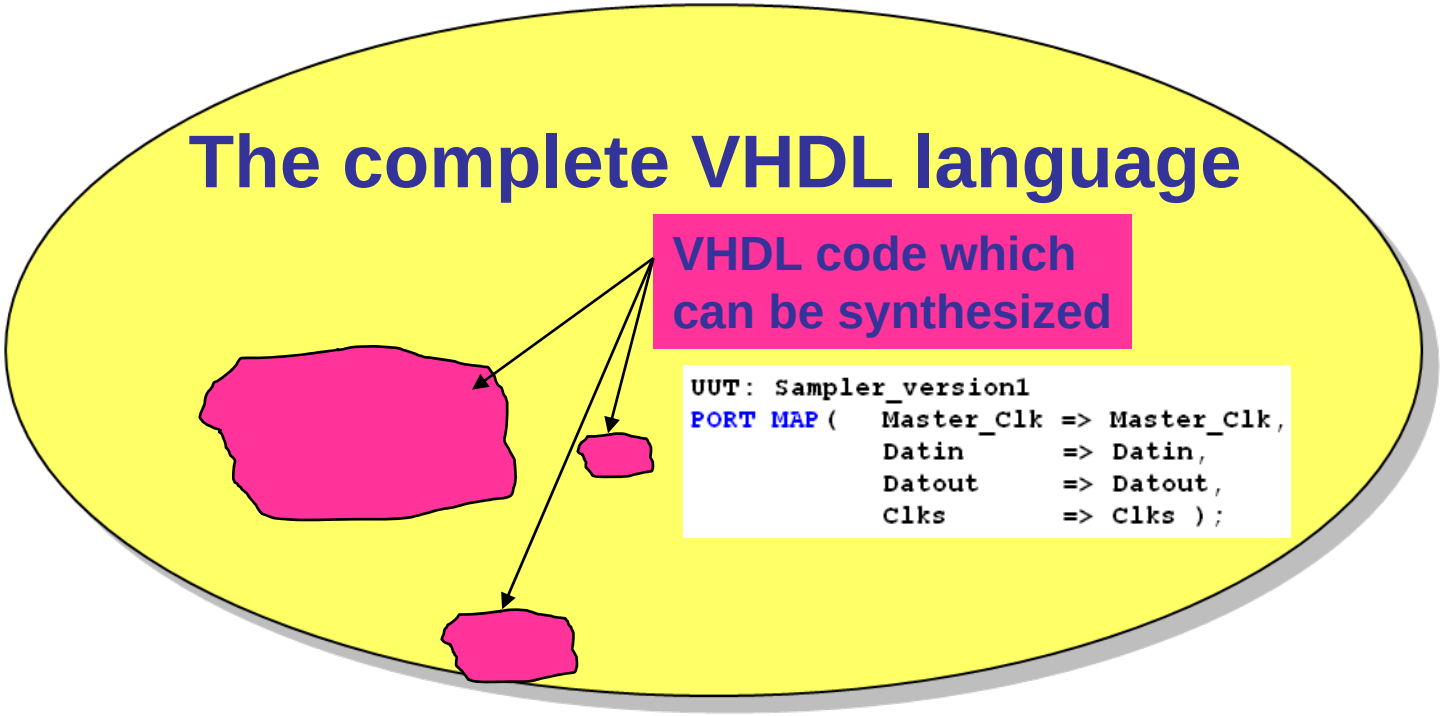
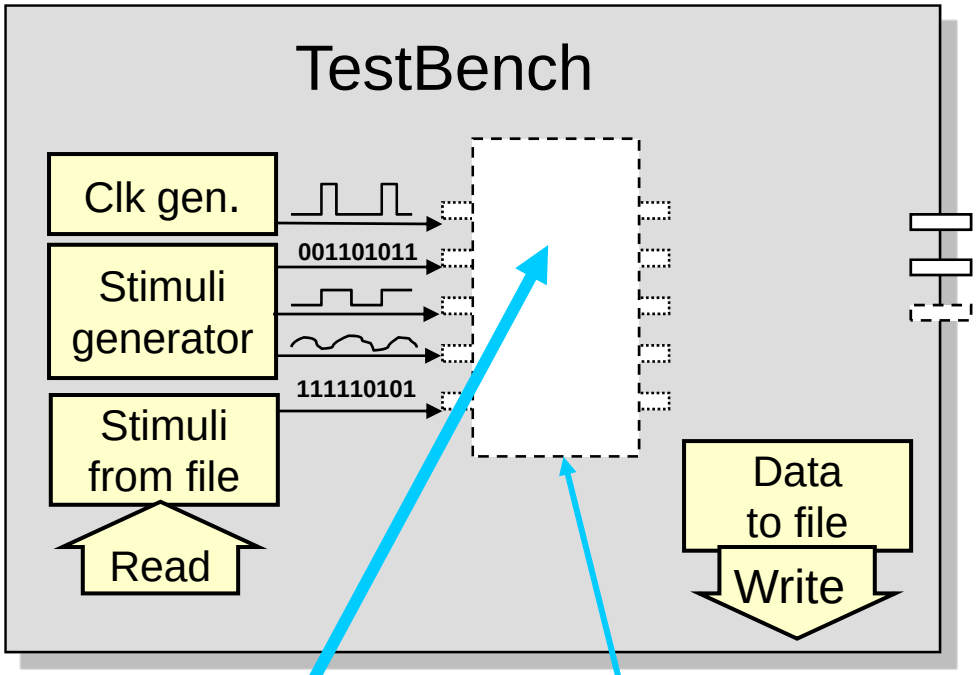
VHDL for simulation



The original purposes for VHDL was to write code which:

- **Described existing hardware – System Models**
- **Generated stimuli for a UUT (Unit Under Test)**
- **Evaluated the output and gave errors/warnings**

VHDL for simulation - TestBenches



Normally will the ENTITY of a TestBench be empty

```
ENTITY Test_of_Sampler1_selfcheck_beh IS
END Test_of_Sampler1_selfcheck_beh;
```

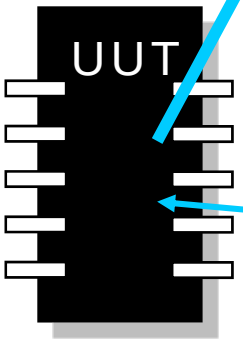
But your allowed to create OUT and INOUT signal for the presentation of Data

```
ENTITY Min_TestBench_Sampler1_vhd IS
    Port ( CLKX:      out  STD_LOGIC;
           Dataout :  inout  STD_LOGIC_VECTOR (7 downto 0));
END Min_TestBench_Sampler1_vhd;
```

```
UUT: Sampler_version1
PORT MAP (   Master_Clk => Master_Clk,
             Datin      => Datin,
             Datout     => Datout,
             Clks       => Clks );
```

