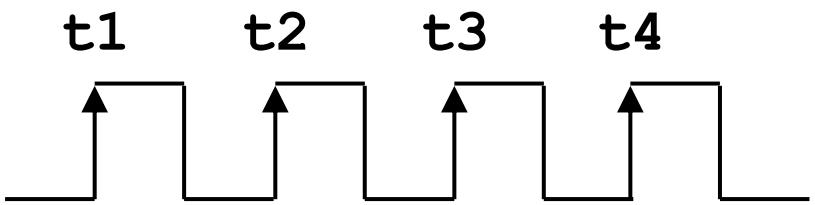
Signals Assignments (<=) inside a Process:
Only *the last* assignment for a particular signal takes effect.
Variables Assignments (:=) inside a Process:
All assignments take effect immediately and sequentially.

Class Exercise 8.3

Student ID: _____ Date:
Name: _____ _____

- Find the signal results after clock edges **t1 ~ t4**:

```
process
variable v1: integer:=1;
variable v2: integer:=2;
variable v3: integer:=3;
begin
wait until rising_edge(clk);
   v1 := v2 + v3;
   v2 := v1;
   v3 := v2;
   sum <= v1 + v2 + v3;
end process
end
```



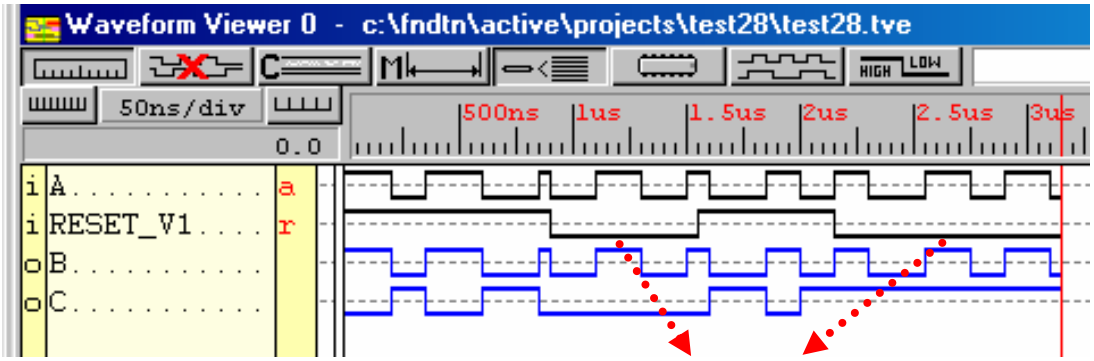
	t1	t2	t3	t4
v1				
v2				
v3				
sum				

Signals Assignments (<=) inside a Process:
Only *the last* assignment for a particular signal takes effect.
Variables Assignments (:=) inside a Process:
All assignments take effect immediately and sequentially.

Do Variables Have Memory?

