CMSC 132, First Semester AY 2020-2021

Computer Architecture

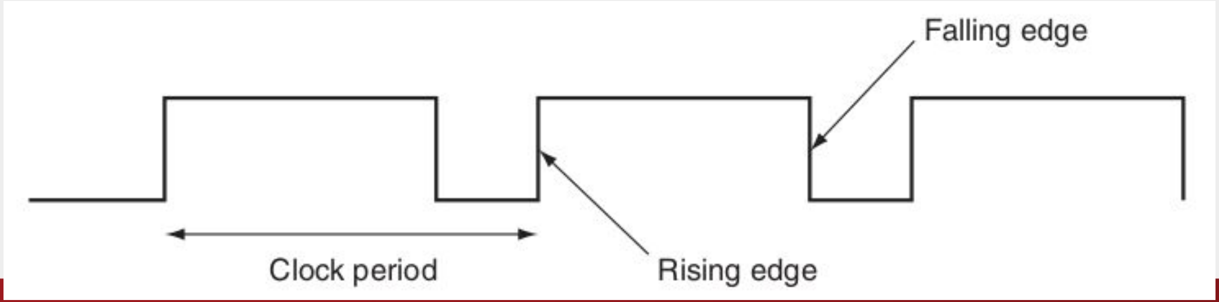
Joseph Anthony C. Hermocilla

Sequential/State Elements

Clocks

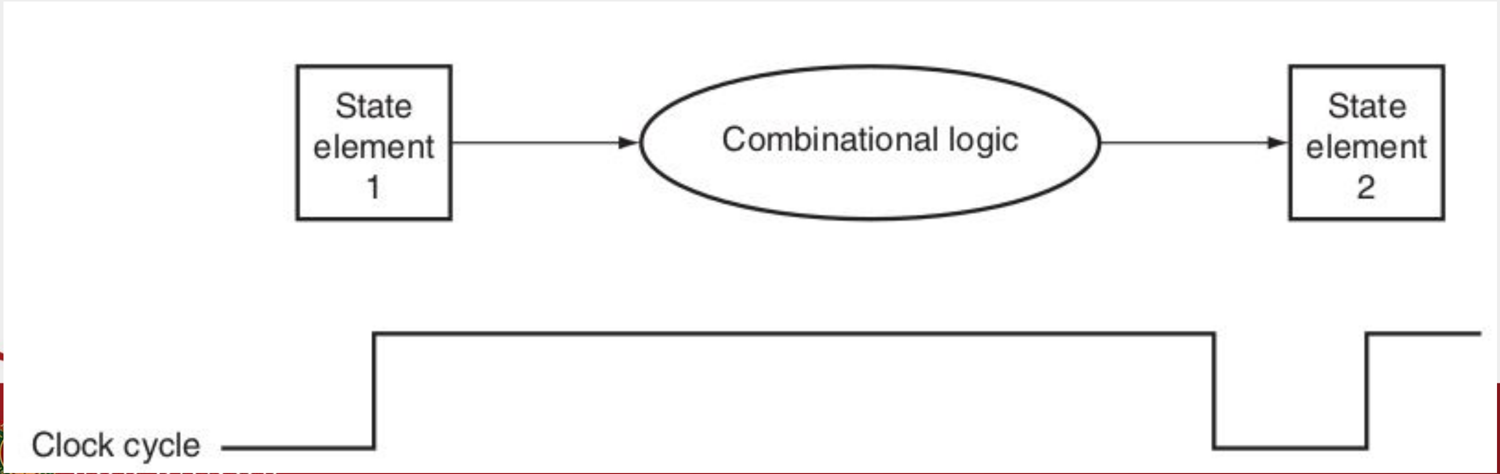
Clocks

- Needed in sequential logic to decide when an element that contains state should be updated
- A free running signal with a fixed cycle time or clock period
- Cycle time is divided into two portions: clock is high or clock is low
- Edge-triggered clocking - all changes in state elements occur on a clock edge



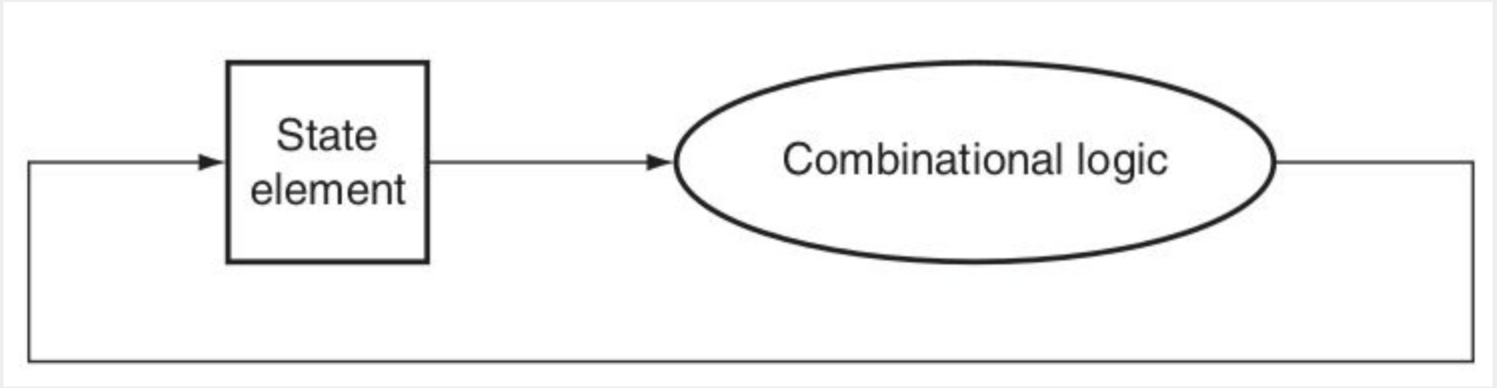
Clocks (cont.)

- State elements, whose outputs change only after a clock edge, provide valid input to the combinational block
- To ensure that the values written into the state elements on the active clock edge are valid, the clock must have a long enough period

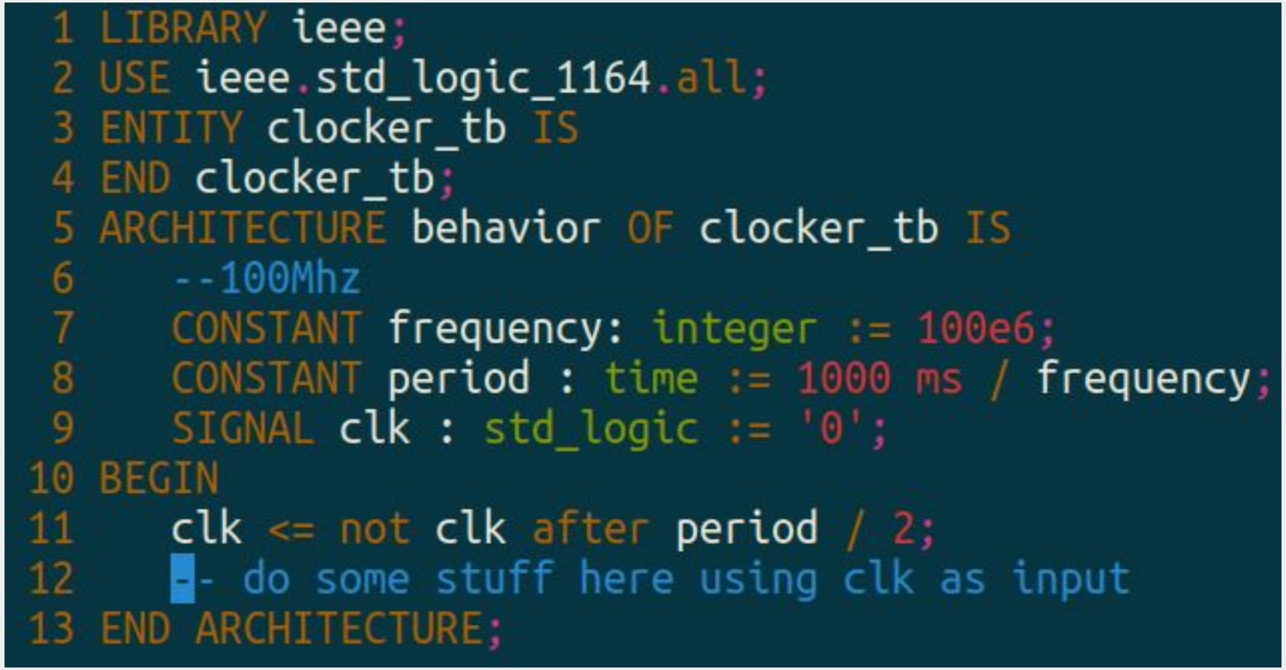


Clocks (cont.)

- Using edge-triggered methodology allows the same state element to be used as both input and output to the same combinational block



Clock in VHDL

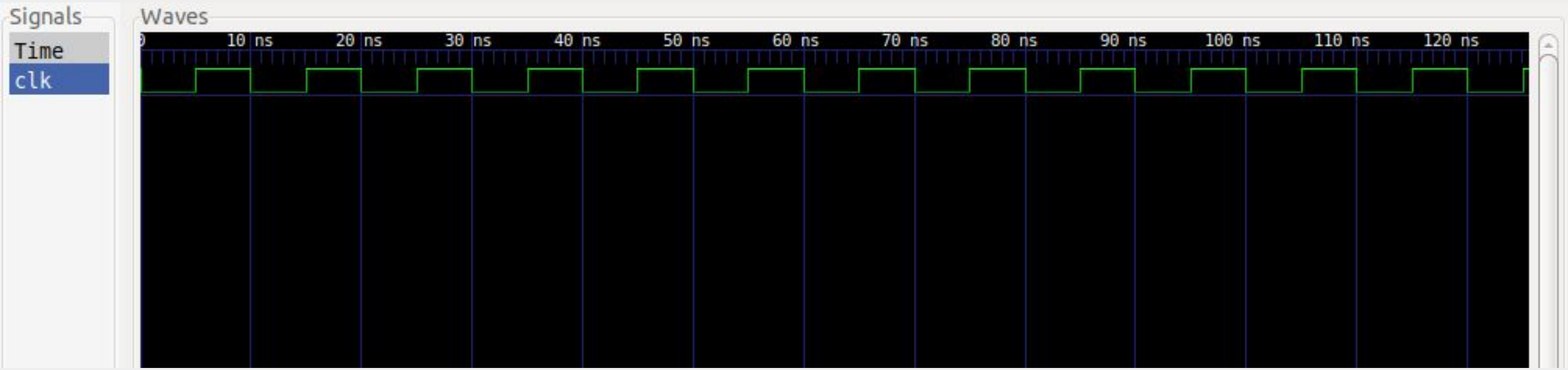


```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY clocker_tb IS
END clocker_tb;
ARCHITECTURE behavior OF clocker_tb IS
    --100Mhz
    CONSTANT frequency: integer := 100e6;
    CONSTANT period : time := 1000 ms / frequency;
    SIGNAL clk : std_logic := '0';
BEGIN
    clk <= not clk after period / 2;
    -- do some stuff here using clk as input
END ARCHITECTURE;
```

Testing

```
$ ghdl -a clocker_tb.vhdl
$ ghdl -e clocker_tb
$ ghdl -r clocker_tb --vcd=clocker.vcd --stop-time=500ns
$ gtkwave clocker.vcd
```

