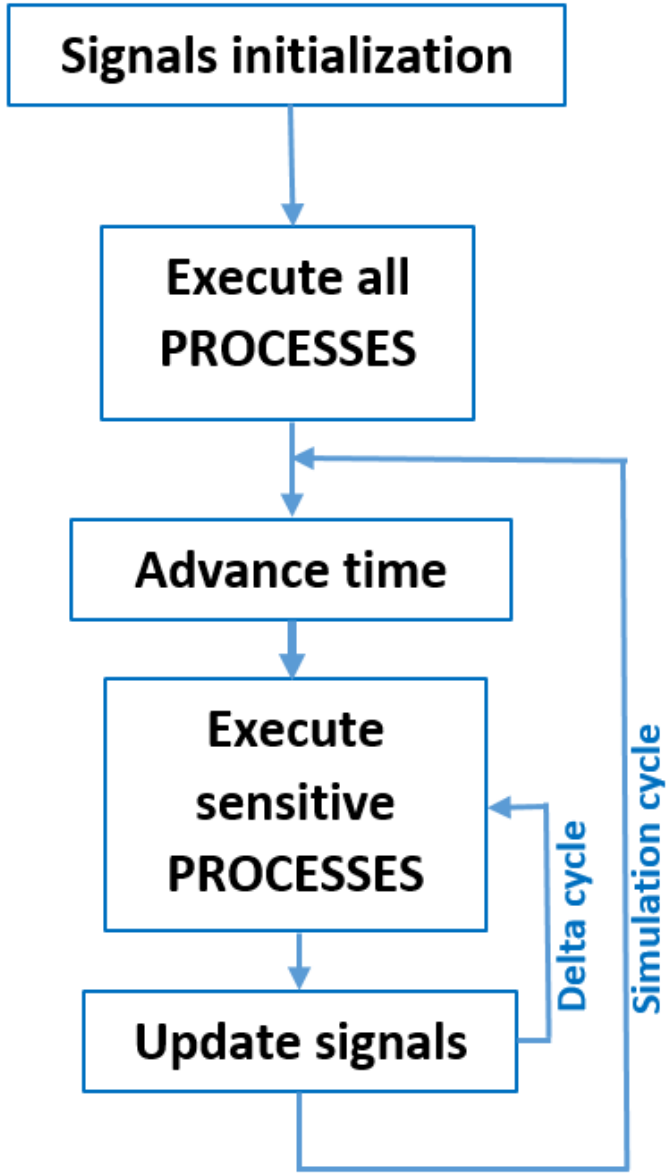
VHDL - Sequential Code (Simulation **vs** Synthesis)

Simulation and Delta cycles



- **Phase 1:** the compiler reads the whole code for signals initialization.
- **Phase 2:** the compiler waits for at least a SIGNAL event from PROCESSES sensitive list **or** from concurrent statements (called implied PROCESS).
- **Phase 3:** when SIGNAL event has happened, the compiler makes for each stimulated PROCESS a sequential list of buffers for the consecutive SIGNALS assignments. At line END PROCESS all those assignments are performed in parallel (causes inner loops – Delta cycles).
- **Phase 4:** advance time (causes outer loops – Simulation cycles)

Simulation and Delta cycles

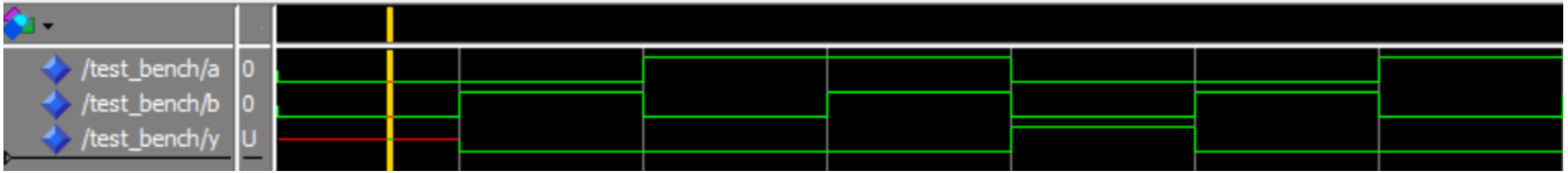
```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
-----------------------------------
entity case3 is port(
    a,b : in std_logic;
    y   : out std_logic);
end case3;
-----------------------------------
architecture arc of case3 is
  signal c : std_logic;
begin
  process (a, b)
    begin
      c <= a and b ;
      y <= c ;
  end process ;
end architecture arc;
```

Don't use **SIGNALS** for Intermediate calculations

List - Default

ps / delta		/test_bench/a	/test_bench/b	/test_bench/y
0	+1	0	0	U
50000	+2	0	1	0
100000	+1	1	0	0
150000	+1	1	1	0
200000	+2	0	0	1
250000	+2	0	1	0
300000	+1	1	0	0
350000	+1	1	1	0
400000	+3	0	0	1
450000	+2	0	1	0
500000	+1	1	0	0
550000	+2	1	1	0
600000	+2	0	0	1
650000	+2	0	1	0
700000	+1	1	0	0
750000	+2	1	1	0
800000	+2	0	0	1

case3 code simulation doesn't describe **AND** gate



Simulation and Delta cycles

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
-----------------------------------
entity case1 is port(
    a,b : in std_logic;
    y   : out std_logic);
end entity;
-----------------------------------
architecture rtl of case1 is
  signal c : std_logic;
begin
  c <= a and b ; -- implied process
  y <= c ;       -- implied process
end architecture rtl;
```

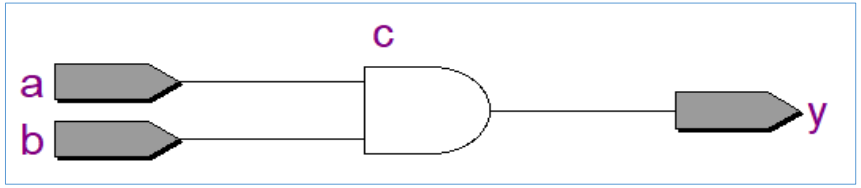
Case1, Case2, Case4 codes simulation describe **AND** gate

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
-----------------------------------
entity case2 is port(
    a,b : in std_logic;
    y   : out std_logic);
end entity;
-----------------------------------
architecture case2 of case2 is
  signal c : std_logic;
begin
  process1 : process (a, b)
    begin
      c <= a and b ;
  end process process1 ;

  process2 : process (c)
    begin
      y <= c ;
  end process process2 ;

end architecture case2;
```

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
-----------------------------------
entity case4 is port(
    a,b : in std_logic;
    y   : out std_logic);
end case4;
-----------------------------------
architecture rtl of case4 is
begin
  process (a, b)
    variable c : std_logic;
  begin
      c := a and b ;
      y <= c ;
  end process ;
end architecture rtl;
```

Simulation **vs** Synthesis - summary

- Case3 code doesn't describe **AND** gate while Case3 code synthesis does describe **AND** gate!



- Case1, Case2, Case4 codes simulation and Synthesis describe **AND** gate.
- Synthesis tools and Simulation tools translate PROCESS based VHDL code in different ways (concurrent code translated in the same way).
- Synthesis tools search for adjustment of HDL code to one of the next three template kinds (*ieee-1076.6 standard*):

  Combinational Logic, Synchronous Logic, Latch based Logic.

- **Our goal: writing of HDL code which will be translated in the same exact way by all Synthesis and Simulation tools.**