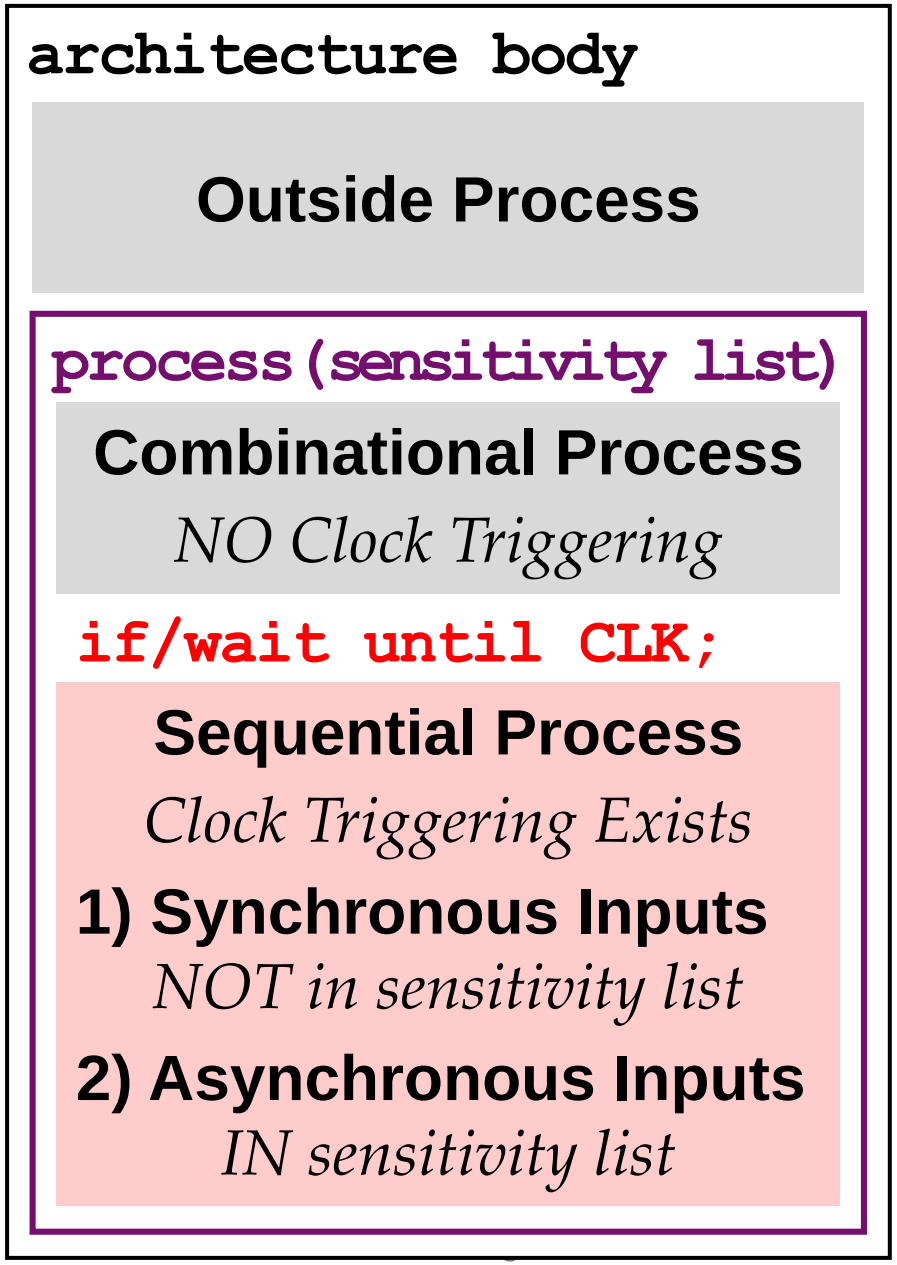
- **Yes.** After a process is called, the state of a variable will be kept for being used again next time.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity test is port (a, reset_v1: in std_logic;
                     b, c: out std_logic); end test;
architecture test_arch of test is
begin
label_proc1: process (a, reset_v1)
variable v1 : std_logic;
begin
  if reset_v1 ='1' then
    v1:= not a;
  end if;
  b <= a;
  c <= v1;
end process label_proc1;
end test_arch;
```



v1 stays at two different levels depending on previous result.

Overview: Use of Signals and Variables

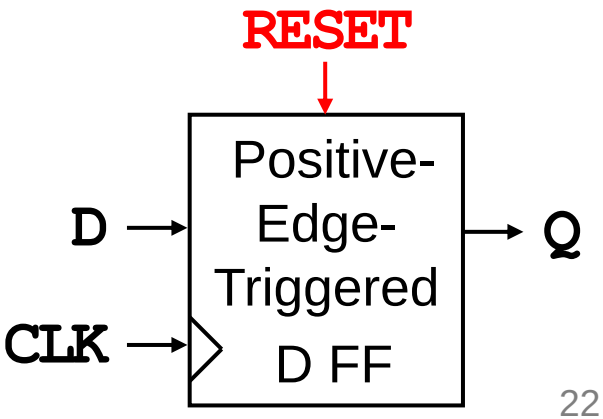


- **Outside Process**
  - Concurrent Statements
- **Inside Process**
  - Sequential Statements
  1) **Combinational Process**: NO `CLK` triggering
     - "<=" is a combinational logic
     - All involved inputs should be in the sensitivity list
  2) **Sequential Process**: Has `CLK` triggering
     - "<=" is a flip-flop
     - **Synchronous Inputs**: should NOT be in the sensitivity list
     - **Asynchronous Inputs**: should be in sensitivity list

Synchronous & Asynchronous Inputs

- Besides of the clock signal (**CLK**), other signals in a **clocked process** can be classified into two types:
  1) **Synchronous Inputs** (e.g., **D** input of flip-flops)
     - Inputs that should be checked only at the next clock edge.
     - *NO* need to put synchronous input signals in the sensitivity list.
  2) **Asynchronous Inputs** (e.g., **RESET** input of flip-flops)
     - Inputs that should be checked either at the next clock edge or when any asynchronous input in the sensitivity list changes.
     - Asynchronous inputs *NEVER* exist in **wait-until** clocked processes.