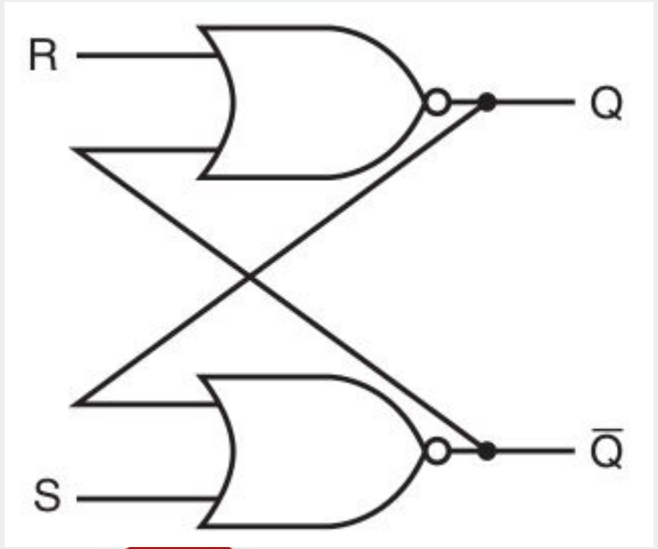
GTKWave output of clocker

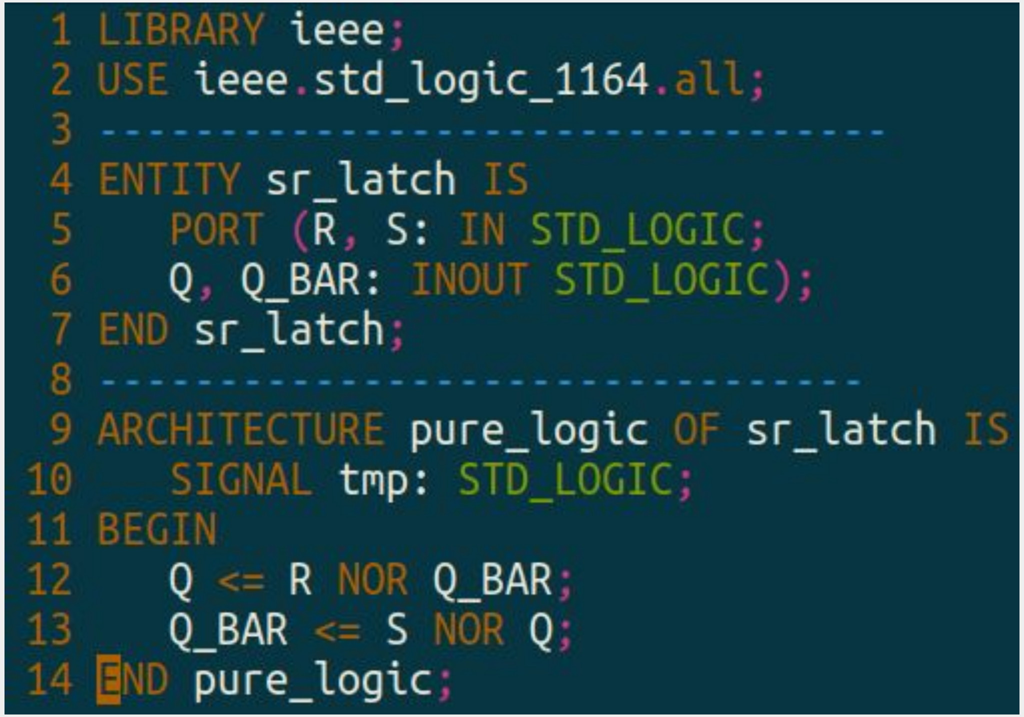


Memory Elements:
Flip-Flops, Latches, and Registers

S-R latch (set-reset)

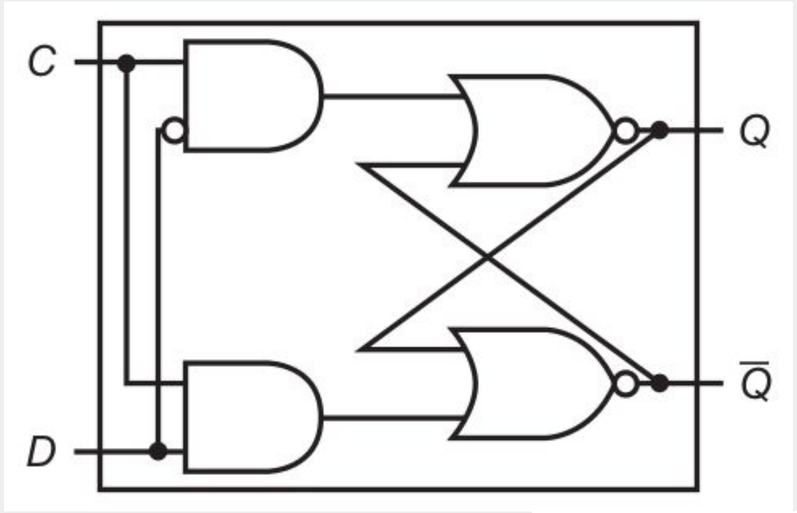
- Unclocked - no clock input

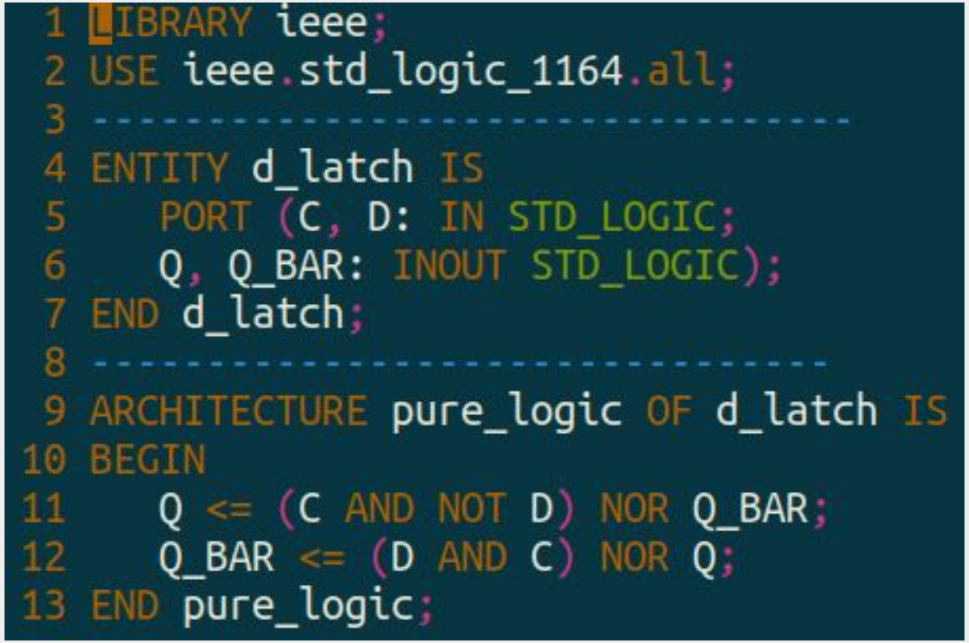




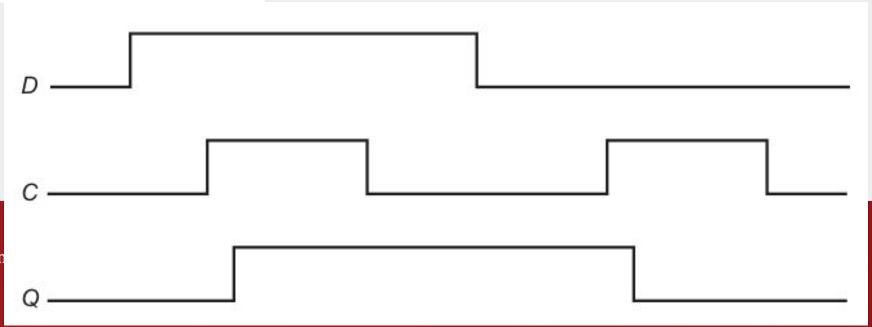
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-----------------------------------
ENTITY sr_latch IS
   PORT (R, S: IN STD_LOGIC;
   Q, Q_BAR: INOUT STD_LOGIC);
END sr_latch;
-----------------------------------
ARCHITECTURE pure_logic OF sr_latch IS
   SIGNAL tmp: STD_LOGIC;
BEGIN
   Q <= R NOR Q_BAR;
   Q_BAR <= S NOR Q;
END pure_logic;
```

D latch

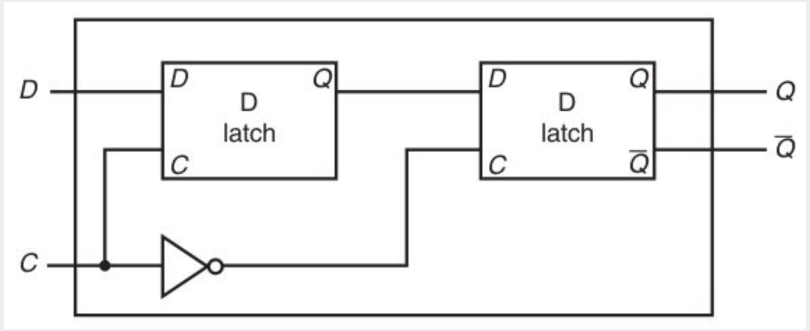


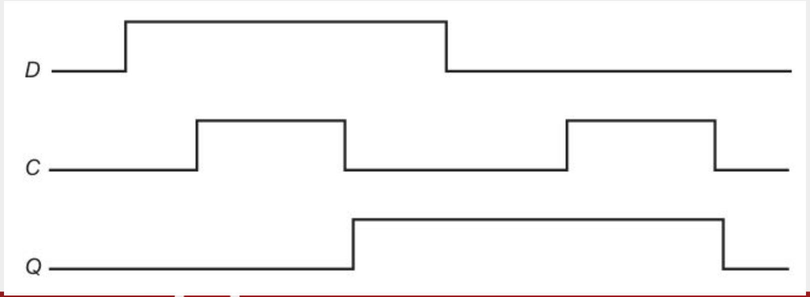


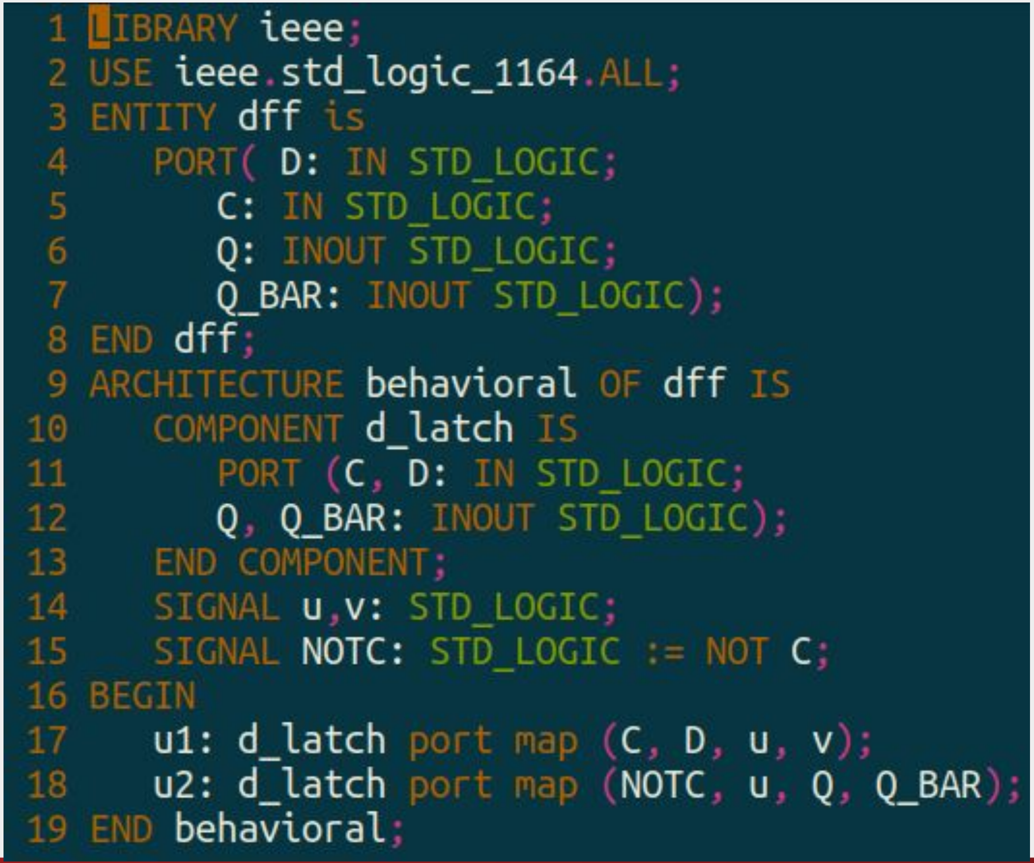
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------------------
ENTITY d_latch IS
   PORT (C, D: IN STD_LOGIC;
   Q, Q_BAR: INOUT STD_LOGIC);
END d_latch;
----------------------------------------
ARCHITECTURE pure_logic OF d_latch IS
BEGIN
   Q <= (C AND NOT D) NOR Q_BAR;
   Q_BAR <= (D AND C) NOR Q;
END pure_logic;
```