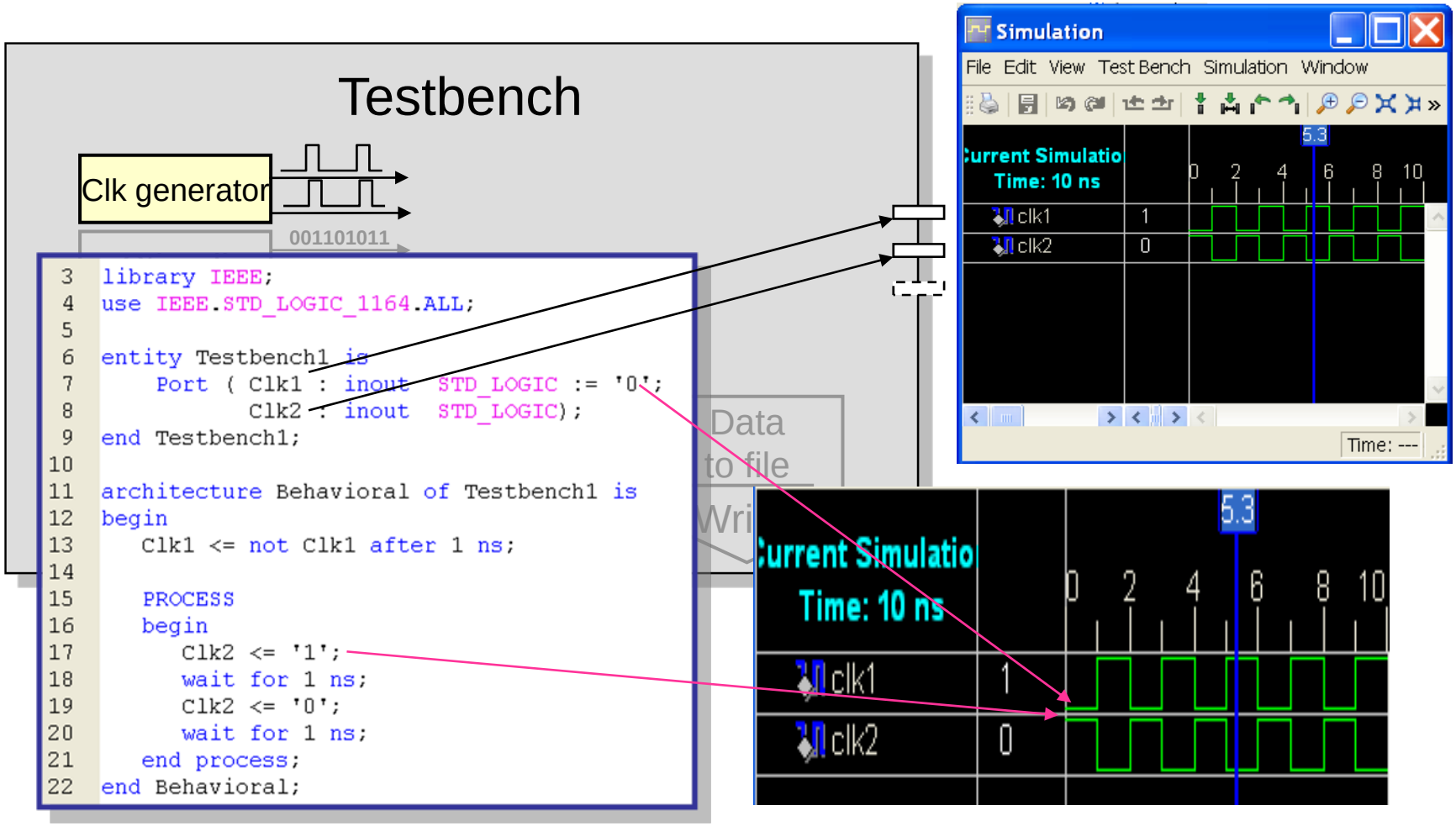
The Unit Under Test will normally be some VHDL code for Synthesize

The TestBench provide stimuli for UUT.




```
COMPONENT Sampler_version1
PORT(    Master_Clk : IN  std_logic;
         Datin :      IN  std_logic_vector(7 downto 0);
         Datout :     OUT std_logic_vector(7 downto 0);
         Clks :       OUT std_logic );
END COMPONENT;
```

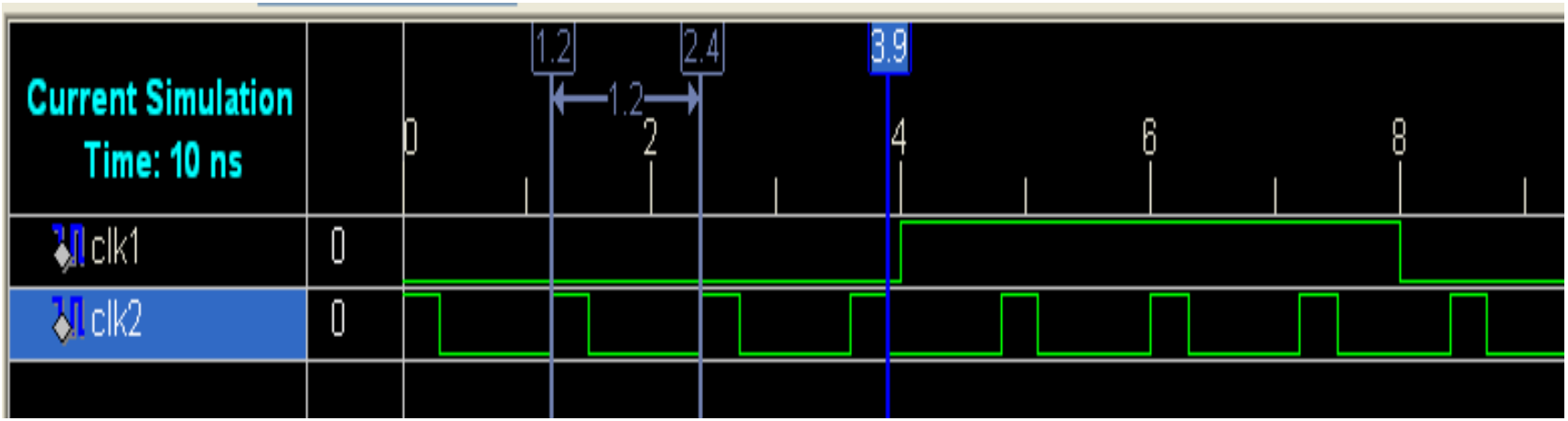
The Clock generators – Example of Stimuli (1)



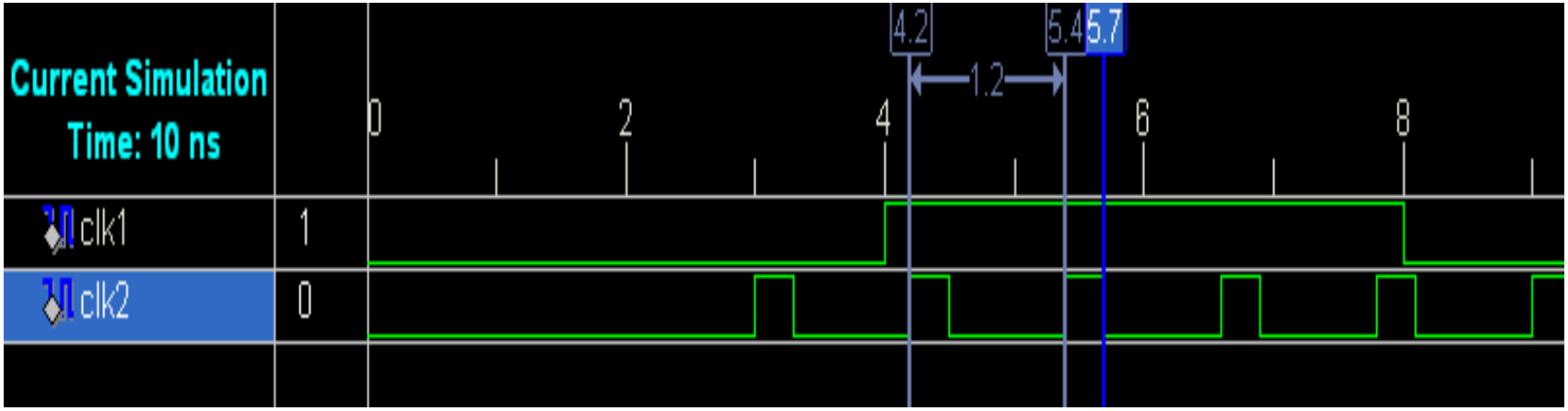

```
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5
6  entity Testbench1 is
7      Port ( Clk1 : inout  STD_LOGIC := '0';
8             Clk2 : inout  STD_LOGIC);
9  end Testbench1;
10
11 architecture Behavioral of Testbench1 is
12 begin
13    Clk1 <= not Clk1 after 1 ns;
14
15    PROCESS
16    begin
17       Clk2 <= '1';
18       wait for 1 ns;
19       Clk2 <= '0';
20       wait for 1 ns;
21    end process;
22 end Behavioral;
```

The VHDL statements **after** and **wait for** are only ment for simulation

Exercises – Asymmetric clock-signal



Create an Asymmetric clock-signal



Create an Asymmetric clock-signal – with a start offset (3 ns)