```
process(CLK, RESET) -- no need to put D, why?
begin
  if (RESET = '1') then
    Q <= '0'; -- Reset Q immediately
  elsif CLK = '1' and CLK'event then
    Q <= D;   -- Q follows input D
  end if;
end process;
```



Class Exercise 8.4

Student ID: _____ Date:
Name: _____ _____

- What are processes p1 and p2 (combinational or sequential)?
- Which signals are sync., async., or combinational inputs?

```
…
port(clock,reset: in std_logic;
         t_light: out std_logic_vector (2 downto 0));
…
type traffic_state_type is (s0, s1,s2,s3);
signal t_state: traffic_state_type; -- internal signal
p1: process(t_state)
begin
  case (t_state) is
    when s0 => t_light <= "100";
    when s1 => t_light <= "110";
    when s2 => t_light <= "001";
    when s3 => t_light <= "010";
  end case;
end process;
```

```
p2: process
begin
  wait until clock='1';
  if reset = '1' then
     t_state <= s0;
  else
    case t_state is
      when s0 => t_state <= s1;
      when s1 => t_state <= s2;
      when s2 => t_state <= s3;
      when s3 => t_state <= s0;
    end case;
  end if;
end process;
```

Class Exercise 8.5

Student ID: _____ Date:
Name: _____ _____

- Based on Class Exercise 7.4, rewrite process p2 using asynchronous reset.

```
sync_p2: process
begin
  wait until clock='1';
  if reset = '1' then
     t_state <= s0;
  else
    case t_state is
      when s0 => t_state <= s1;
      when s1 => t_state <= s2;
      when s2 => t_state <= s3;
      when s3 => t_state <= s0;
    end case;
  end if;
end process;
```