D flip-flop

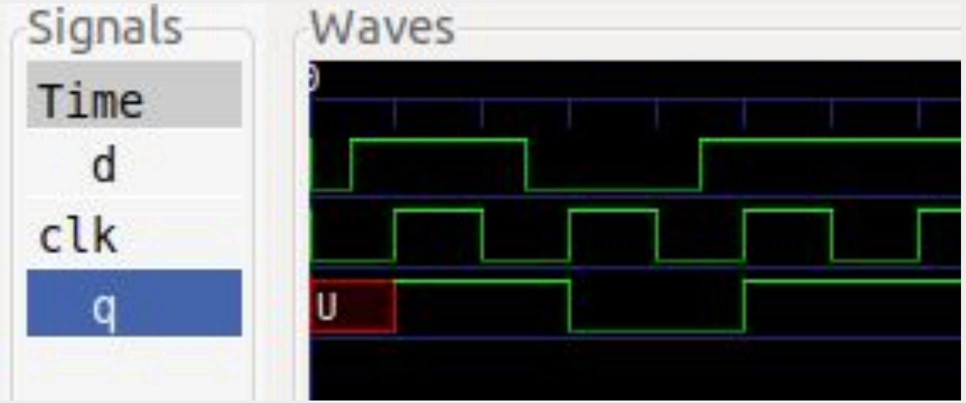


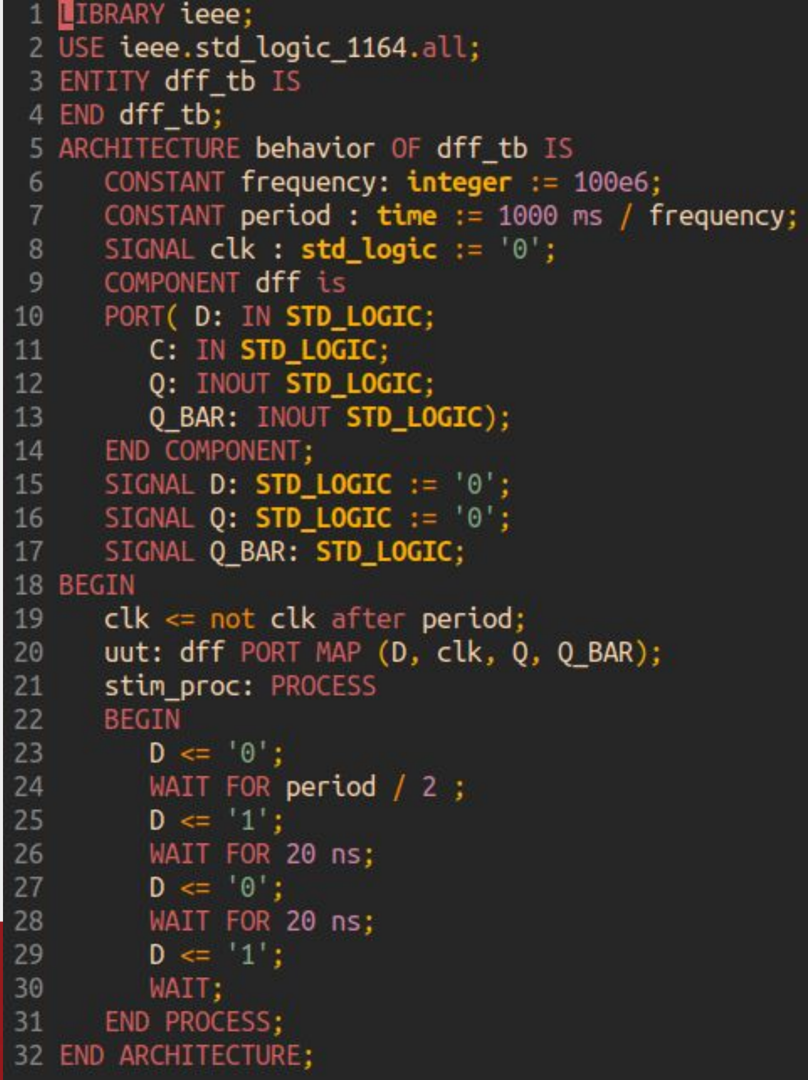




```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY dff is
    PORT( D: IN STD_LOGIC;
        C: IN STD_LOGIC;
        Q: INOUT STD_LOGIC;
        Q_BAR: INOUT STD_LOGIC);
END dff;
ARCHITECTURE behavioral OF dff IS
    COMPONENT d_latch IS
        PORT (C, D: IN STD_LOGIC;
        Q, Q_BAR: INOUT STD_LOGIC);
    END COMPONENT;
    SIGNAL u,v: STD_LOGIC;
    SIGNAL NOTC: STD_LOGIC := NOT C;
BEGIN
    u1: d_latch port map (C, D, u, v);
    u2: d_latch port map (NOTC, u, Q, Q_BAR);
END behavioral;
```

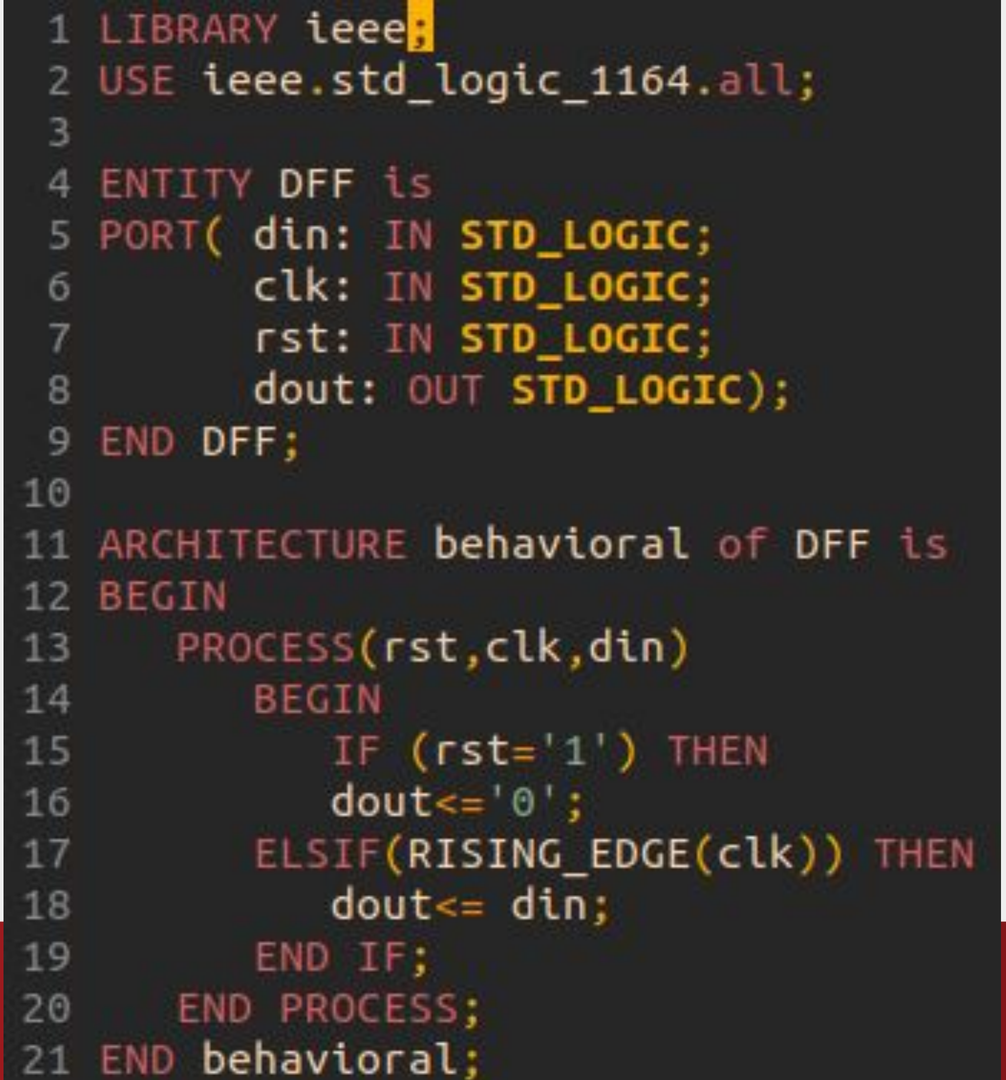
D Flip-Flop Testbench




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY dff_tb IS
END dff_tb;
ARCHITECTURE behavior OF dff_tb IS
   CONSTANT frequency: integer := 100e6;
   CONSTANT period : time := 1000 ms / frequency;
   SIGNAL clk : std_logic := '0';
   COMPONENT dff is
   PORT( D: IN STD_LOGIC;
      C: IN STD_LOGIC;
      Q: INOUT STD_LOGIC;
      Q_BAR: INOUT STD_LOGIC);
   END COMPONENT;
   SIGNAL D: STD_LOGIC := '0';
   SIGNAL Q: STD_LOGIC := '0';
   SIGNAL Q_BAR: STD_LOGIC;
BEGIN
   clk <= not clk after period;
   uut: dff PORT MAP (D, clk, Q, Q_BAR);
   stim_proc: PROCESS
   BEGIN
      D <= '0';
      WAIT FOR period / 2 ;
      D <= '1';
      WAIT FOR 20 ns;
      D <= '0';
      WAIT FOR 20 ns;
      D <= '1';
      WAIT;
   END PROCESS;
END ARCHITECTURE;
```

University of the Philippines
LOS BAÑOS

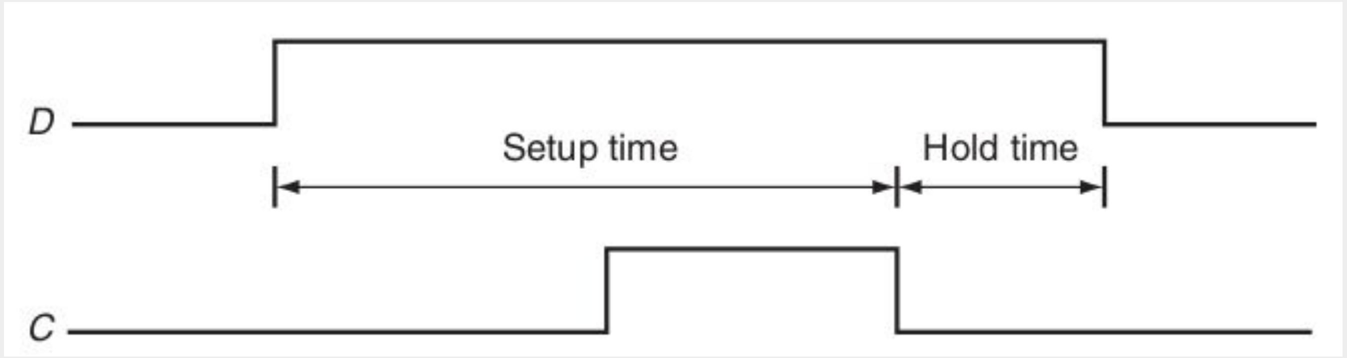
Simpler Code for DFF



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY DFF is
PORT( din: IN STD_LOGIC;
      clk: IN STD_LOGIC;
      rst: IN STD_LOGIC;
      dout: OUT STD_LOGIC);
END DFF;

ARCHITECTURE behavioral of DFF is
BEGIN
    PROCESS(rst,clk,din)
        BEGIN
            IF (rst='1') THEN
            dout<='0';
        ELSIF(RISING_EDGE(clk)) THEN
            dout<= din;
        END IF;
    END PROCESS;
END behavioral;
```