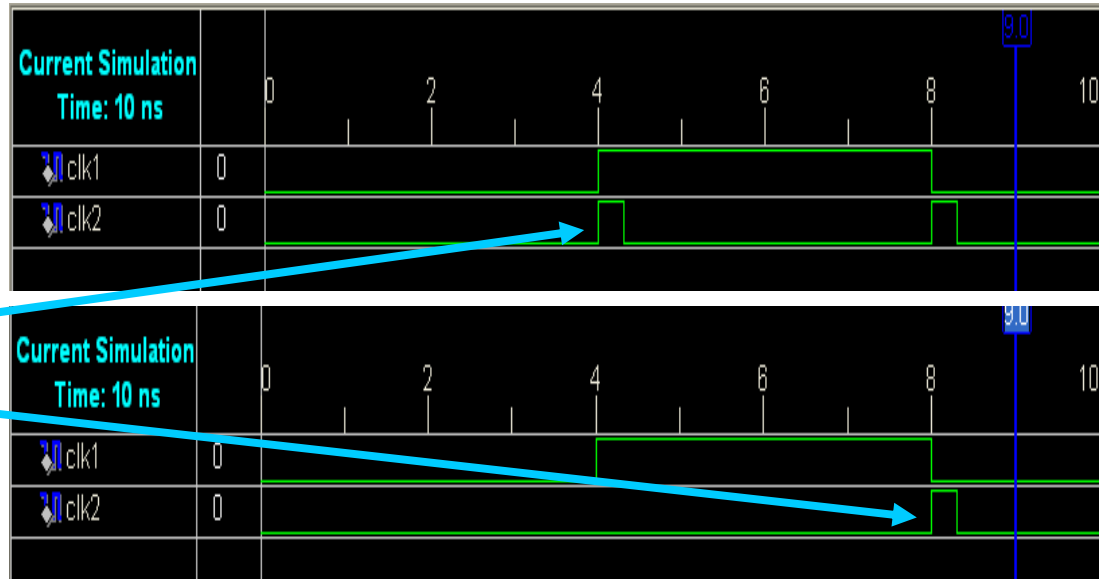
The Clock generators – Example of Stimuli (2)

```
7   entity Testbench2_SigGen is
8       Port ( Clk1 : inout  STD_LOGIC := '0';
9              Clk2 : inout  STD_LOGIC := '0');
10  end Testbench2_SigGen;
11
12  architecture Behavioral of Testbench2_SigGen is
13
14  begin
15     clk1 <= not clk1 after 4 ns;
16
17     process
18     begin
19        wait for 3 ns; -- Start offset
20        clock_loop:
21        LOOP
22  --       wait on clk1;
23  --       wait until clk1='0';
24  --       wait on clk1 for 1500 ps;
25           clk2 <= '1';
26           wait for 300 ps;
27           clk2 <= '0';
28           wait for 900 ps;
29        end LOOP clock_loop;
30     end process;
31  end Behavioral;
```

The **Wait** statement can take three forms:

Wait for *time* **ns**;
Wait until *condition true;*
Wait on *change of signal;*

Finally can you mix those forms if needed.
The **Wait** statement only for processes



Note – A **Process** ”born” with an infinite loop, but your allowed to make your own with **LOOP** .. **End LOOP**

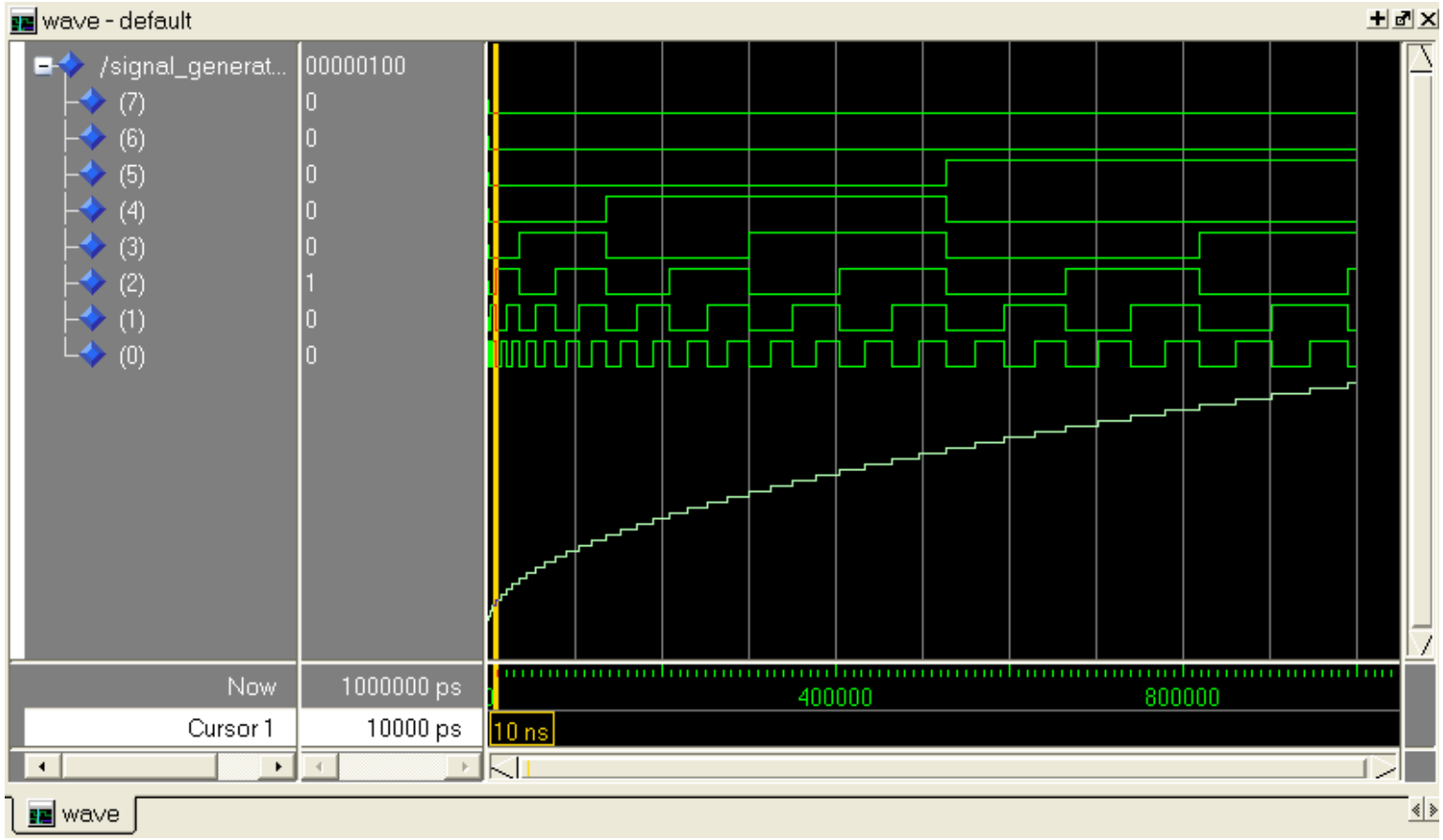
”Slope” generator – Example of Stimuli (1)

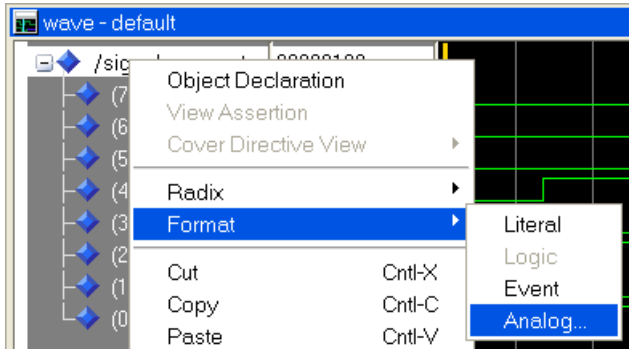
```
8   entity Signal_generator1 is
9       Port ( Data : out  STD_LOGIC_VECTOR (7 downto 0));
10  end Signal_generator1;
11
12  architecture Behavioral of Signal_generator1 is
13     -- Shared variables also called global variables as they can be reached
14     -- by other processes.
15     -- Try to remove the comments from the process below and simulate again
16     ----------------------------------------------------------------------
17     shared variable n: STD_LOGIC_VECTOR (7 downto 0) := (others=>'0');
18     shared variable i: integer := 1;
19  begin
20     ---------------------- This process produce a "slope" signal ----------
21     process
22        variable delay: integer := 0;
23     begin
24        Data <= n;
25        n := n +1;
26        i := i +1;
27        Delay := i;
28       --- Wait for "Delay" ns --
29        while Delay >0 loop
30           wait for 1 ns;
31           delay := delay-1;
32        end loop;
33     end process;
34
35  ------------------------ Try to remove the comments from this --------
36  -- process
37  -- begin
38  --    wait for 500 ns;
39  --    i := 1;
40  -- end process;
41  end Behavioral;
```

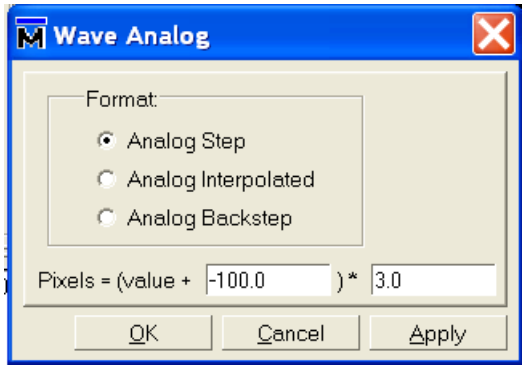
Shared variable can be very usefull if processes need to exchange data during a simulation.