```
async_p2: process
begin




end process;
```

Recall: "wait until" vs. "if"

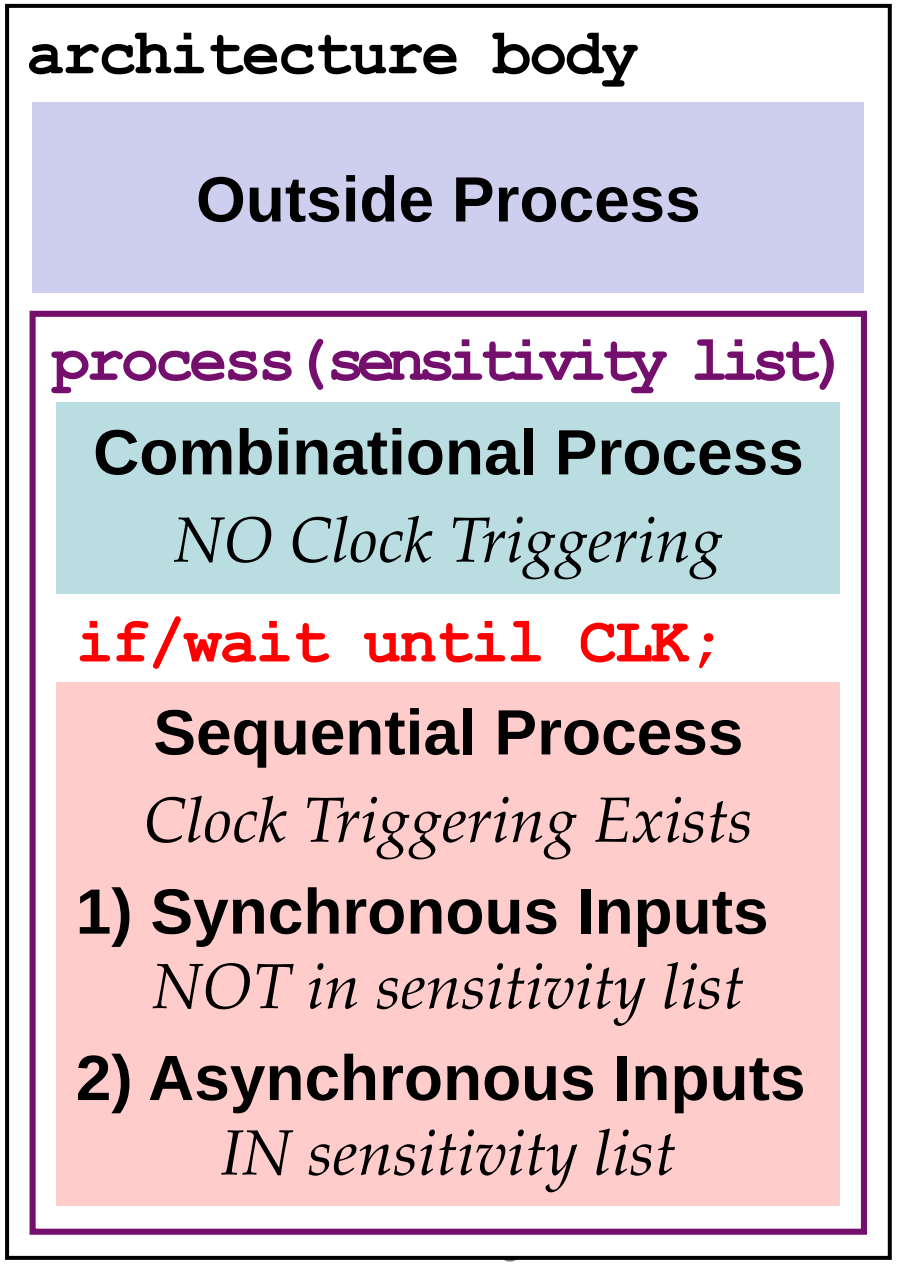
- **Asynchronous Process**: Computes values on clock edges or when asynchronous conditions are TRUE.
  - That is, it must be sensitive to the clock signal (if any), and to all inputs that may affect the asynchronous behavior.
  - **Rule:** Only use "**if**" for **asynchronous** process:

Usage of "if"

```
process (clk, input_a, input_b, …)
begin
   …
   if( rising_edge(clk) )
   …
end process
```

← The sensitivity list should include the clock signal, and all inputs that may affect asynchronous behavior.

Summary: Use of Signals and Variables



- **Outside Process**
  - Concurrent Statements
- **Inside Process**
  - Sequential Statements
  1) **Combinational Process**: NO `CLK` triggering
     - "<=" is a combinational logic
     - All involved inputs should be in the sensitivity list
  2) **Sequential Process**: Has `CLK` triggering
     - "<=" is a flip-flop
     - **Synchronous Inputs**: should NOT be in the sensitivity list
     - **Asynchronous Inputs**: should be in sensitivity list

Summary: Inside Process

- **Signals Assignments (<=) inside a Process**
  - Only the ***last*** assignment for a particular signal takes effect.
  - **Combinational Process**: ***No*** clock (**`CLK`**) triggering
    - Each "<=" is a combinational logic.
    - All involved inputs should be in the sensitivity list.
  - **Sequential Process**: Has clock (**`CLK`**) triggering
    - Signal assignments ***before*** or ***outside*** the clock edge detection:
      - As the same as combinational process (be careful!).
    - Signal assignments ***after*** or ***inside*** the clock edge detection:
      - Each "<=" can be treated as a flip-flop: The signal assignment will take effect at the next clock edge.
      - **Synchronous inputs** should ***NOT*** be in the sensitivity list.
      - **Asynchronous inputs** should be in the sensitivity list.
- **Variables Assignments (:=) inside a Process**
  - ***All*** assignments take effect immediately and sequentially.

Summary: Multiple Assignments

- **Signals**
  - **Outside Process**
    - Signals can be assigned with multiple values (i.e., "multi-value" or "multi-driven") only if "***resolved logic***" is allowed.
    - If not allowed? **Avoid assigning** a signal from multiple processes (or multiple concurrent statement).
  - **Inside Process**
    - Only ***the last*** assignment for that signal will take effect.
- **Variables**
  - **Outside Process**
    - Variables can only live ***inside*** processes!

  - **Inside Process**
    - ***ALL*** assignments take effect immediately and sequentially.