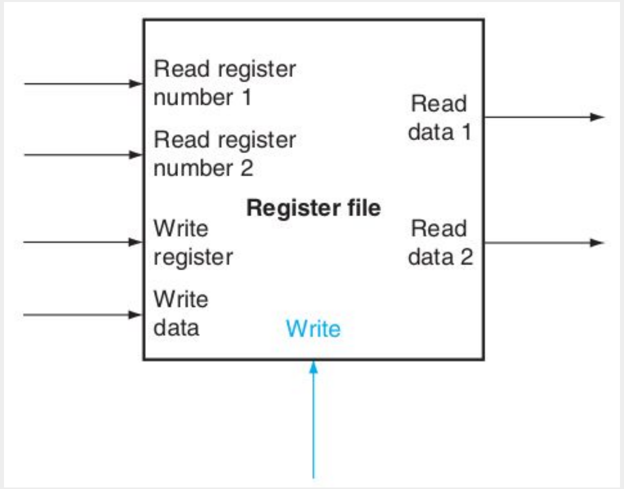
Setup Time and Hold Time

- Setup time - minimum time that input must be valid before the clock edge
- Hold time - minimum time that input must be valid after the clock edge



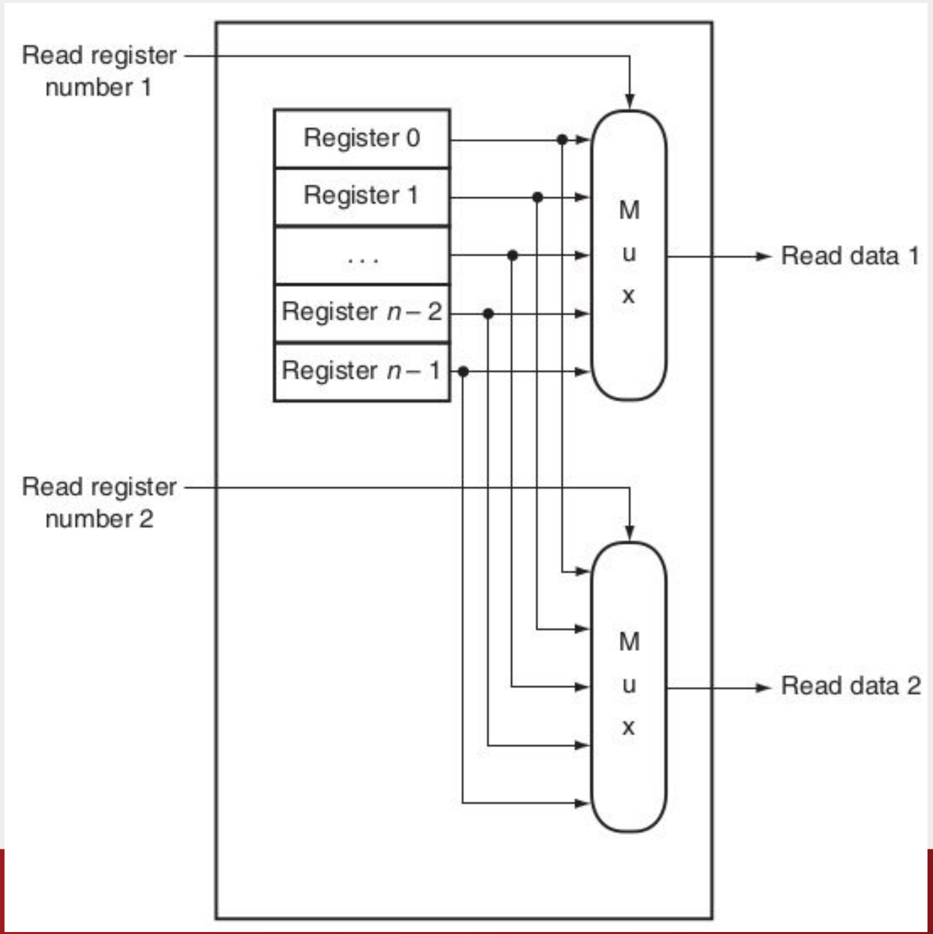
Register Files

- A register file consists of a set of registers that can be read and written by supplying a register number to be accessed, example: ‘al’



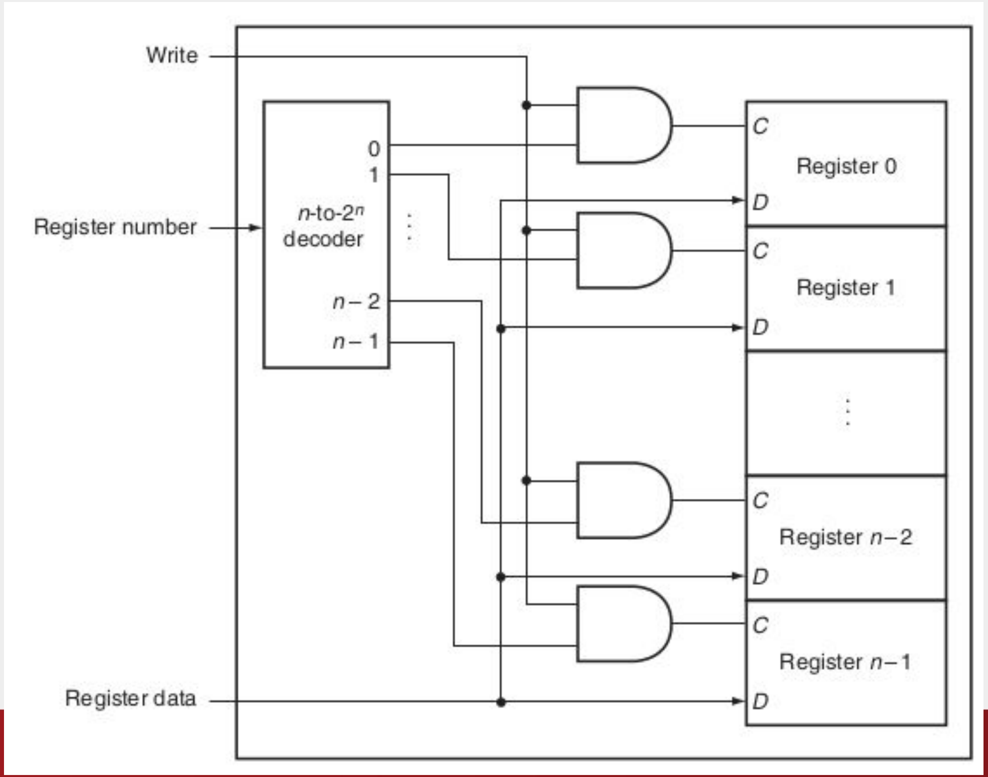
Register Files (cont..)

- Implementation of read



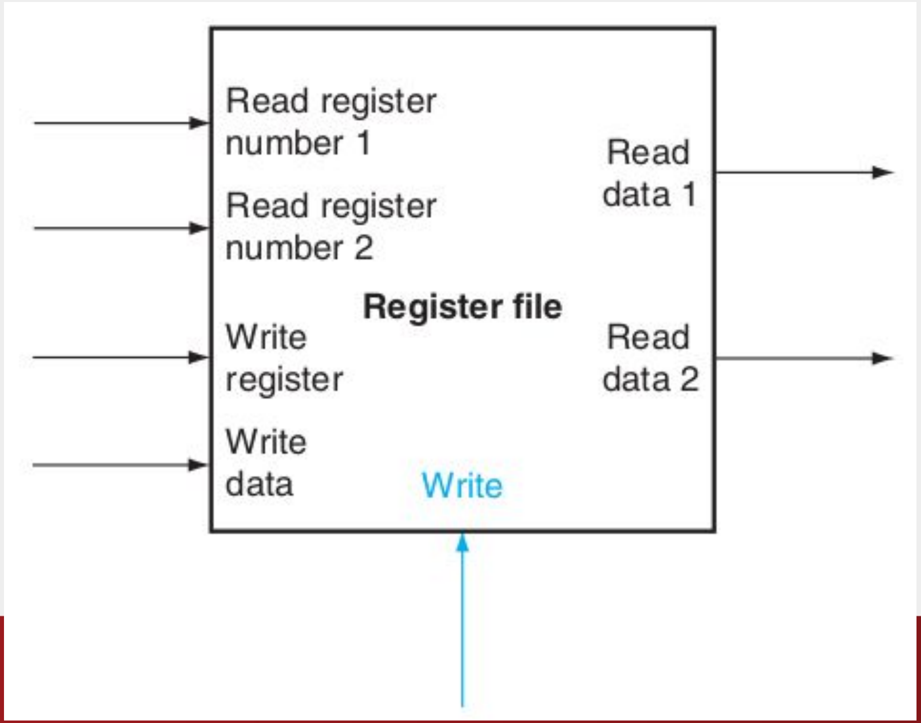
Register Files (cont..)