Please note!!
For simulation purposes, are you allowed to set or change a **Shared variable** from more then one **process**.
This not allowed when you are dealing with code for synthesizing.

”Slope” generator – Example of Stimuli (3)







MATH_REAL

Constants and subprograms for real numbers. Is precompiled into the library "IEEE" (accessed via **USE IEEE.MATH_REAL.ALL**).

```
package MATH_REAL is
  constant MATH_E            : REAL := 2.71828_18284_59045_23536;
  constant MATH_1_OVER_E     : REAL := 0.36787_94411_71442_32160;
  constant MATH_PI           : REAL := 3.14159_26535_89793_23846;
  constant MATH_2_PI         : REAL := 6.28318_53071_79586_47693;
  constant MATH_1_OVER_PI    : REAL := 0.31830_98861_83790_67154;
  constant MATH_PI_OVER_2    : REAL := 1.57079_63267_94896_61923;
  constant MATH_PI_OVER_3    : REAL := 1.04719_75511_96597_74615;
  constant MATH_PI_OVER_4    : REAL := 0.78539_81633_97448_30962;
  constant MATH_3_PI_OVER_2  : REAL := 4.71238_89803_84689_85769;
  constant MATH_LOG_OF_2     : REAL := 0.69314_71805_59945_30942;
  constant MATH_LOG_OF_10    : REAL := 2.30258_50929_94045_68402;
  constant MATH_LOG2_OF_E    : REAL := 1.44269_50408_88963_4074;
  constant MATH_LOG10_OF_E   : REAL := 0.43429_44819_03251_82765;
  constant MATH_SQRT_2       : REAL := 1.41421_35623_73095_04880;
  constant MATH_1_OVER_SQRT_2: REAL := 0.70710_67811_86547_52440;
  constant MATH_SQRT_PI      : REAL := 1.77245_38509_05516_02730;
  constant MATH_DEG_TO_RAD   : REAL := 0.01745_32925_19943_29577;
  constant MATH_RAD_TO_DEG   : REAL := 57.29577_95130_82320_87680;
```

MATH_REAL

```
    function SIGN    (X  :REAL) return REAL;
    function CEIL    (X  :REAL) return REAL;
    function FLOOR   (X  :REAL) return REAL;
    function ROUND   (X  :REAL) return REAL;
    function TRUNC   (X  :REAL) return REAL;
    function "MOD"   (X,Y:REAL) return REAL;
    function REALMAX (X,Y:REAL) return REAL;
    function REALMIN (X,Y:REAL) return REAL;

    procedure UNIFORM (variable SEED1,SEED2:inout POSITIVE;
                       variable X:out REAL);
    function SQRT    (X:REAL)            return REAL;
    function CBRT    (X:REAL)            return REAL;
    function "**"    (X:INTEGER; Y:REAL) return REAL;
    function "**"    (X:REAL;    Y:REAL) return REAL;
    function EXP     (X:REAL)            return REAL;
    function LOG     (X:REAL)            return REAL;
    function LOG2    (X:REAL)            return REAL;
    function LOG10   (X:REAL)            return REAL;
    function LOG     (X:REAL; BASE:REAL) return REAL;
    function SIN     (X:REAL)            return REAL;
    function COS     (X:REAL)            return REAL;
    function TAN     (X:REAL)            return REAL;
    function ARCSIN  (X:REAL)            return REAL;
    function ARCCOS  (X:REAL)            return REAL;
    function ARCTAN  (Y:REAL)            return REAL;
    function ARCTAN  (Y:REAL; X:REAL)    return REAL;
    function SINH    (X:REAL)            return REAL;
    function COSH    (X:REAL)            return REAL;
    function TANH    (X:REAL)            return REAL;
    function ARCSINH (X:REAL)            return REAL;
    function ARCCOSH (X:REAL)            return REAL;
    function ARCTANH (X:REAL)            return REAL;
  end package MATH_REAL;
```

”Sinus” generator – Example of Stimuli (4)

```
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5  use IEEE.STD_LOGIC_ARITH.ALL;
6  use IEEE.STD_LOGIC_UNSIGNED.ALL;
7  use IEEE.MATH_REAL.ALL;              -- <<<<< Note the use of MATH_REAL
8
9  entity Signal_generator2 is
10     Port ( Dataout : out  STD_LOGIC_VECTOR (7 downto 0));
11 end Signal_generator2;
12
13 architecture Behavioral of Signal_generator2 is
14 begin
15    Sinus_generator:
16    process
17       constant Umax:      integer := 127;      -- Max amplitude
18       constant f:         real    := 2.0E6;   -- Frequency [Hz]
19       constant Tper:      real    := 1.0/f;    -- Period of fr.
20    -- If you can find a way to convert real to time please let me know
21       constant Delta:     real    := 1000.0E-12; -- delta time - sec
22       constant DeltaWait: time    := 1000 ps;    -- delta time - ps
23    ------------------------------------------------------------------
24       variable t:     Real := 0.0;  -- Actual time
25       variable angle: real := 0.0;  -- Actual angle in radians
26       variable Usin:  real := 0.0;  -- The sin value [real]
27       variable Usin_int: integer;   -- The sin value as integer
28    begin
29       while angle < MATH_2_PI*2.0 loop          -- go for to periodes
30          angle    := 2.0 * MATH_PI * t * f;     -- calculate angle
31          t        := t + Delta;                 -- next time
32          Usin     := real(Umax)*( SIN( angle)+1.0); -- Usin calculation
33          Usin_int := integer(Usin);                 -- convert real to integer
34          Dataout  <= conv_std_logic_vector( Usin_int, 8);  -- to vector
35          wait for DeltaWait;                         -- Wait one delta time
36       end loop;
37       wait;      -- forever / stop the process
38    end process;
39 end Behavioral;
```

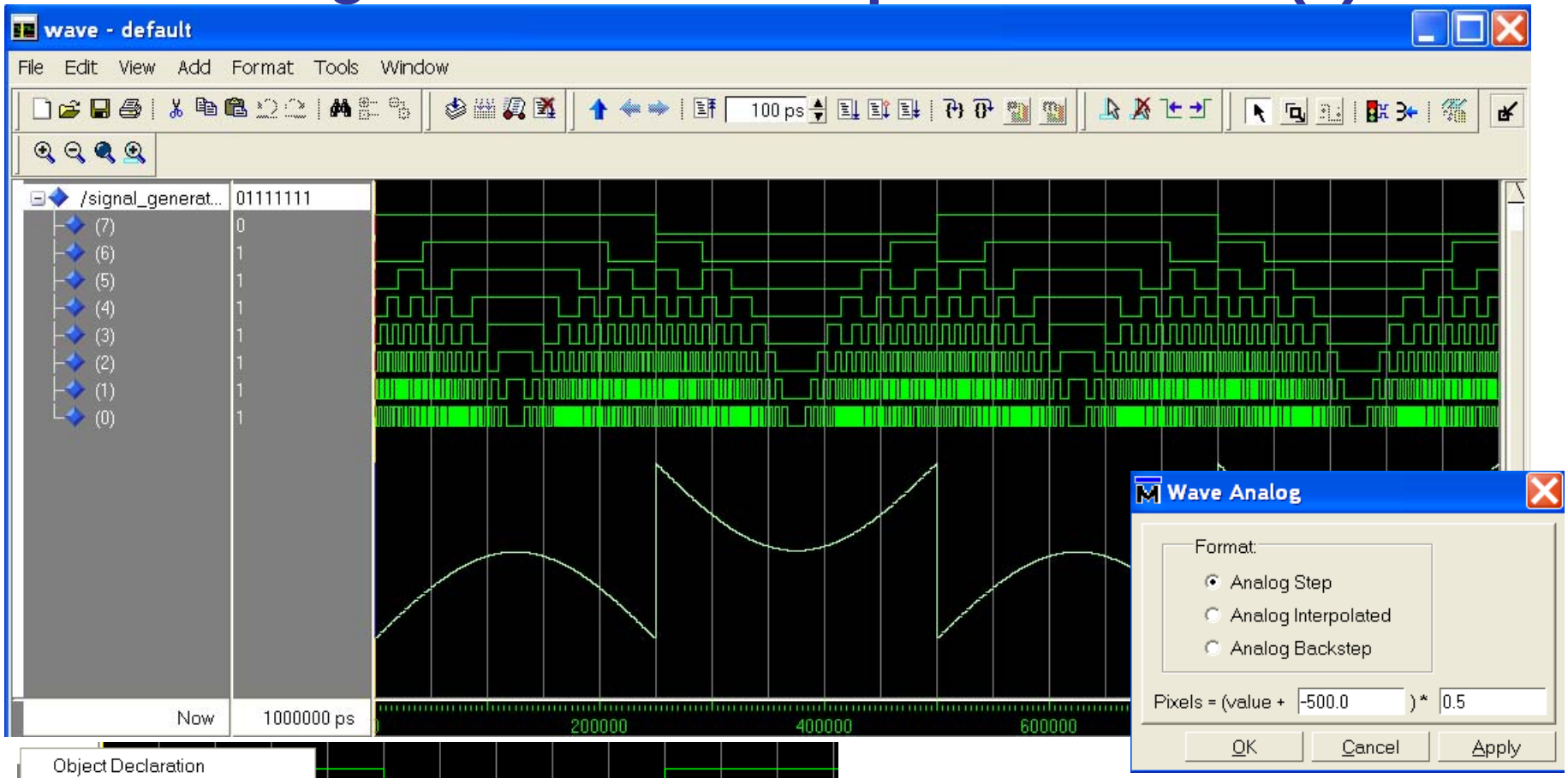
Conversion between real, integer .. vectors etc.

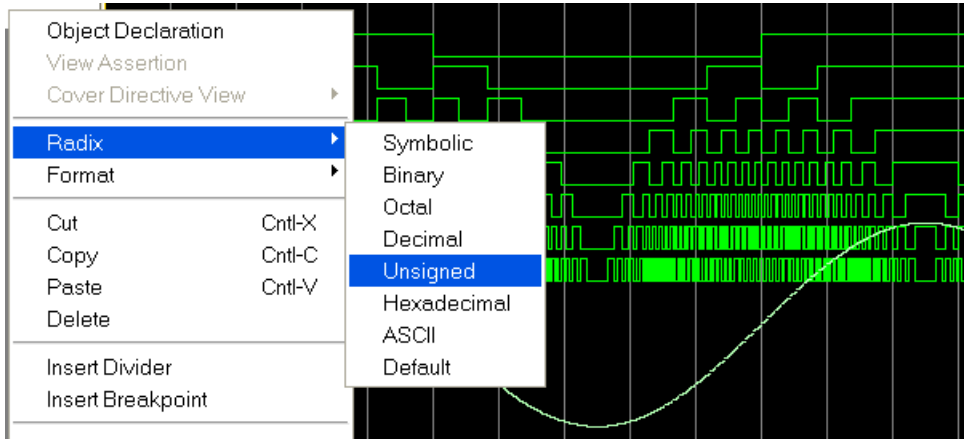
```
Port ( Dataout : out  STD_LOGIC_VECTOR (7 downto 0));

constant Umax:       integer := 127;      -- Max amplitude

variable Usin:  real := 0.0;  -- The sin value [real]
variable Usin_int: integer;   -- The sin value as integer

Usin     := real(Umax)*( SIN( angle)+1.0);
Usin_int := integer(Usin);
Dataout  <= conv_std_logic_vector( Usin_int, 8);

-- If you can find a way to convert real to time please let me know
constant Delta:     real     := 1000.0E-12; -- delta time - sec
constant DeltaWait: time     := 1000 ps;    -- delta time - ps
-------------------------------------------------------------------
```

VHDL is known for its ”hard” use of types – However is it possible to convert between most types, eighter by using build in functions like **real**( ) and **integer**( ).

Other conversion functions can be found in libraries – like for instance: **Conv_Integer**( ) and **Conv_std_logic_vector**(,)

”Sinus” generator – Example of Stimuli (5)





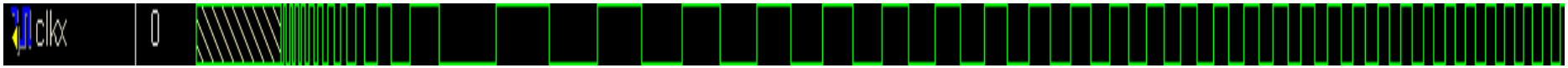
When it comes to watch a vector as an analog value, choose the modelsim simulator instead of the ISE simulator.

”Sinus-Sweep” generator – Example of Stimuli (6)

```
2  ---------------------------------------------------------------------------
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5  use IEEE.STD_LOGIC_ARITH.ALL;
6  use IEEE.STD_LOGIC_UNSIGNED.ALL;
7  use IEEE.MATH_REAL.ALL;              -- This contains the SIN( )
8
9  entity Signal_generator3 is
10     Port ( CLKX:      out  STD_LOGIC;
11            Dataout : out  STD_LOGIC_VECTOR (7 downto 0));
12 end Signal_generator3;
13
14 architecture Behavioral of Signal_generator3 is
15    signal Clk_v1:          STD_LOGIC := '0';
16    -- Please note - Shared variables can be used for interprocess data
17    -- exchange. Moreover can they be observed under a simulation as well
18    shared variable Delta_step: real := 100.0;
19    shared variable Delta:      real := 1000.0;
20    shared variable Delta_xx:   real := -1.0;
21 begin
22    CLKX <= Clk_v1;
23    ---------------------------------------------------------------------
24    -- This process creates a clk-signal with a variable frequency
25    -- not use if its useful in practice, but it demonstrates what can
26    -- be done.
27    -- The shared variable "Delta" will decrease with the value "Delta_step"
28    -- for each step will the "Delta_step" value change with "Delta_xx"
29    ---------------------------------------------------------------------
```

”Sinus-Sweep” generator – Example of Stimuli (7)

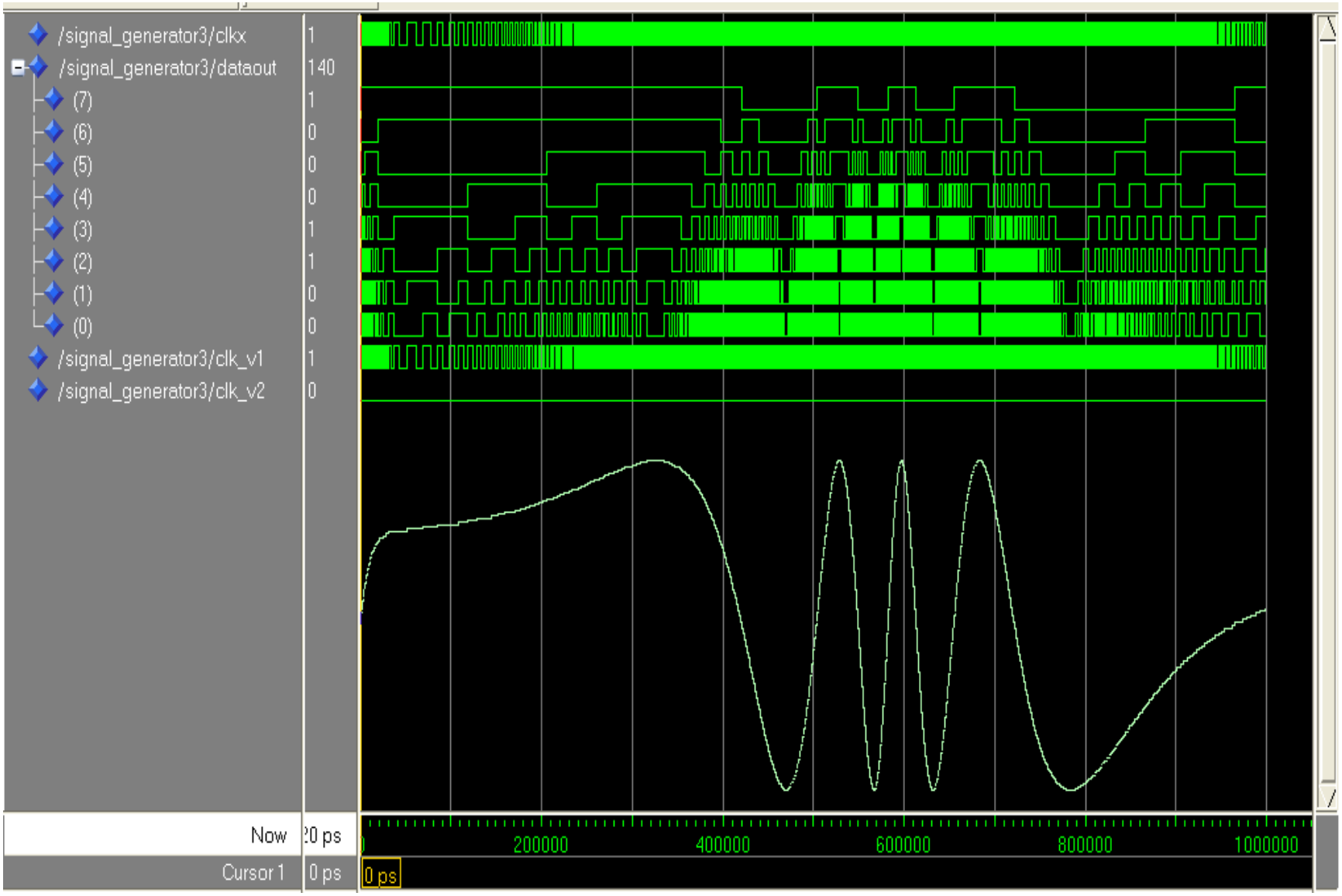
```
22    CLKX <= Clk_v1;
23    ----------------------------------------------------------------------
24    -- This process creates a clk-signal with a variable frequency
25    -- not use if its useful in practice, but it demonstrates what can
26    -- be done.
27    -- The shared variable "Delta" will decrease with the value "Delta_step"
28    -- for each step will the "Delta_step" value change with "Delta_xx"
29    ----------------------------------------------------------------------
30    Clk_generator: process
31    begin
32       Clk_v1 <= not Clk_v1;   -- Toogle the Clk
33       Delta := 1000.0;        -- Ready for a new count down
34       while Delta>0.0 loop    -- while not done
35          wait for 10 ps;      -- adjust this if needed
36          Delta := Delta - Delta_step;  -- one step down
37       end loop;
38
39       if Delta_xx < 0.0 then
40          if Delta_step < 2.0 then
41             Delta_xx := 0.1;
42          end if;
43       else
44          if Delta_step > 198.0 then
45             Delta_xx := -0.1;
46          end if;
47       end if;
48       Delta_Step := Delta_step + Delta_xx;
49    end process Clk_generator;
```