- Implementation of write

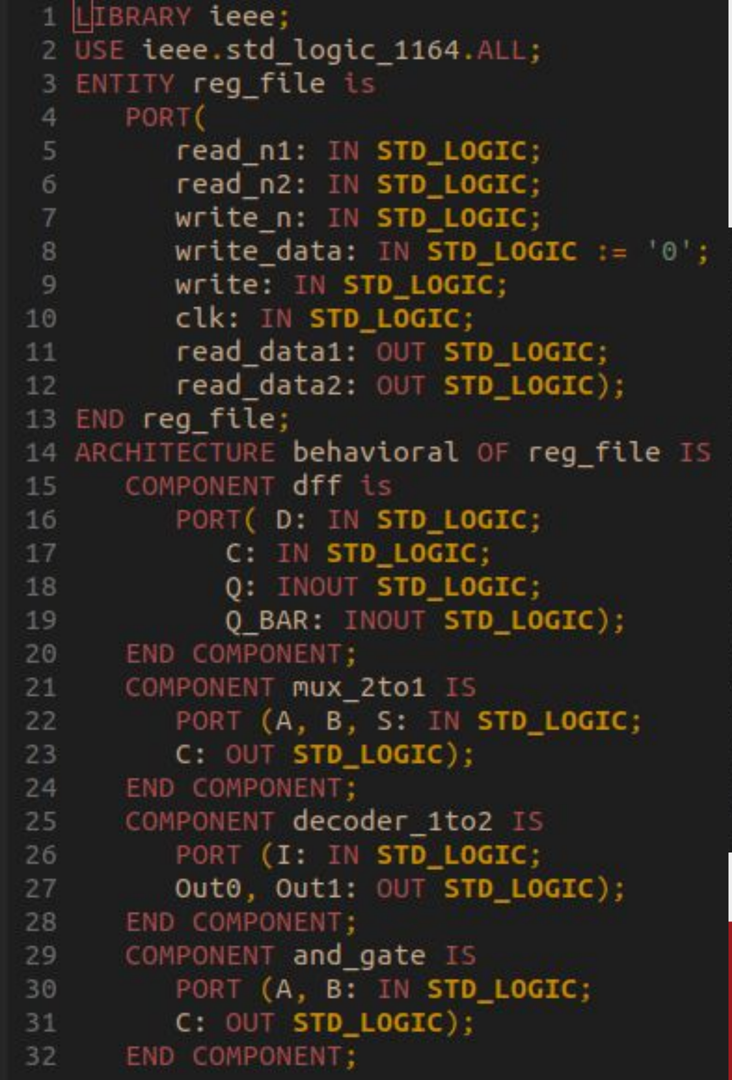


Register Files Example: with 2 1-bit registers

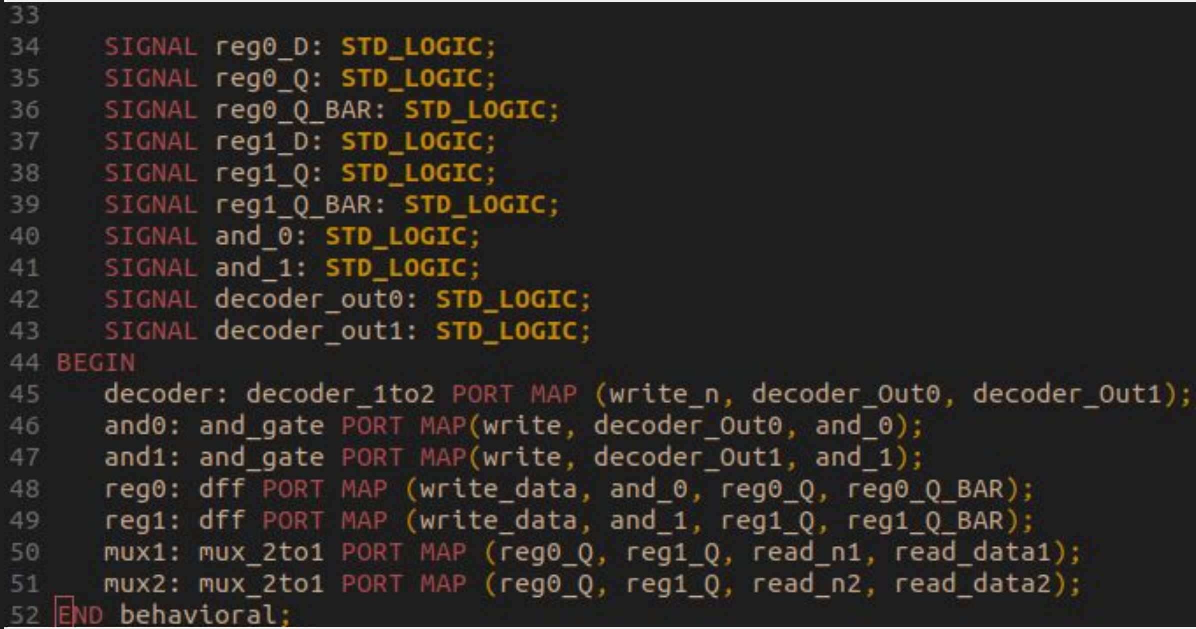
- With 2 1-bit registers



Register Files Example: with 2 1-bit registers (v1)

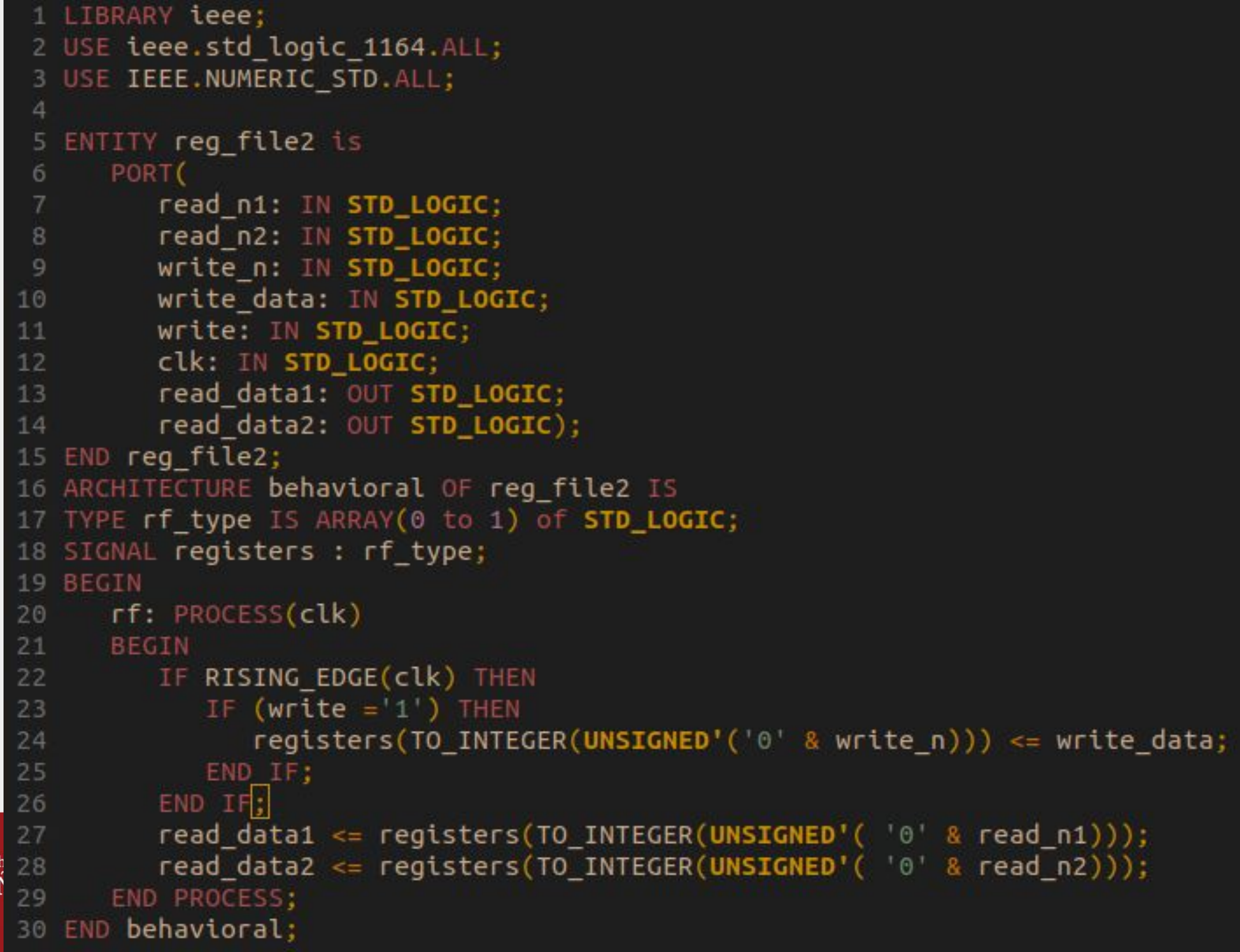


```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY reg_file is
    PORT(
        read_n1: IN STD_LOGIC;
        read_n2: IN STD_LOGIC;
        write_n: IN STD_LOGIC;
        write_data: IN STD_LOGIC := '0';
        write: IN STD_LOGIC;
        clk: IN STD_LOGIC;
        read_data1: OUT STD_LOGIC;
        read_data2: OUT STD_LOGIC);
END reg_file;
ARCHITECTURE behavioral OF reg_file IS
    COMPONENT dff is
        PORT( D: IN STD_LOGIC;
            C: IN STD_LOGIC;
            Q: INOUT STD_LOGIC;
            Q_BAR: INOUT STD_LOGIC);
    END COMPONENT;
    COMPONENT mux_2to1 IS
        PORT (A, B, S: IN STD_LOGIC;
        C: OUT STD_LOGIC);
    END COMPONENT;
    COMPONENT decoder_1to2 IS
        PORT (I: IN STD_LOGIC;
        Out0, Out1: OUT STD_LOGIC);
    END COMPONENT;
    COMPONENT and_gate IS
        PORT (A, B: IN STD_LOGIC;
        C: OUT STD_LOGIC);
    END COMPONENT;
```



```vhdl

    SIGNAL reg0_D: STD_LOGIC;
    SIGNAL reg0_Q: STD_LOGIC;
    SIGNAL reg0_Q_BAR: STD_LOGIC;
    SIGNAL reg1_D: STD_LOGIC;
    SIGNAL reg1_Q: STD_LOGIC;
    SIGNAL reg1_Q_BAR: STD_LOGIC;
    SIGNAL and_0: STD_LOGIC;
    SIGNAL and_1: STD_LOGIC;
    SIGNAL decoder_out0: STD_LOGIC;
    SIGNAL decoder_out1: STD_LOGIC;
BEGIN
    decoder: decoder_1to2 PORT MAP (write_n, decoder_Out0, decoder_Out1);
    and0: and_gate PORT MAP(write, decoder_Out0, and_0);
    and1: and_gate PORT MAP(write, decoder_Out1, and_1);
    reg0: dff PORT MAP (write_data, and_0, reg0_Q, reg0_Q_BAR);
    reg1: dff PORT MAP (write_data, and_1, reg1_Q, reg1_Q_BAR);
    mux1: mux_2to1 PORT MAP (reg0_Q, reg1_Q, read_n1, read_data1);
    mux2: mux_2to1 PORT MAP (reg0_Q, reg1_Q, read_n2, read_data2);
END behavioral;
```

Register Files Example: with 2 1-bit registers (v2)