”Sinus-Sweep” generator – Example of Stimuli (8)

```
51      ------------------------------------------------------------------
52      -- This process driven by an external clock signal
53      -- the statement "wait until rising_edge( clk_v1)" do the trick
54      ------------------------------------------------------------------
55      Sinus_generator: process
56         constant Umax:      integer := 127;      -- Max amplitude
57         constant f:         real    := 2.0E6;   -- Frequency [Hz]
58         constant Tper:      real    := 1.0/f;    -- Period of fr.
59      -- If you can find a way to convert real to time please let me know
60         constant Delta:     real    := 1000.0E-12; -- delta time - sec
61         constant DeltaWait: time    := 1000 ps;    -- delta time - ps
62      ------------------------------------------------------------------
63         variable t:     Real := 0.0;  -- Actual time
64         variable angle: real := 0.0;  -- Actual angle in radians
65         variable Usin:  real := 0.0;  -- The sin value [real]
66         variable Usin_int: integer;   -- The sin value as integer
67      begin
68         wait until rising_edge( Clk_v1);
69
70         angle    := 2.0 * MATH_PI * t * f;     -- calculate angle
71         t        := t + Delta;                 -- next time
72         Usin     := real(Umax)*( SIN( angle)+1.0);  -- Usin calculation
73         Usin_int := integer(Usin);            -- convert real to integer
74         Dataout  <= conv_std_logic_vector( Usin_int, 8);  -- to vector
75      end process sinus_generator;
76   end Behavioral;
```

”Sinus-Sweep” generator – Example of Stimuli (9)

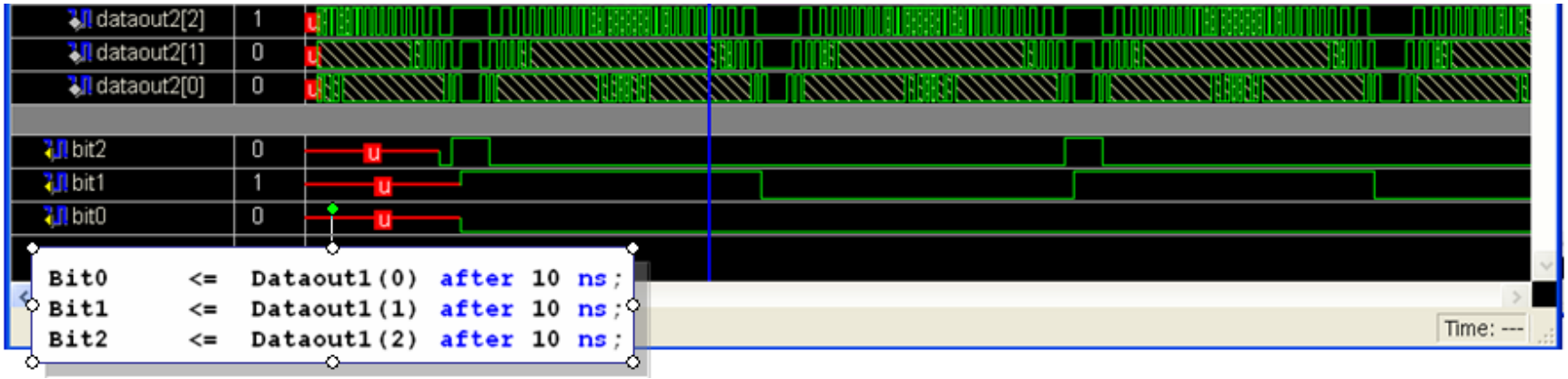


Difference Between – Transport .. After & After (1)

```
 9  entity Signal_generator4 is
10      Port ( Dataout1,
11             Dataout2 :        inout STD_LOGIC_VECTOR (7 downto 0);
12             Bit0,Bit1,Bit2: out     STD_LOGIC);
13  end Signal_generator4;
14
15  architecture Behavioral of Signal_generator4 is
16
17  begin
18     Dataout2 <= Dataout1 after 10 ns;
19
20  -- Bit0      <=  Dataout1(0) after 10 ns;
21  -- Bit1      <=  Dataout1(1) after 10 ns;
22  -- Bit2      <=  Dataout1(2) after 10 ns;
23
24     Bit0      <= transport Dataout1(0) after 10 ns;
25     Bit1      <= transport Dataout1(1) after 10 ns;
26     Bit2      <= transport Dataout1(2) after 10 ns;
27
28     Sinus_generator:
29     process
```

The **After** delay will normally be used to simulate a gates propagation delay – hence glitch rejection needed.
The signal should at least be as long as the delay time to change the output.