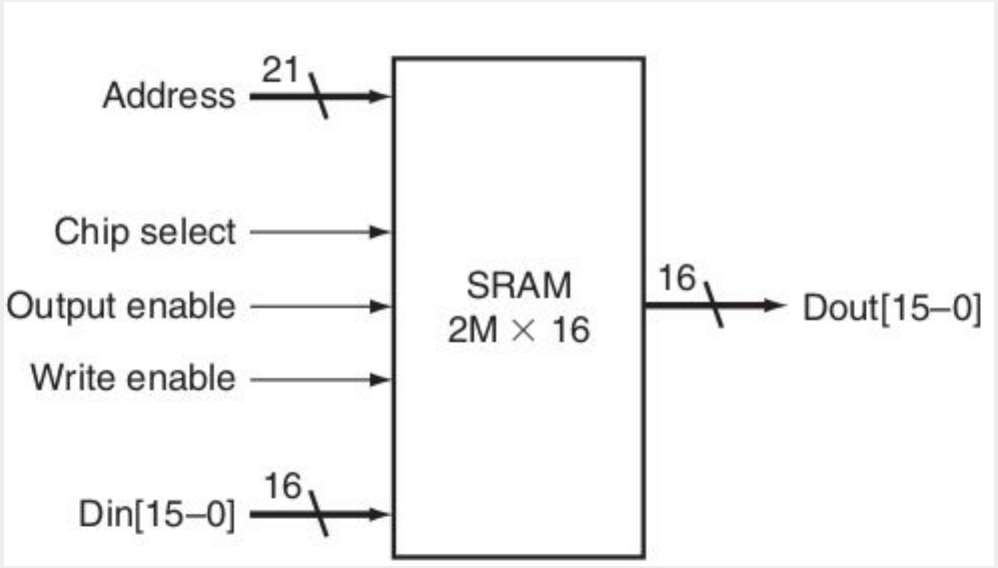
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY reg_file2 is
    PORT(
        read_n1: IN STD_LOGIC;
        read_n2: IN STD_LOGIC;
        write_n: IN STD_LOGIC;
        write_data: IN STD_LOGIC;
        write: IN STD_LOGIC;
        clk: IN STD_LOGIC;
        read_data1: OUT STD_LOGIC;
        read_data2: OUT STD_LOGIC);
END reg_file2;
ARCHITECTURE behavioral OF reg_file2 IS
TYPE rf_type IS ARRAY(0 to 1) of STD_LOGIC;
SIGNAL registers : rf_type;
BEGIN
    rf: PROCESS(clk)
    BEGIN
        IF RISING_EDGE(clk) THEN
            IF (write ='1') THEN
                registers(TO_INTEGER(UNSIGNED'('0' & write_n))) <= write_data;
            END IF;
        END IF;
        read_data1 <= registers(TO_INTEGER(UNSIGNED'( '0' & read_n1)));
        read_data2 <= registers(TO_INTEGER(UNSIGNED'( '0' & read_n2)));
    END PROCESS;
END behavioral;
```

Memory Elements:
SRAMs and DRAMs

University of the Philippines
LOS BAÑOS

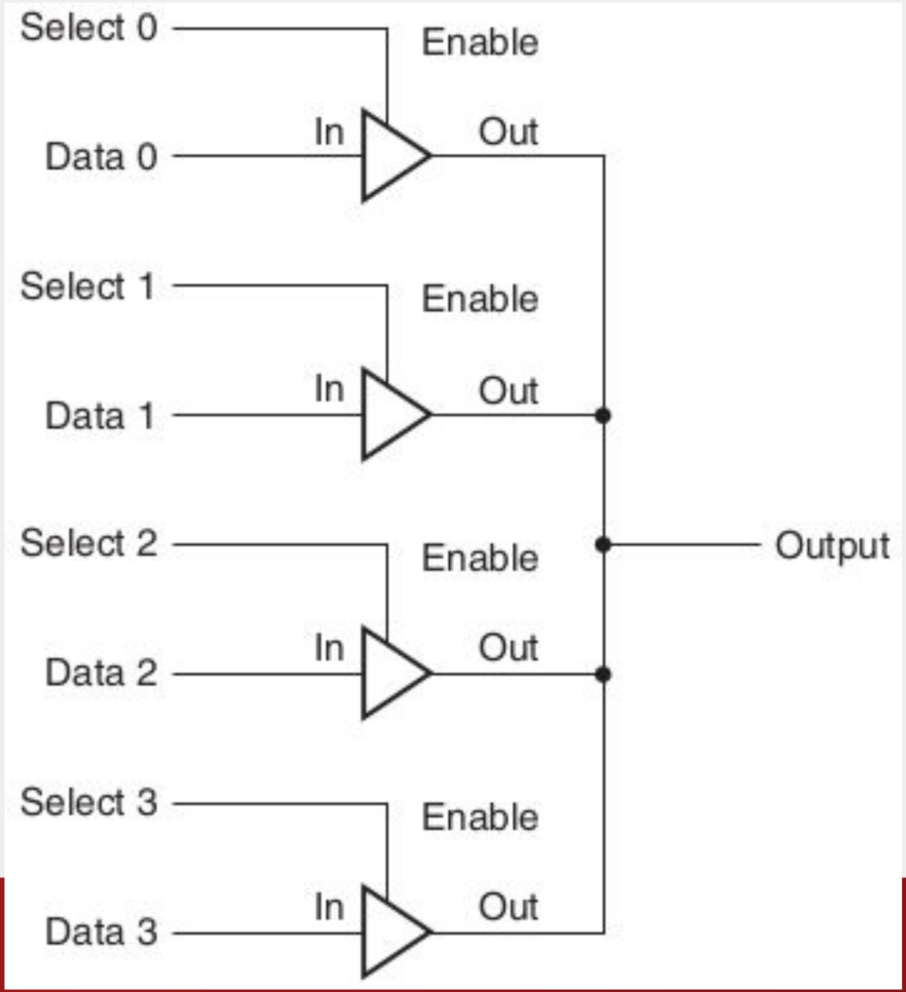
Static RAM

- Has a specific configuration in terms of the number of addressable locations and width of each addressable location
- Example: 2M x 16 SRAM
- Height = # of addressable locations, 2M
- Width = # bits per addressable unit, 16 bits

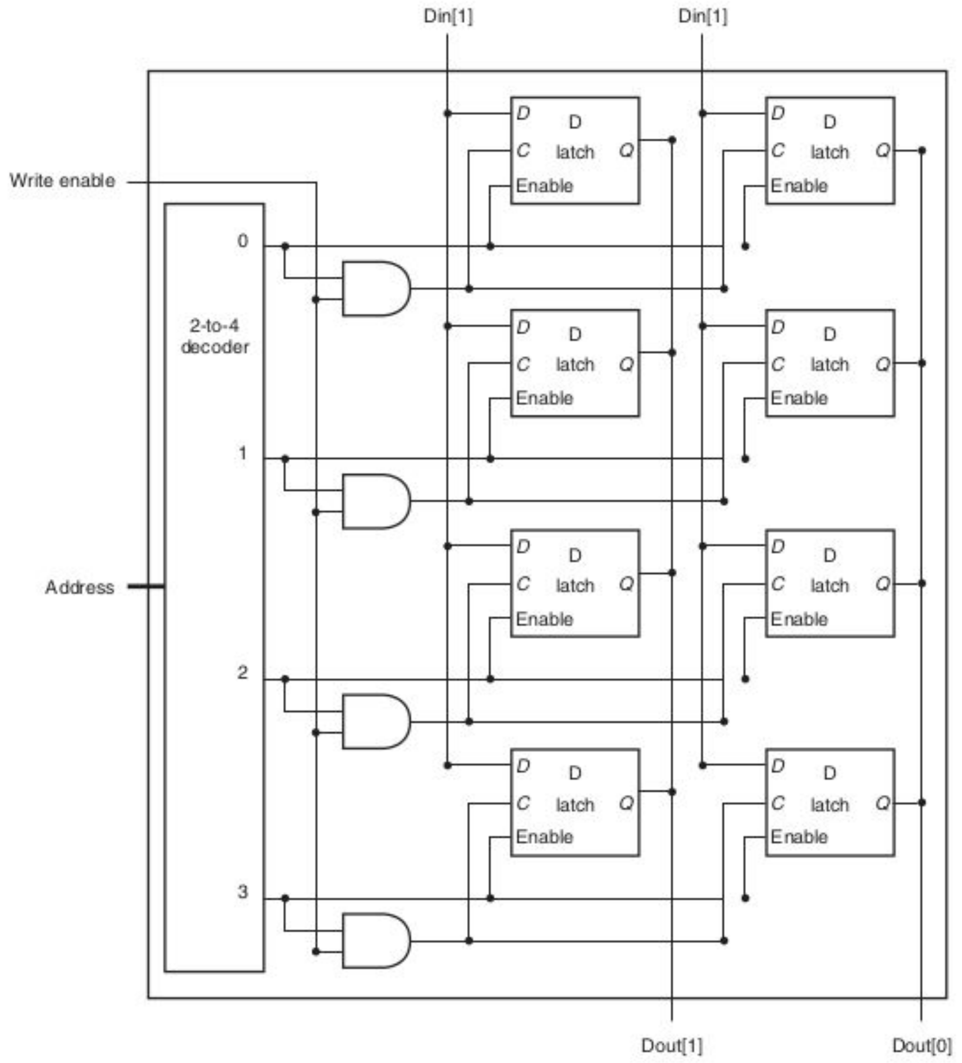


Three-state buffer

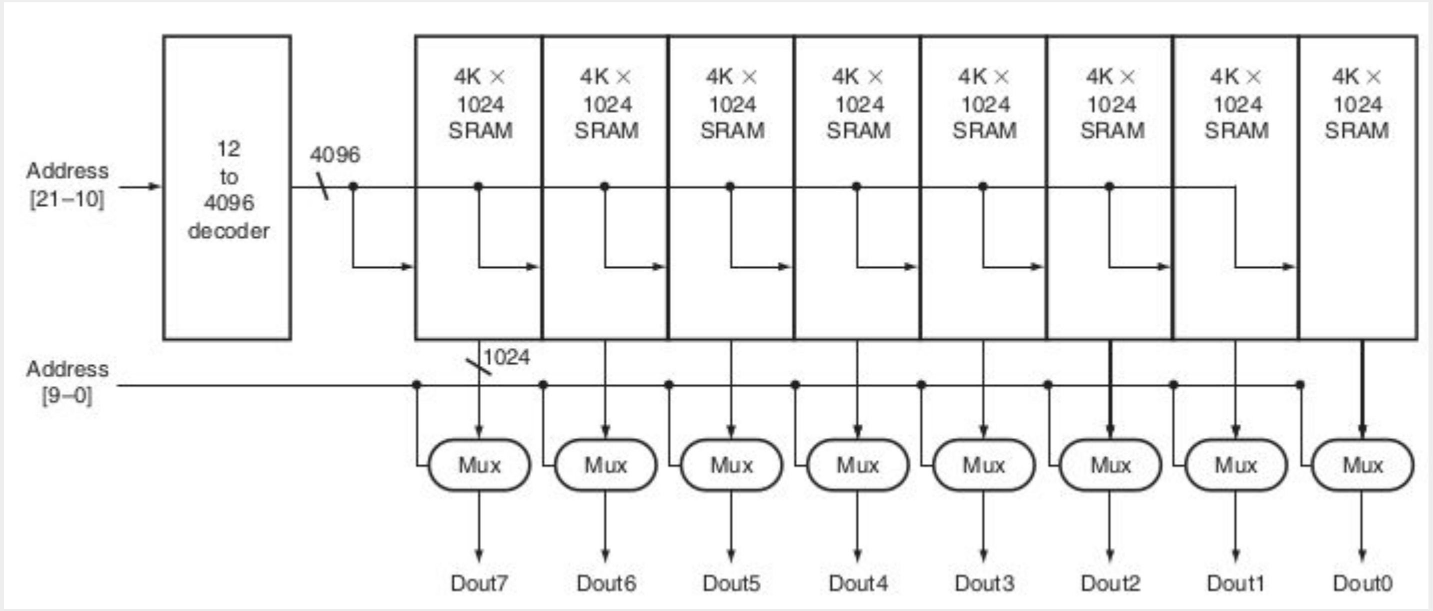
- Used as shared line which memory cells can assert
- As substitute for giant multiplexer



4 x 2 SRAM


Din[1]
Din[1]
Write enable
Address
2-to-4 decoder
0
1
2
3
D
D latch
C
Q
Enable
Dout[1]
Dout[0]

4M × 8 SRAM as an array of 4K × 1024 arrays

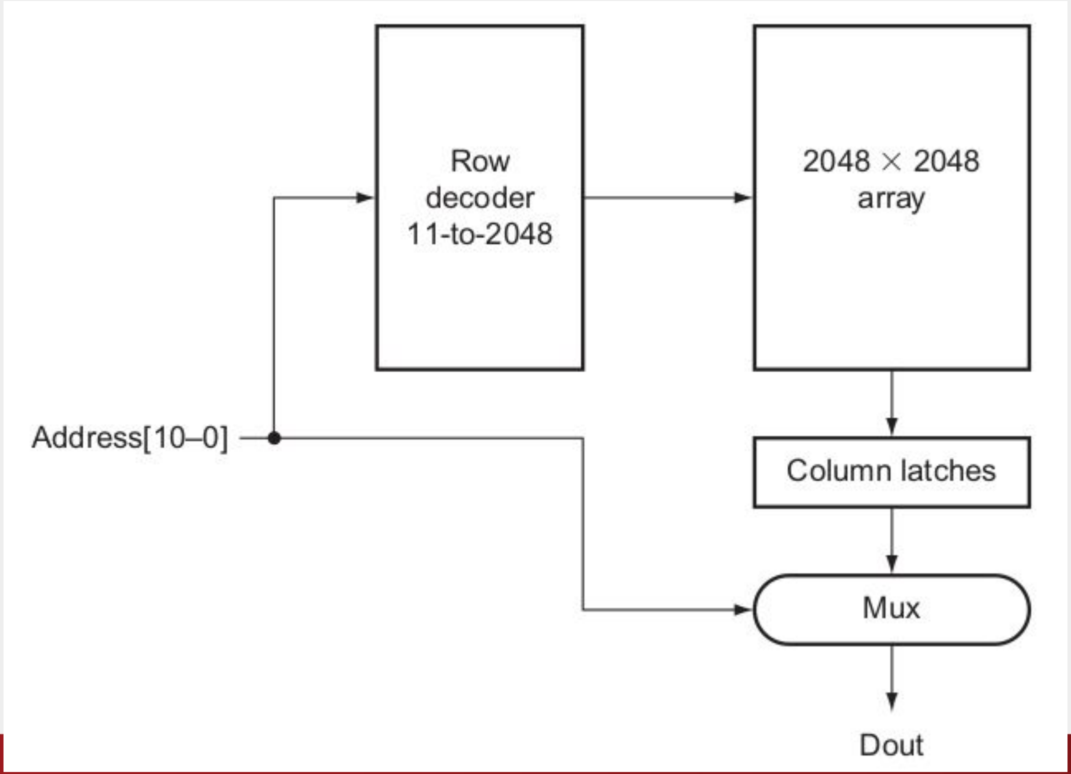


Dynamic RAM

- the value kept in a cell is stored as a charge in a capacitor
- A single transistor is then used to access this stored charge, either to read the value or to overwrite the charge stored there
- use only a single transistor per bit of storage - cheaper than SRAM

DRAM

- 4M × 1 DRAM is built with a 2048 × 2048 array
- Parity codes are used to detect memory error



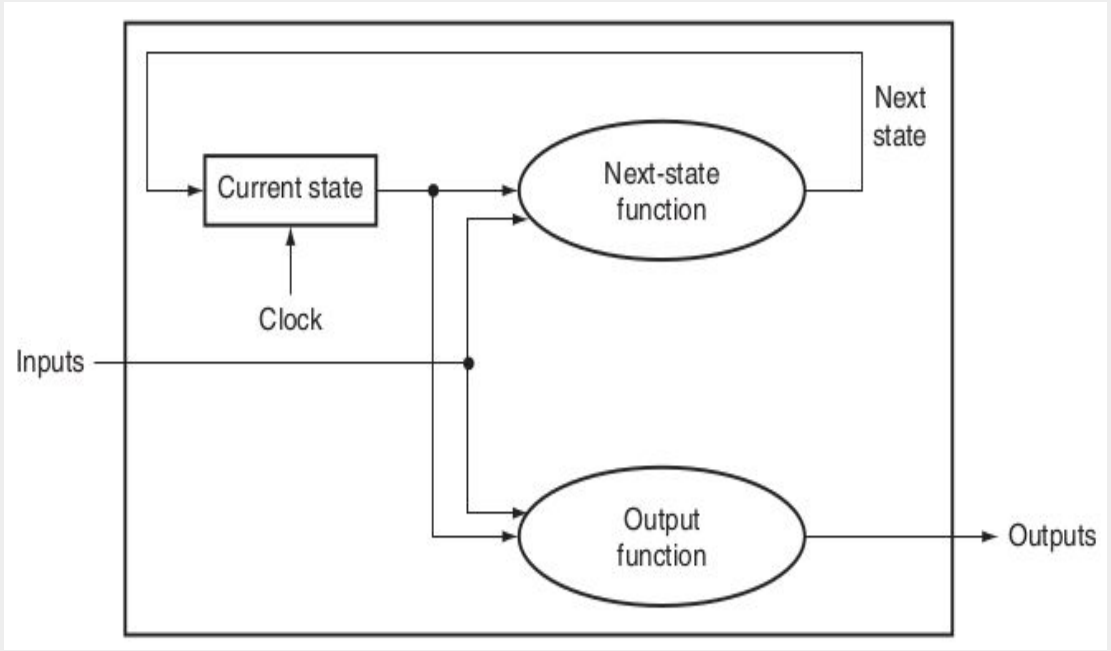
Finite State Machines

Finite State Machines

- a sequential system cannot be described with a truth table
- A finite-state machine has a set of states and two functions, called the next-state function and the output function
- The set of states corresponds to all the possible values of the internal storage
- The next-state function is a combinational function that, given the inputs and the current state, determines the next state of the system
- The output function produces a set of outputs from the current state and the inputs

Finite State Machines

- Moore machine - output function is dependent only on the current state
- Mealy machine - output function is dependent on current state and current input

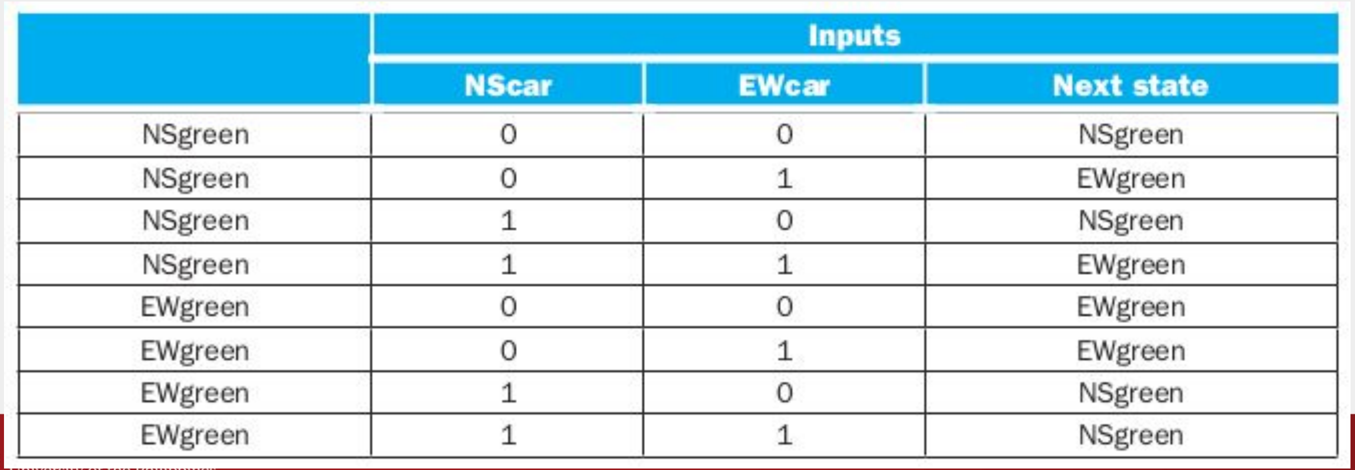


Example: Traffic Light

- Outputs
  - NSLite - When this signal is asserted, the light on the north-south road is green; when this signal is deasserted, the light on the north-south road is red
  - EWLite - When this signal is asserted, the light on the east-west road is green when this signal is deasserted, the light on the east-west road is red
- Inputs
  - NScar: Indicates that a car is over the detector placed in the roadbed in front of the light on the north-south road (going north or south).
  - EWcar: Indicates that a car is over the detector placed in the roadbed in front of the light on the east-west road (going east or west).

Example: Traffic Light

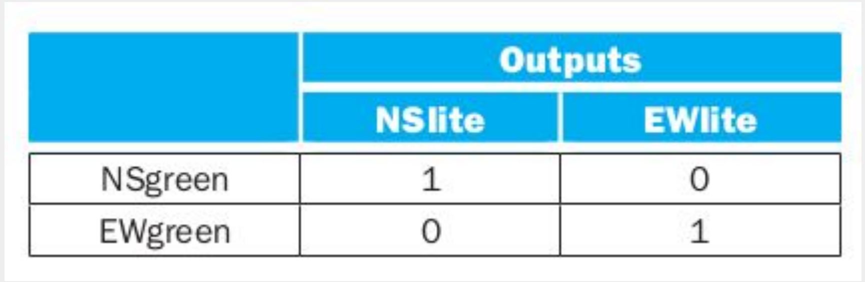
- States
  - NSgreen: The traffic light is green in the north-south direction.
  - EWgreen: The traffic light is green in the east-west direction.
- Next state function (user-defined)



	Inputs		
	NScar	EWcar	Next state
NSgreen	0	0	NSgreen
NSgreen	0	1	EWgreen
NSgreen	1	0	NSgreen
NSgreen	1	1	EWgreen
EWgreen	0	0	EWgreen
EWgreen	0	1	EWgreen
EWgreen	1	0	NSgreen
EWgreen	1	1	NSgreen

Example: Traffic Light

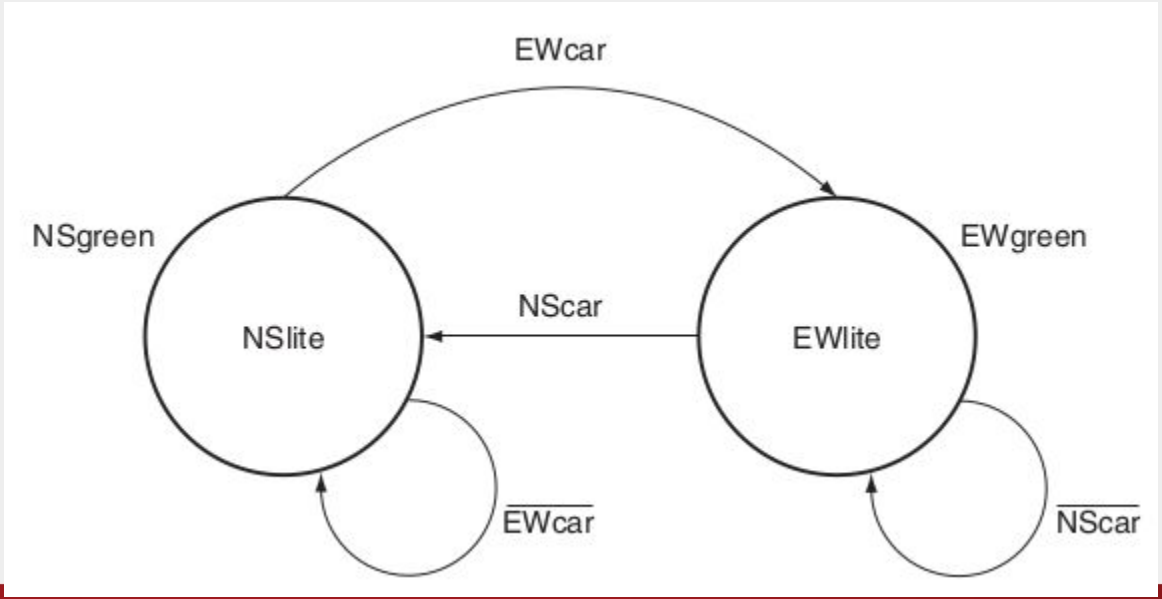
- Output function



	Outputs	
	NSlite	EWlite
NSgreen	1	0
EWgreen	0	1

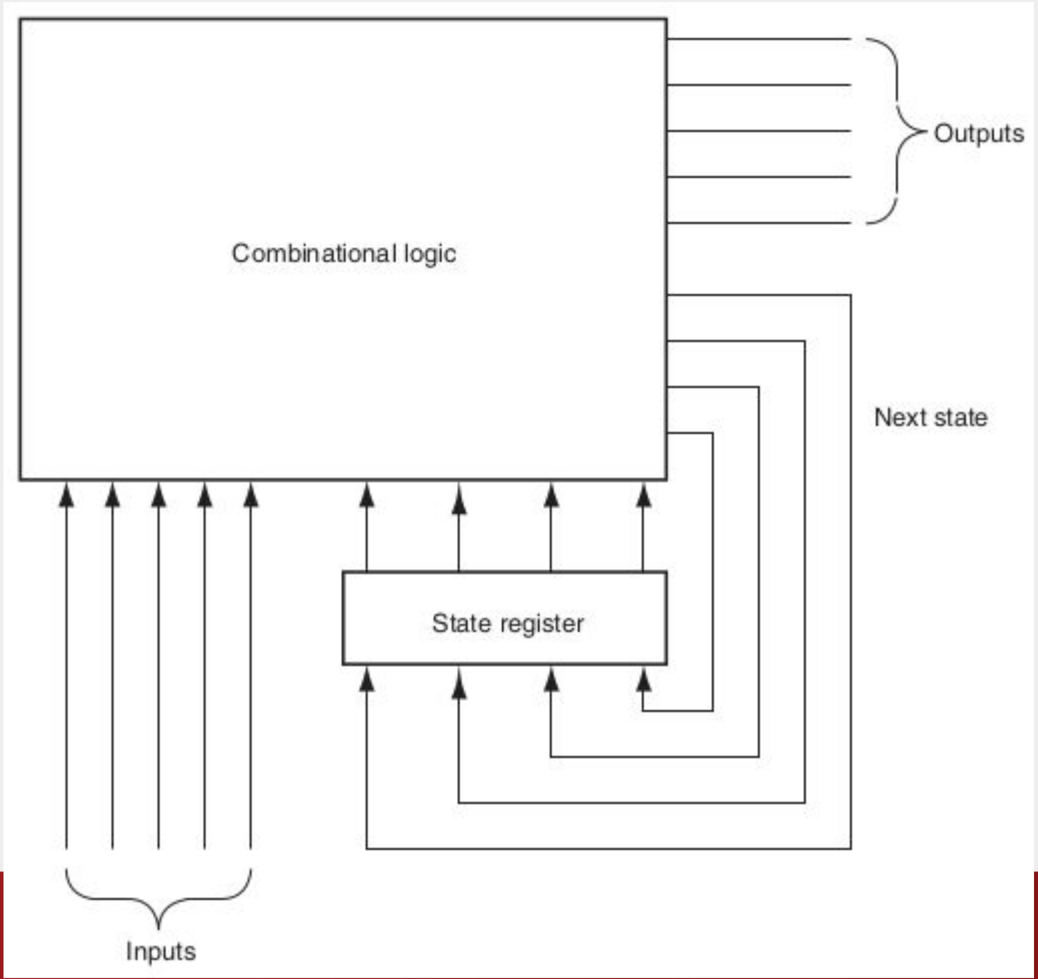
Example: Traffic Light

- Graphical Representation of State Transitions (simplified)



Example: Traffic Light

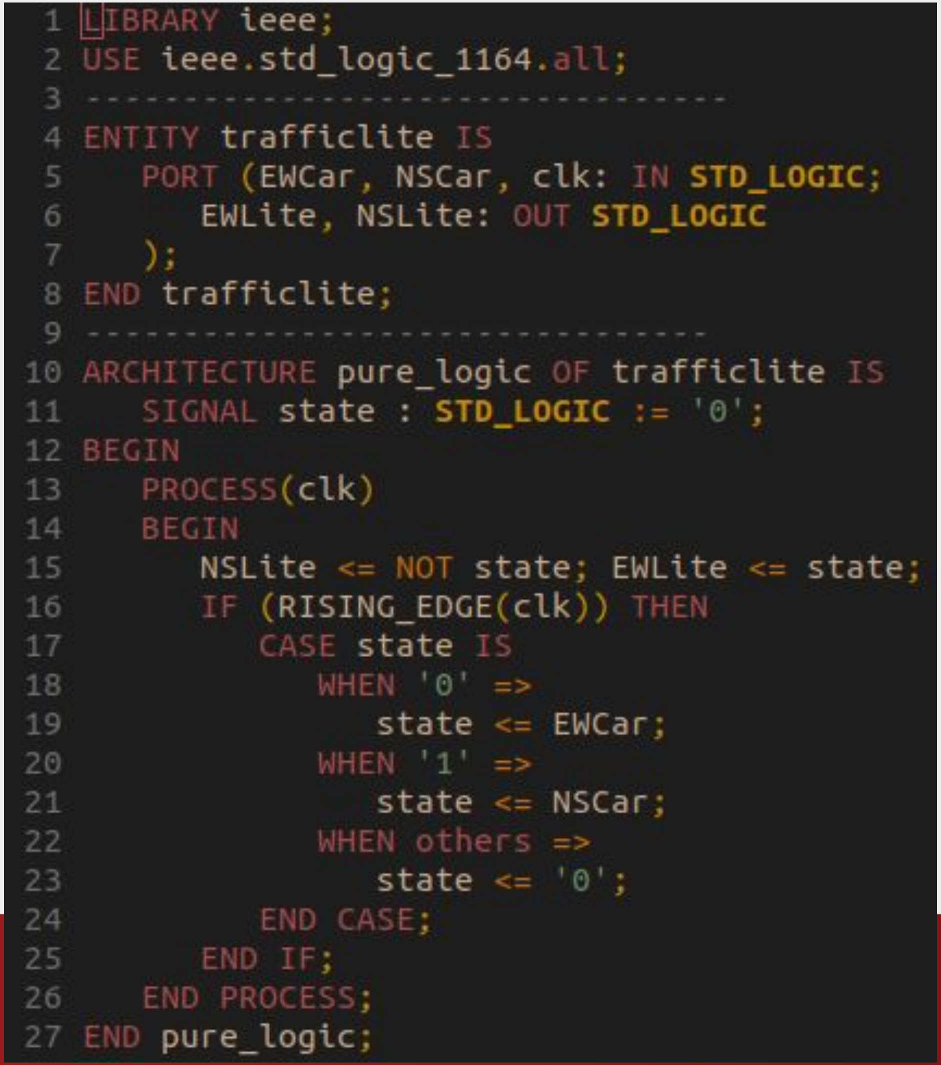
1. Assign number to states
2. NextState = (Not CurrentState AND EWCar) OR (CurrentState and NOT NSCar)
3. Output:
   a. NSLite = NOT CurrentState
   b. EWLite = CurrentState



University of the Philippines
LOS BAÑOS

VHDL Code

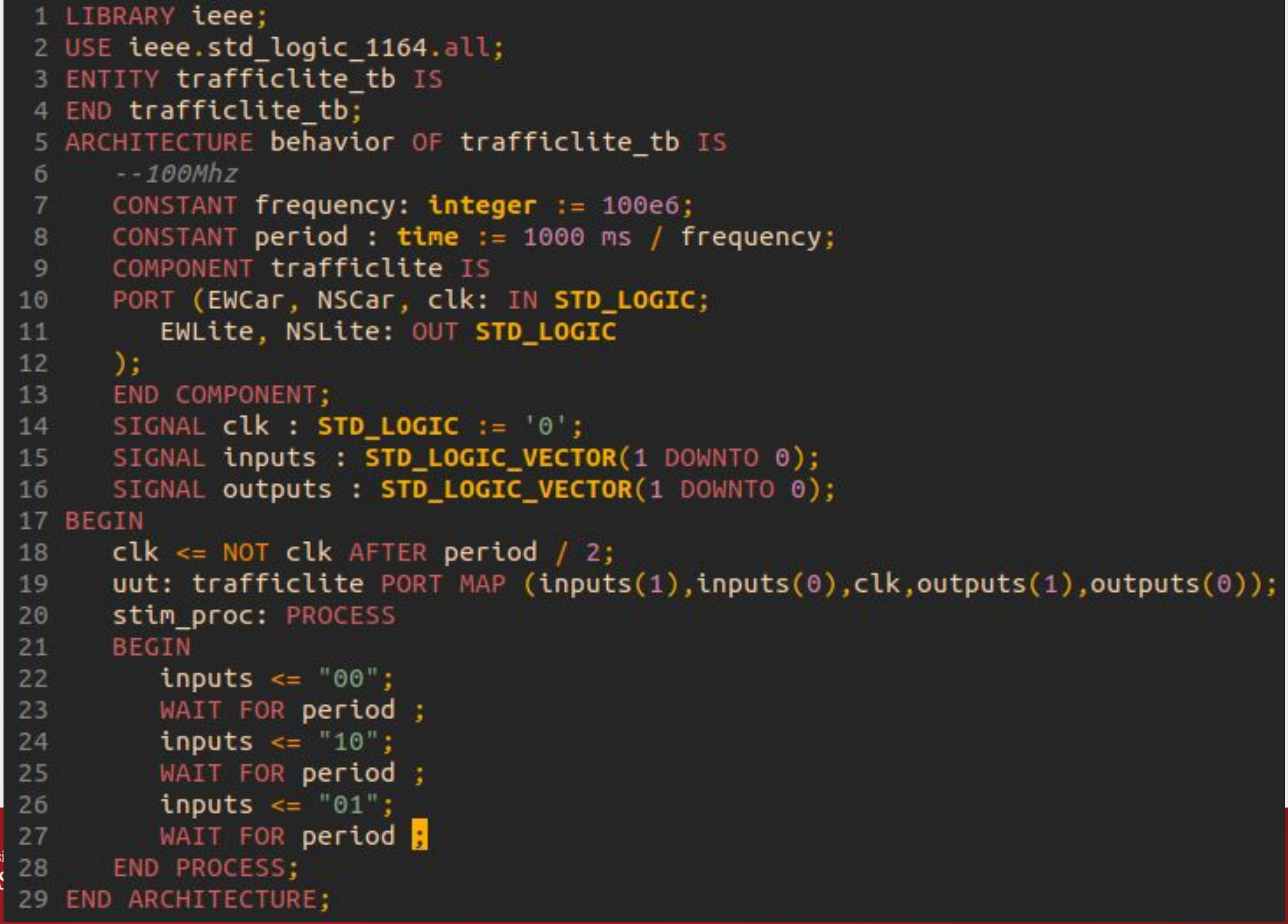
-



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------
ENTITY trafficlite IS
    PORT (EWCar, NSCar, clk: IN STD_LOGIC;
        EWLite, NSLite: OUT STD_LOGIC
    );
END trafficlite;
-------------------------------------
ARCHITECTURE pure_logic OF trafficlite IS
    SIGNAL state : STD_LOGIC := '0';
BEGIN
    PROCESS(clk)
    BEGIN
        NSLite <= NOT state; EWLite <= state;
        IF (RISING_EDGE(clk)) THEN
            CASE state IS
                WHEN '0' =>
                    state <= EWCar;
                WHEN '1' =>
                    state <= NSCar;
                WHEN others =>
                    state <= '0';
            END CASE;
        END IF;
    END PROCESS;
END pure_logic;
```