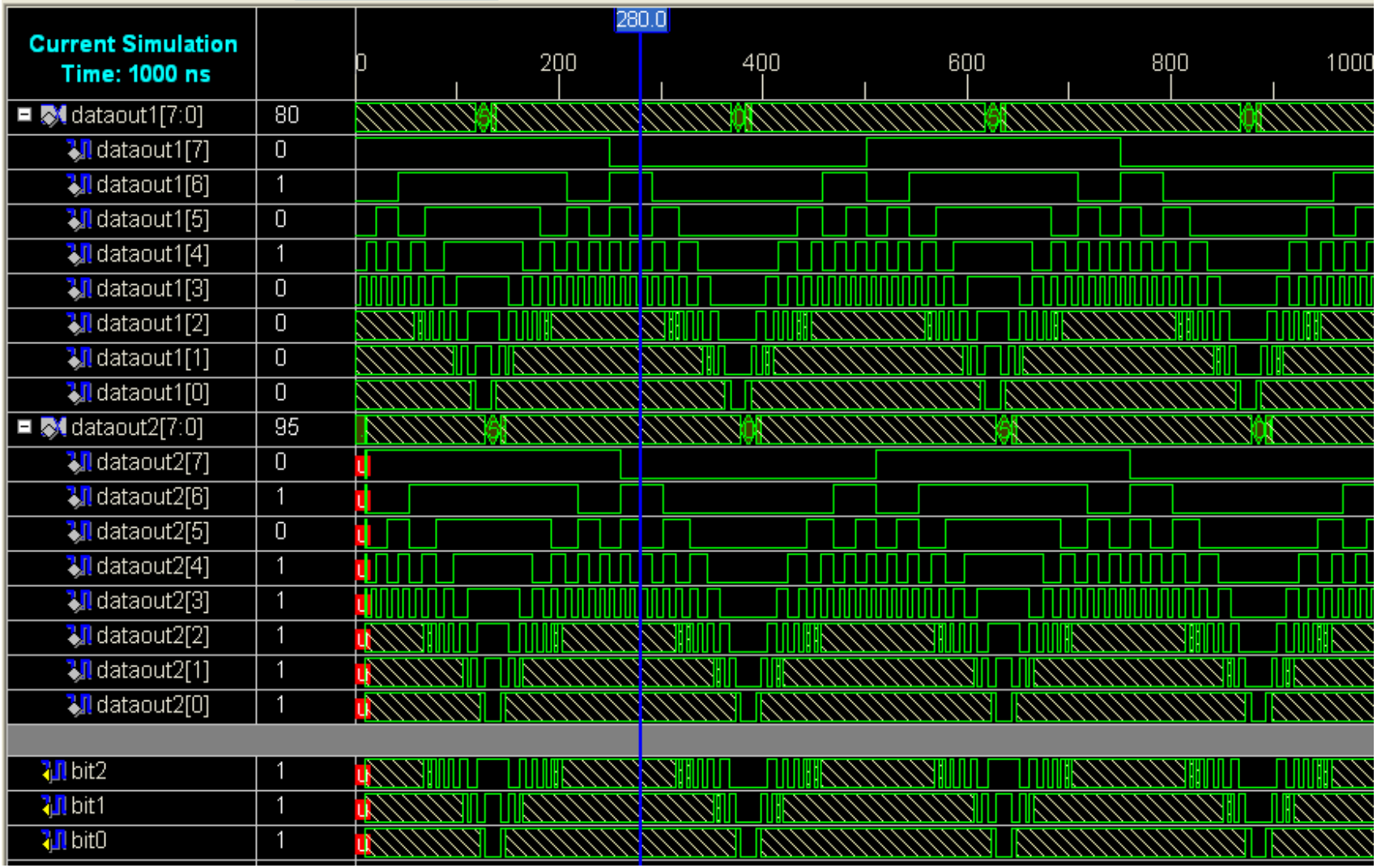
The **Transport** .. **After** used for a transmission line.
The signal will show up after the delay time.

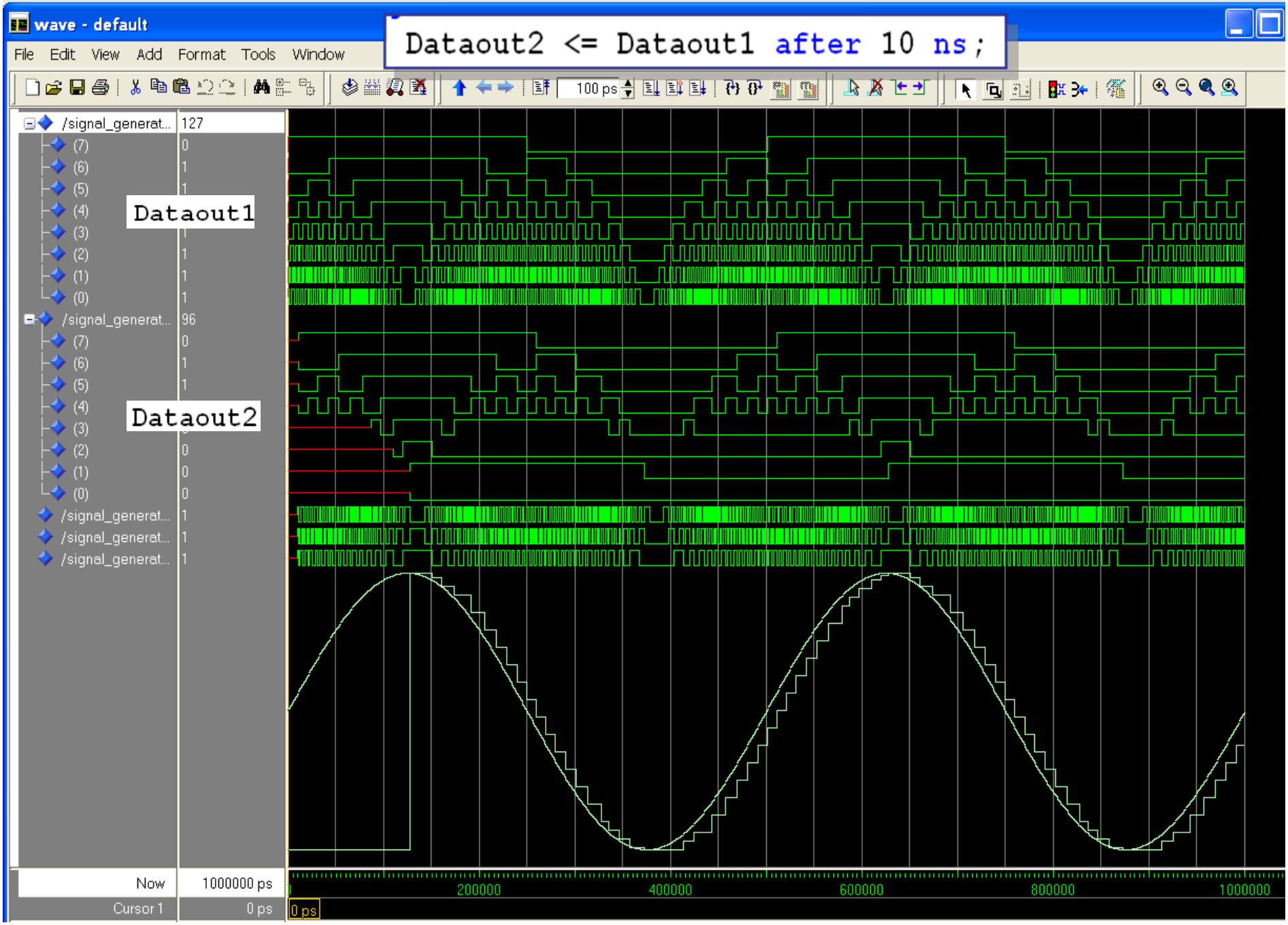


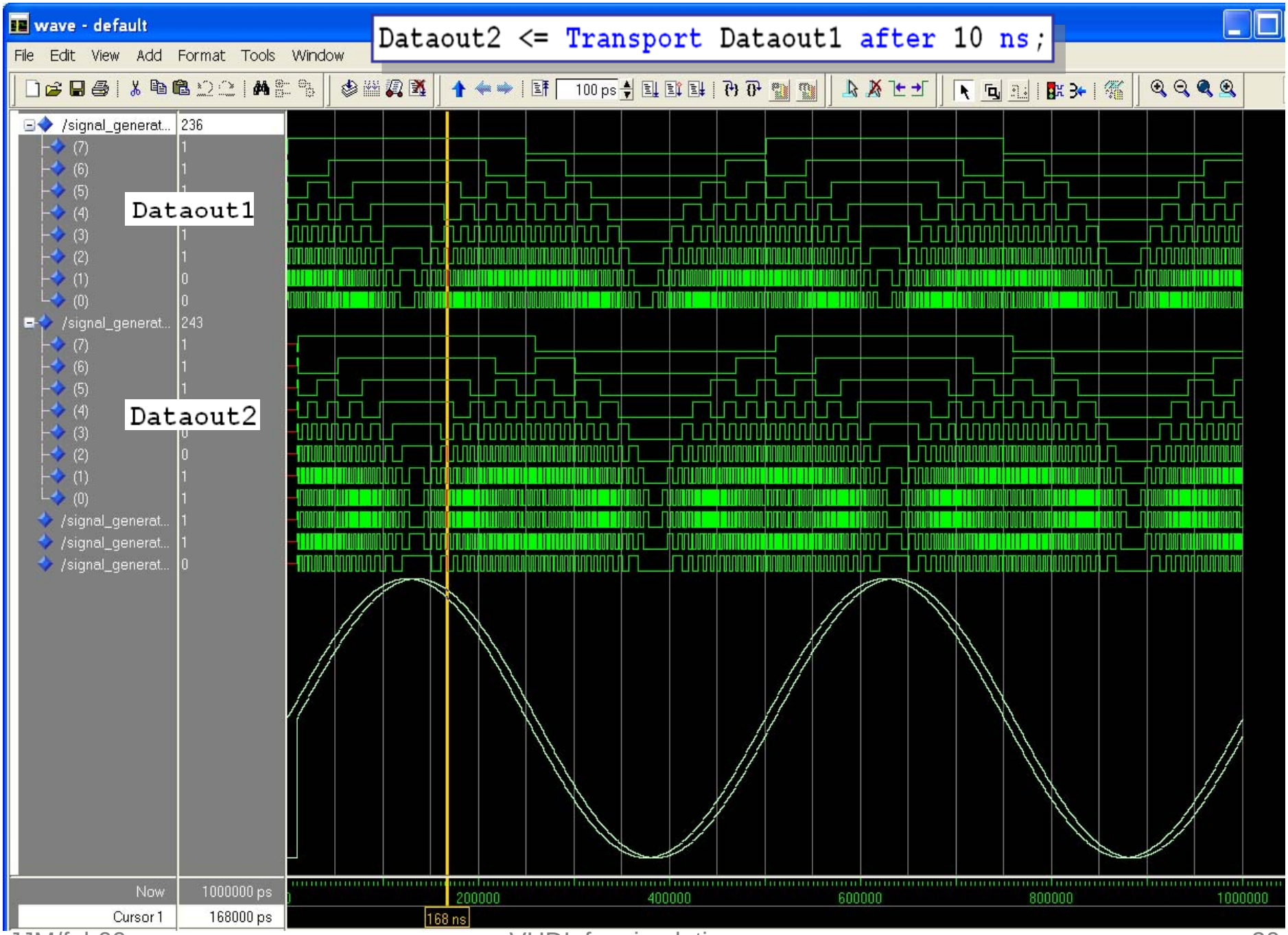
Difference Between – Transport .. After & After (2)

Please compare the ISE simulation and next two slides with Modelsim simulation ??

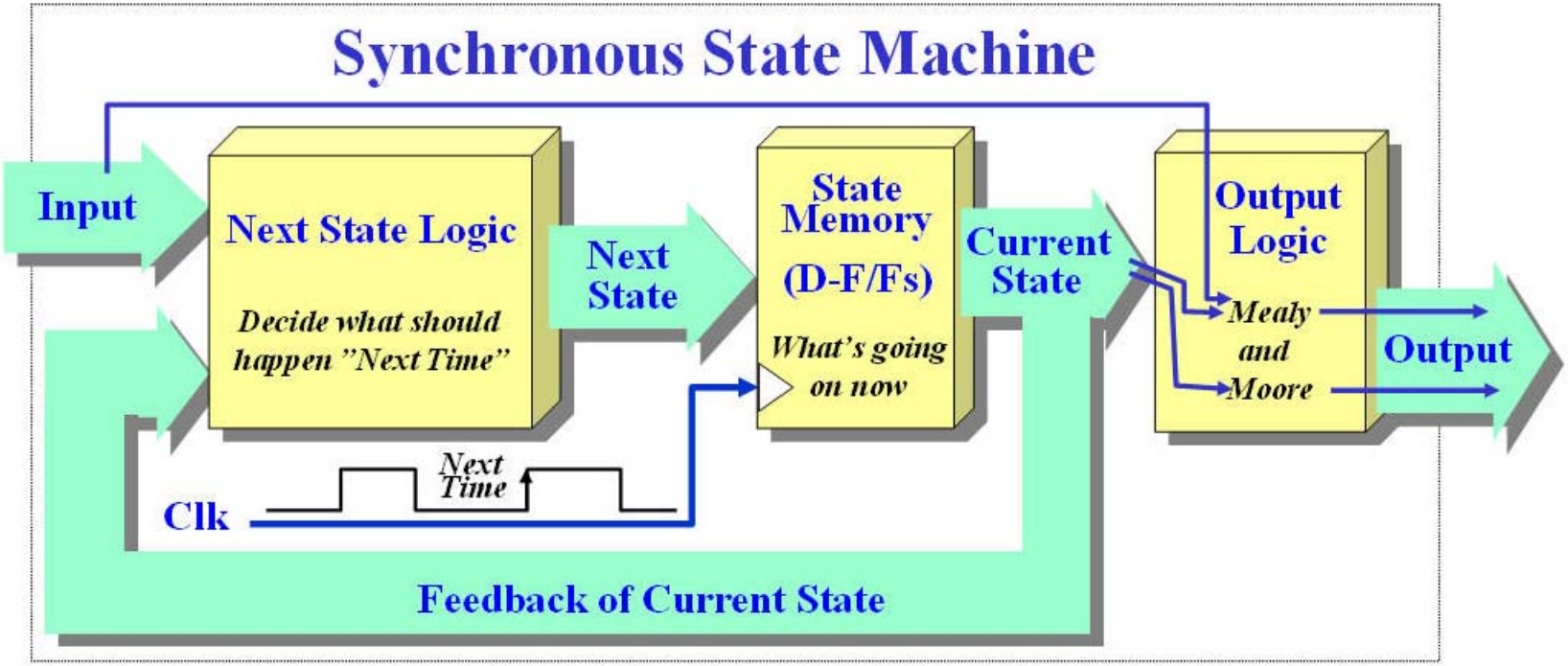
```
Dataout2 <= Dataout1 after 10 ns;
```




wave - default
Dataout2 <= Dataout1 after 10 ns;
File Edit View Add Format Tools Window
100 ps
/signal_generat... 127
(7) 0
(6) 1
(5) 1
(4)
(3)
(2) 1
(1) 1
(0) 1
Dataout1
/signal_generat... 96
(7) 0
(6) 1
(5) 1
(4)
(3)
(2) 0
(1) 0
(0) 0
Dataout2
/signal_generat... 1
/signal_generat... 1
/signal_generat... 1
Now 1000000 ps
Cursor 1 0 ps
200000
400000
600000
800000
1000000
0 ps


wave - default
File Edit View Add Format Tools Window
Dataout2 <= Transport Dataout1 after 10 ns;
100 ps
/signal_generat... 236
(7) 1
(6) 1
(5) 1
(4)
(3) 1
(2) 1
(1) 0
(0) 0
Dataout1
/signal_generat... 243
(7) 1
(6) 1
(5) 1
(4)
(3) 0
(2) 0
(1) 1
(0) 1
Dataout2
/signal_generat... 1
/signal_generat... 1
/signal_generat... 0
Now 1000000 ps
Cursor 1 168000 ps
200000
400000
600000
800000
1000000
168 ns

State Machines for simulation



The structure of a hardware state machine more or less fixed and can be drawn as above

State Machines often used in concern with highlevel languages like C and Java.
When it comes to simulation in VHDL will the idea of thinking in states surely shows to very usefull as well.

How ever will you learn that you can make the rules and structure of a Simulation State Machine like you like them to be.

”Function” generator – Example of Stimuli (10)

```
8  entity State_machine_demo is
9      Port ( DataOut : out  STD_LOGIC_VECTOR (7 downto 0));
10 end State_machine_demo;
11
12 architecture Behavioral of State_machine_demo is
13    -- The statenames can be used
14    type States is (Start, State1, State2, State3);
15    shared variable State: States;
16    signal            Clk: BIT := '0'; -- NOTE the type BIT
17 begin
18
19    Clk <= not Clk after 1 ns;-- Seems to work better together with not
20
21    -- This demonstrates how a statemachine for simulation could be
22    -- implemented, please note the difference compared with "real"
23    -- statemachines for hardware.
24    -- You can use the "Wait for" statement to created timing and delay
25    -- Also internal "for-loops" allowed ...
26    -- Finally can the statemachine wait for a Rising_egde or similar
27    -- inside a state (surely not for synthesize)
28    State_machine:
29    process
30       variable i,Value: integer;
31    begin
32       State := Start;
33       loop
34          case State is
35             when start =>
36                      Dataout <= conv_std_logic_vector( 10,8);
37                      wait for 30 ns;
38                      State   := State1;
```

”Function” generator – Example of Stimuli (10)

```
39            when state1 =>
40                    for i in 0 to 10 loop
41                        Dataout <= conv_std_logic_vector( i*25,8);
42                        wait for 50 ns;
43                    end loop;
44                    Value    := 0;
45                    State    := State2;
46            when state2 =>
47                    wait until Clk'event and Clk='1'; -- Rising_edge
48                    Dataout <= conv_std_logic_vector( Value,8);
49                    Value := Value + 1;
50                    if Value >254 then
51                       State := State3; -- Change state now
52                    end if;
53            when state3 =>
54                    Dataout <= conv_std_logic_vector( 127,8);
55                    wait for 30 ns;
56                    State    := State1;
57          end case;
58          -- wait for 1 ns;
59       end loop;
60    end process;
61
62 end Behavioral;
```

”Function” generator – Example of Stimuli (10)