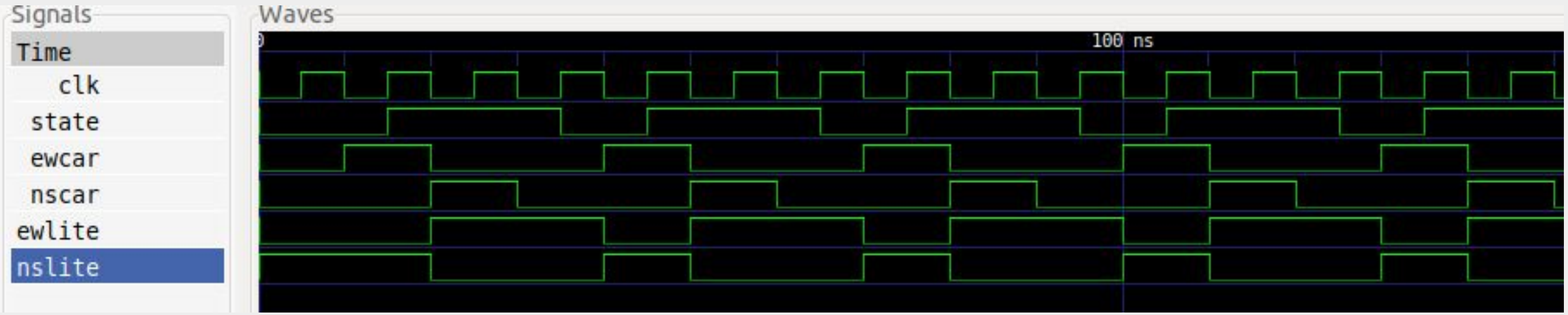
University of the Philippines
LOS BAÑOS

Testbench



```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY trafficlite_tb IS
END trafficlite_tb;
ARCHITECTURE behavior OF trafficlite_tb IS
    --100Mhz
    CONSTANT frequency: integer := 100e6;
    CONSTANT period : time := 1000 ms / frequency;
    COMPONENT trafficlite IS
    PORT (EWCar, NSCar, clk: IN STD_LOGIC;
        EWLite, NSLite: OUT STD_LOGIC
    );
    END COMPONENT;
    SIGNAL clk : STD_LOGIC := '0';
    SIGNAL inputs : STD_LOGIC_VECTOR(1 DOWNTO 0);
    SIGNAL outputs : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
    clk <= NOT clk AFTER period / 2;
    uut: trafficlite PORT MAP (inputs(1),inputs(0),clk,outputs(1),outputs(0));
    stim_proc: PROCESS
    BEGIN
        inputs <= "00";
        WAIT FOR period ;
        inputs <= "10";
        WAIT FOR period ;
        inputs <= "01";
        WAIT FOR period ;
    END PROCESS;
END ARCHITECTURE;
```

Waveform


Signals
Time
clk
state
ewcar
nscar
ewlite
nslite
Waves
0
100 ns

Enjoy! :)